```
**********************************************************
    1180 Mon Aug  5 08:37:45 2019
new/usr/src/Makefile.smatch
11506 smatch resync
**********************************************************
   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 2019 Joyent, Inc.
  12 #

  14 #
  15 # smatch/sparse checks we always disable, due to too many false positives (or
  16 # simply too much legacy).
  17 #

  19 SMATCH_ARGS = --disable=uninitialized,check_check_deref,unreachable

  21 # VLAs are OK by us
  22 SMATCH_ARGS += -Wno-vla
  23 # don't care
  24 SMATCH_ARGS += -Wno-one-bit-signed-bitfield
  25 # there are lots of "extern void myfunc() { ... }" around
  26 SMATCH_ARGS += -Wno-external-function-has-definition
  27 # we have lots of legacy "void foo();" in headers
  28 SMATCH_ARGS += -Wno-old-style-definition
  29 SMATCH_ARGS += -Wno-strict-prototypes
  30 SMATCH_ARGS += --fatal-checks
  31 SMATCH_ARGS += --timeout=0
  31 SMATCH_ARGS += --timeout=120

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

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

  37 SMATCH_ =
  38 SMATCH_on =
  39 SMATCH_off = -_smatch=off

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

```
**********************************************************
    7040 Mon Aug  5 08:37:45 2019
new/usr/src/boot/lib/libstand/Makefile.inc
11506 smatch resync
**********************************************************
#
# 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 2016 Toomas Soome <tsoome@me.com>
# Copyright 2019 Joyent, Inc.
#

#
# Notes:
# - We don't use the libc strerror/sys_errlist because the string table is
#   quite large.
#

# standalone components and stuff we have modified locally
SRCS=   $(ZLIB)/gzguts.h $(ZLIB)/zutil.h
SRCS += $(SASRC)/__main.c $(SASRC)/assert.c
SRCS += $(SASRC)/bcd.c $(SASRC)/environment.c
SRCS += $(SASRC)/getopt.c $(SASRC)/strtol.c
SRCS += $(SASRC)/strtoul.c $(SASRC)/random.c
SRCS += $(SASRC)/sbrk.c $(SASRC)/twiddle.c
SRCS += $(SASRC)/zalloc.c $(SASRC)/zalloc_malloc.c

OBJS=   __main.o assert.o bcd.o environment.o \
        getopt.o gets.o globals.o pager.o panic.o printf.o \
        strdup.o strerror.o strtol.o strtoul.o random.o \
        sbrk.o twiddle.o zalloc.o zalloc_malloc.o

# private (pruned) versions of libc string functions
SRCS += $(SASRC)/strcasecmp.c
OBJS += strcasecmp.o

# from libc
SRCS += $(LIBSRC)/libc/net/ntoh.c
OBJS += ntoh.o

# string functions from libc
SRCS += $(LIBSRC)/libc/string/bcmp.c $(LIBSRC)/libc/string/bcopy.c
SRCS += $(LIBSRC)/libc/string/bzero.c $(LIBSRC)/libc/string/ffs.c
SRCS += $(LIBSRC)/libc/string/fls.c $(LIBSRC)/libc/string/memccpy.c
SRCS += $(LIBSRC)/libc/string/memchr.c $(LIBSRC)/libc/string/memcmp.c
SRCS += $(LIBSRC)/libc/string/memcpy.c $(LIBSRC)/libc/string/memmove.c
SRCS += $(LIBSRC)/libc/string/memset.c $(LIBSRC)/libc/string/strcat.c
SRCS += $(LIBSRC)/libc/string/strchr.c $(LIBSRC)/libc/string/strcmp.c
SRCS += $(LIBSRC)/libc/string/strcpy.c $(LIBSRC)/libc/string/stpcpy.c
SRCS += $(LIBSRC)/libc/string/stpncpy.c $(LIBSRC)/libc/string/strcspn.c
SRCS += $(LIBSRC)/libc/string/strlcat.c $(LIBSRC)/libc/string/strlcpy.c
SRCS += $(LIBSRC)/libc/string/strlen.c $(LIBSRC)/libc/string/strncat.c
SRCS += $(LIBSRC)/libc/string/strncmp.c $(LIBSRC)/libc/string/strncpy.c
SRCS += $(LIBSRC)/libc/string/strpbrk.c $(LIBSRC)/libc/string/strrchr.c
SRCS += $(LIBSRC)/libc/string/strsep.c $(LIBSRC)/libc/string/strspn.c
SRCS += $(LIBSRC)/libc/string/strstr.c $(LIBSRC)/libc/string/strtok.c
SRCS += $(LIBSRC)/libc/string/swab.c
```



```
SRCS += $(SASRC)/qdivrem.c

OBJS += bcmp.o bcopy.o bzero.o ffs.o fls.o \
        memccpy.o memchr.o memcmp.o memcpy.o memmove.o memset.o \
        qdivrem.o strcat.o strchr.o strcmp.o strcpy.o stpcpy.o stpncpy.o \
        strcspn.o strlcat.o strlcpy.o strlen.o strncat.o strncmp.o strncpy.o \
        strpbrk.o strrchr.o strsep.o strspn.o strstr.o strtok.o swab.o

# uuid functions from libc
SRCS += $(LIBSRC)/libc/uuid/uuid_create_nil.c
SRCS += $(LIBSRC)/libc/uuid/uuid_equal.c
SRCS += $(LIBSRC)/libc/uuid/uuid_is_nil.c

SRCS += $(SASRC)/uuid_from_string.c
SRCS += $(SASRC)/uuid_to_string.c

OBJS += uuid_create_nil.o uuid_equal.o uuid_from_string.o uuid_is_nil.o \
        uuid_to_string.o

# decompression functionality from libbz2
# NOTE: to actually test this functionality after libbz2 upgrade compile
# loader(8) with LOADER_BZIP2_SUPPORT defined
_bzlib.o _crctable.o _decompress.o _huffman.o _randtable.o bzipfs.o \
:= CFLAGS += -DBZ_LOADER -DBZ_NO_STDIO -DBZ_NO_COMPRESS
SRCS += libstand_bzlib_private.h

# too hairy
_inflate.o := SMATCH=off

SRCS += _bzlib.c _crctable.c _decompress.c _huffman.c _randtable.c
OBJS += _bzlib.o _crctable.o _decompress.o _huffman.o _randtable.o
CLEANFILES +=   _bzlib.c _crctable.c _decompress.c _huffman.c _randtable.c

_bzlib.c: $(SRC)/common/bzip2/bzlib.c
        sed "s|bzlib_private\.h|libstand_bzlib_private.h|" $^ > $@

_crctable.c: $(SRC)/common/bzip2/crctable.c
        sed "s|bzlib_private\.h|libstand_bzlib_private.h|" $^ > $@

_decompress.c: $(SRC)/common/bzip2/decompress.c
        sed "s|bzlib_private\.h|libstand_bzlib_private.h|" $^ > $@

_huffman.c: $(SRC)/common/bzip2/huffman.c
        sed "s|bzlib_private\.h|libstand_bzlib_private.h|" $^ > $@

_randtable.c: $(SRC)/common/bzip2/randtable.c
        sed "s|bzlib_private\.h|libstand_bzlib_private.h|" $^ > $@

CLEANFILES += libstand_bzlib_private.h
libstand_bzlib_private.h: $(SRC)/common/bzip2/bzlib_private.h
        sed -e 's|<stdlib.h>|"stand.h"|' $^ > $@

# decompression functionality from zlib
adler32.o crc32.o _infback.o _inffast.o _inflate.o _inftrees.o _zutil.o \
gzipfs.o gzip.o := CPPFLAGS += -I$(ZLIB)
SRCS += $(ZLIB)/adler32.c $(ZLIB)/crc32.c \
        libstand_zutil.h libstand_gzguts.h
OBJS += adler32.o crc32.o

_infback.c: $(ZLIB)/infback.c
        sed -e "s|zutil\.h|libstand_zutil.h|" \
            -e "s|gzguts\.h|libstand_gzguts.h|" \
            $^ > $@
_inffast.c: $(ZLIB)/inffast.c
        sed -e "s|zutil\.h|libstand_zutil.h|" \
```

```
128             -e "s|gzguts\.h|libstand_gzguts.h|" \
129             $^ > $@
130 _inflate.c: $(ZLIB)/inflate.c
131         sed -e "s|zutil\.h|libstand_zutil.h|" \
132             -e "s|gzguts\.h|libstand_gzguts.h|" \
133             $^ > $@
134 _inftrees.c: $(ZLIB)/inftrees.c
135         sed -e "s|zutil\.h|libstand_zutil.h|" \
136             -e "s|gzguts\.h|libstand_gzguts.h|" \
137             $^ > $@
138 _zutil.c: $(ZLIB)/zutil.c
139         sed -e "s|zutil\.h|libstand_zutil.h|" \
140             -e "s|gzguts\.h|libstand_gzguts.h|" \
141             $^ > $@

143 SRCS += _infback.c _inffast.c _inflate.c _inftrees.c _zutil.c
144 OBJS += _infback.o _inffast.o _inflate.o _inftrees.o _zutil.o
145 CLEANFILES +=   _infback.c _inffast.c _inflate.c _inftrees.c _zutil.c

147 # depend on stand.h being able to be included multiple times
148 libstand_zutil.h: $(ZLIB)/zutil.h
149         sed -e 's|<fcntl.h>|"stand.h"|' \
150             -e 's|<stddef.h>|"stand.h"|' \
151             -e 's|<string.h>|"stand.h"|' \
152             -e 's|<stdio.h>|"stand.h"|' \
153             -e 's|<stdlib.h>|"stand.h"|' \
154             $^ > $@

156 libstand_gzguts.h: $(ZLIB)/gzguts.h
157         sed -e 's|<fcntl.h>|"stand.h"|' \
158             -e 's|<stddef.h>|"stand.h"|' \
159             -e 's|<string.h>|"stand.h"|' \
160             -e 's|<stdio.h>|"stand.h"|' \
161             -e 's|<stdlib.h>|"stand.h"|' \
162             $^ > $@

164 CLEANFILES += libstand_zutil.h libstand_gzguts.h

166 # io routines
167 SRCS += $(SASRC)/closeall.c $(SASRC)/dev.c \
168         $(SASRC)/ioctl.c $(SASRC)/nullfs.c \
169         $(SASRC)/stat.c $(SASRC)/fstat.c $(SASRC)/close.c \
170         $(SASRC)/lseek.c $(SASRC)/open.c $(SASRC)/read.c \
171         $(SASRC)/write.c $(SASRC)/readdir.c

173 OBJS += closeall.o dev.o ioctl.o nullfs.o stat.o fstat.o close.o lseek.o \
174         open.o read.o write.o readdir.o

176 # network routines
177 SRCS += $(SASRC)/arp.c $(SASRC)/ether.c $(SASRC)/ip.c \
178         $(SASRC)/inet_ntoa.c $(SASRC)/in_cksum.c \
179         $(SASRC)/net.c $(SASRC)/udp.c $(SASRC)/netif.c \
180         $(SASRC)/rpc.c
181 OBJS += arp.o ether.o ip.o inet_ntoa.o in_cksum.o net.o udp.o netif.o rpc.o

183 # network info services:
184 SRCS += $(SASRC)/bootp.c $(SASRC)/rarp.c \
185         $(SASRC)/bootparam.c
186 OBJS += bootp.o rarp.o bootparam.o

188 # boot filesystems
189 SRCS += $(SASRC)/ufs.c
190 SRCS += $(SASRC)/nfs.c
191 SRCS += $(SASRC)/cd9660.c
192 SRCS += $(SASRC)/tftp.c
193 SRCS += $(SASRC)/gzipfs.c
```



```
194 SRCS += $(SASRC)/bzipfs.c
195 SRCS += $(SASRC)/dosfs.c
196 OBJS += ufs.o
197 OBJS += nfs.o
198 OBJS += cd9660.o
199 OBJS += tftp.o
200 OBJS += gzipfs.o
201 OBJS += bzipfs.o
202 OBJS += dosfs.o
203 #
204 .PARALLEL:
```

```
**********************************************************
    2493 Mon Aug  5 08:37:46 2019
new/usr/src/boot/sys/boot/Makefile.inc
11506 smatch resync
**********************************************************
```

```
#
# 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 2017 Toomas Soome <tsoome@me.com>
# Copyright 2019 Joyent, Inc.
#

# loader.help build needs better awk
AWK=            /usr/xpg4/bin/awk
LD=             $(GNU_ROOT)/bin/gld
OBJCOPY=        $(GNU_ROOT)/bin/gobjcopy
OBJDUMP=        $(GNU_ROOT)/bin/gobjdump
GSTRIP=         $(GNU_ROOT)/bin/gstrip

# Default Console font setup.
# We want it to be the same as kernel.
# We build compressed, stripped down version of the default font, so we have
# bare minimum for case we can not load font from the OS root.

FONT=           8x16
FONT_SRC=       ter-u16n.bdf
FONT_DIR=       $(SRC)/data/consfonts

PNGLITE=        $(SRC)/common/pnglite

BOOTSRC=        $(SRC)/boot/sys/boot
LIBSRC=         $(SRC)/boot/lib
SASRC=          $(LIBSRC)/libstand
ZFSSRC=         $(SASRC)/zfs
ZLIB=           $(SRC)/contrib/zlib

# set standard values
AS_CPPFLAGS=
CPPFLAGS=       -D_STANDALONE -_gcc=-nostdinc
CFLAGS64=       -_gcc=-mno-red-zone

CFLAGS=         -_gcc=-Os -_gcc=-fPIC -_gcc=-ffreestanding -_gcc=-fno-builtin
CFLAGS +=       -_gcc=-ffunction-sections -_gcc=-fdata-sections
CFLAGS +=       -_gcc=-mno-mmx -_gcc=-mno-3dnow -_gcc=-mno-sse -_gcc=-mno-sse2
CFLAGS +=       -_gcc=-mno-sse3 -_gcc=-msoft-float
CFLAGS +=       -_gcc=-mno-avx -_gcc=-mno-aes
CFLAGS +=       -_gcc=-Wall
CFLAGS +=       $(CCNOAUTOINLINE) $(CCNOREORDER) $(CSTD_GNU99)
CCASFLAGS=      -fPIC -Wa,--divide
ASFLAGS=        --divide

SMATCH_ =
SMATCH_on =
SMATCH_off = -_smatch=off

# smatch does not define __amd64 and __amd64__
SMATCH_amd64=   -_smatch=-D__amd64 -_smatch=-D__amd64__
```



```
# SMATCH_ARGS will bring in set of -Wno-* options.
#CFLAGS +=      $(SMATCH_ARGS:%=-_smatch=%)
CFLAGS +=       $(SMOFF:%=-_smatch=--disable=%)
CFLAGS +=       $(SMATCH_$(MACHINE))
CFLAGS +=       $(SMATCH_$(SMATCH))
CFLAGS +=       -_smatch=--timeout=0

COMPILE.S=      $(CC) $(SMATCH_off) $(CCASFLAGS) $(CPPFLAGS) -c

ROOT_BOOT=      $(ROOT)/boot
ROOTBOOTPROG=$(PROG:%=$(ROOT_BOOT)/%)

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

#.if ${MACHINE_CPUARCH} == "arm"
# Do not generate movt/movw, because the relocation fixup for them does not
# translate to the -Bsymbolic -pie format required by self_reloc() in loader(8).
# Also, the fpu is not available in a standalone environment.
#CFLAGS.clang+=  -mllvm -arm-use-movt=0
#CFLAGS.clang+=  -mfpu=none
#.endif
```

****
721 Mon Aug 5 08:37:47 2019
new/usr/src/boot/sys/boot/efi/libefi/i386/Makefile
11506 smatch resync
****

```
   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 #

  12 #
  13 # Copyright 2016 Toomas Soome <tsoome@me.com>
  14 # Copyright 2016 RackTop Systems.
  15 # Copyright 2019 Joyent, Inc.
  16 #

  18 MACHINE=        $(MACH)

  20 all:            libefi.a

  22 SRCS=   time.c
  23 include ../Makefile.com

  25 CFLAGS +=       -m32

  27 # false positive only with a 64-bit smatch
  28 SMOFF += uninitialized

  30 CLEANFILES +=   machine x86

  32 $(OBJS): machine x86
```

```
**********************************************************
    1361 Mon Aug  5 08:37:48 2019
new/usr/src/boot/sys/boot/libstand/Makefile.com
11506 smatch resync
**********************************************************
#
# 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 2016 Toomas Soome <tsoome@me.com>
# Copyright 2019 Joyent, Inc.
#

include $(SRC)/Makefile.master
include $(SRC)/boot/sys/boot/Makefile.inc

CPPFLAGS +=     -I../../../../include -I$(SASRC)
CPPFLAGS +=     -I../../.. -I. -I$(SRC)/common/bzip2

$(LIBRARY): $(SRCS) $(OBJS)
        $(AR) $(ARFLAGS) $@ $(OBJS)

include $(SASRC)/Makefile.inc
include $(ZFSSRC)/Makefile.inc

CPPFLAGS +=     -I$(SRC)/uts/common

# needs work
printf.o := SMOFF += 64bit_shift

# too hairy
_inflate.o := SMATCH=off

# 64-bit smatch false positive :/
SMOFF += uninitialized

clean: clobber
clobber:
        $(RM) $(CLEANFILES) $(OBJS) machine $(LIBRARY)

machine:
        $(RM) machine
        $(SYMLINK) ../../../$(MACHINE)/include machine

x86:
        $(RM) x86
        $(SYMLINK) ../../../x86/include x86

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

%.o:    $(LIBSRC)/libc/net/%.c
        $(COMPILE.c) $<

%.o:    $(LIBSRC)/libc/string/%.c
        $(COMPILE.c) $<

%.o:    $(LIBSRC)/libc/uuid/%.c
```



```
        $(COMPILE.c) $<

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

```
**********************************************************
    1660 Mon Aug  5 08:37:48 2019
new/usr/src/cmd/ls/Makefile.com
11506 smatch resync
**********************************************************
```

```
#
# 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 2019 Joyent, Inc.
# cmd/ls/Makefile.com
#

PROG=           ls
XPG4PROG=       ls
XPG6PROG=       ls
OBJS=           $(PROG).o
SRCS=           $(OBJS:%.o=../%.c)

include ../../Makefile.cmd

LDLIBS += -lsec -lnvpair -lcmdutils -lcurses
CFLAGS  +=      $(CCVERBOSE)
$(XPG4) := CFLAGS += -DXPG4

# Include all XPG4 changes in the XPG6 version
$(XPG6) := CFLAGS += -DXPG4 -DXPG6
$(XPG6) := CFLAGS64 += -DXPG4 -DXPG6

CFLAGS64 +=     $(CCVERBOSE)
CPPFLAGS += -D_FILE_OFFSET_BITS=64
LINTFLAGS64 +=  -errchk=longptr64

# main() can be too hairy
SMATCH=off

.KEEP_STATE:

all:    $(PROG) $(XPG4) $(XPG6)

lint:   lint_SRCS

clean:
        $(RM) $(CLEANFILES)

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



```
%.xpg4: ../%.c
        $(LINK.c) -o $@ $< $(LDLIBS)
        $(POST_PROCESS)

%.xpg6: ../%.c
        $(LINK.c) -o $@ $< $(LDLIBS)
        $(POST_PROCESS)

%: ../%.c
        $(LINK.c) -o $@ $< $(LDLIBS)
        $(POST_PROCESS)
```

```
**********************************************************
    4894 Mon Aug  5 08:37:49 2019
new/usr/src/cmd/sgs/libld/Makefile.com
11506 smatch resync
**********************************************************
#
# 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 2010 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#
# Copyright 2019 Joyent, Inc.
# Copyright (c) 2018, Joyent, Inc.
# Copyright 2019 OmniOS Community Edition (OmniOSce) Association.

LIBRARY =       libld.a
VERS =          .4

COMOBJS =       debug.o         globals.o       util.o

COMOBJS32 =     args32.o        entry32.o       exit32.o        groups32.o \
                ldentry32.o     ldlibs32.o      ldmachdep32.o   ldmain32.o \
                libs32.o        files32.o       map32.o         map_core32.o \
                map_support32.o map_v232.o      order32.o       outfile32.o \
                place32.o       relocate32.o    resolve32.o     sections32.o \
                sunwmove32.o    support32.o     syms32.o        update32.o \
                unwind32.o      version32.o     wrap32.o

COMOBJS64 =     args64.o        entry64.o       exit64.o        groups64.o \
                ldentry64.o     ldlibs64.o      ldmachdep64.o   ldmain64.o \
                libs64.o        files64.o       map64.o         map_core64.o \
                map_support64.o map_v264.o      order64.o       outfile64.o \
                place64.o       relocate64.o    resolve64.o     sections64.o \
                sunwmove64.o    support64.o     syms64.o        update64.o \
                unwind64.o      version64.o     wrap64.o

TOOLOBJS =      alist.o         assfail.o       findprime.o     string_table.o \
                strhash.o
AVLOBJ =        avl.o

# Relocation engine objects.
G_MACHOBJS32 =  doreloc_sparc_32.o doreloc_x86_32.o
G_MACHOBJS64 =  doreloc_sparc_64.o doreloc_x86_64.o

# Target specific objects (sparc/sparcv9)
L_SPARC_MACHOBJS32 =    machrel.sparc32.o       machsym.sparc32.o
L_SPARC_MACHOBJS64 =    machrel.sparc64.o       machsym.sparc64.o
```



```
# Target specific objects (i386/amd64)
E_X86_TOOLOBJS =        leb128.o
L_X86_MACHOBJS32 =      machrel.intel32.o
L_X86_MACHOBJS64 =      machrel.amd64.o

# All target specific objects rolled together
E_TOOLOBJS =    $(E_SPARC_TOOLOBJS) \
        $(E_X86_TOOLOBJS)
L_MACHOBJS32 =  $(L_SPARC_MACHOBJS32) \
        $(L_X86_MACHOBJS32)
L_MACHOBJS64 =  $(L_SPARC_MACHOBJS64) \
        $(L_X86_MACHOBJS64)


BLTOBJ =        msg.o
ELFCAPOBJ =     elfcap.o

OBJECTS =       $(BLTOBJ) $(G_MACHOBJS32) $(G_MACHOBJS64) \
                $(L_MACHOBJS32) $(L_MACHOBJS64) \
                $(COMOBJS) $(COMOBJS32) $(COMOBJS64) \
                $(TOOLOBJS) $(E_TOOLOBJS) $(AVLOBJ) $(ELFCAPOBJ)

include         $(SRC)/lib/Makefile.lib
include         $(SRC)/cmd/sgs/Makefile.com

SRCDIR =        ../common

CERRWARN += -_gcc=-Wno-unused-value
CERRWARN += -_gcc=-Wno-parentheses
CERRWARN += -_gcc=-Wno-uninitialized
CERRWARN += -_gcc=-Wno-switch
CERRWARN += -_gcc=-Wno-char-subscripts
CERRWARN += -_gcc=-Wno-type-limits
$(RELEASE_BUILD)CERRWARN += -_gcc=-Wno-unused

SMOFF += no_if_block

# Location of the shared relocation engines maintained under usr/src/uts.
#
KRTLD_I386 = $(SRCBASE)/uts/$(VAR_PLAT_i386)/krtld
KRTLD_AMD64 = $(SRCBASE)/uts/$(VAR_PLAT_amd64)/krtld
KRTLD_SPARC = $(SRCBASE)/uts/$(VAR_PLAT_sparc)/krtld


CPPFLAGS +=     -DUSE_LIBLD_MALLOC -I$(SRCBASE)/lib/libc/inc \
                    -I$(SRCBASE)/uts/common/krtld -I$(SRCBASE)/uts/sparc \
                    $(VAR_LIBLD_CPPFLAGS)
LDLIBS +=       $(CONVLIBDIR) $(CONV_LIB) $(LDDBGLIBDIR) $(LDDBG_LIB) \
                    $(ELFLIBDIR) -lelf $(DLLIB) -lc

DYNFLAGS +=     $(VERSREF) $(CC_USE_PROTO) '-R$$ORIGIN'

native:=        DYNFLAGS        += $(CONVLIBDIR)

# too hairy
pics/sections32.o := SMATCH=off
pics/sections64.o := SMATCH=off

BLTDEFS =       msg.h
BLTDATA =       msg.c
BLTMESG =       $(SGSMSGDIR)/libld

BLTFILES =      $(BLTDEFS) $(BLTDATA) $(BLTMESG)

# Due to cross linking support, every copy of libld contains every message.
```

```
127 # However, we keep target specific messages in their own separate files for
128 # organizational reasons.
129 #
130 SGSMSGCOM =     ../common/libld.msg
131 SGSMSGSPARC =   ../common/libld.sparc.msg
132 SGSMSGINTEL =   ../common/libld.intel.msg
133 SGSMSGTARG =    $(SGSMSGCOM) $(SGSMSGSPARC) $(SGSMSGINTEL)
134 SGSMSGALL =     $(SGSMSGCOM) $(SGSMSGSPARC) $(SGSMSGINTEL)

136 SGSMSGFLAGS1 =  $(SGSMSGFLAGS) -m $(BLTMESG)
137 SGSMSGFLAGS2 =  $(SGSMSGFLAGS) -h $(BLTDEFS) -d $(BLTDATA) -n libld_msg

139 CHKSRCS =       $(SRCBASE)/uts/common/krtld/reloc.h \
140                 $(COMOBJS32:%32.o=../common/%.c) \
141                 $(L_MACHOBJS32:%32.o=../common/%.c) \
142                 $(L_MACHOBJS64:%64.o=../common/%.c) \
143                 $(KRTLD_I386)/doreloc.c \
144                 $(KRTLD_AMD64)/doreloc.c \
145                 $(KRTLD_SPARC)/doreloc.c

147 LIBSRCS =       $(TOOLOBJS:%.o=$(SGSTOOLS)/common/%.c) \
148                 $(E_TOOLOBJS:%.o=$(SGSTOOLS)/common/%.c) \
149                 $(COMOBJS:%.o=../common/%.c) \
150                 $(AVLOBJS:%.o=$(VAR_AVLDIR)/%.c) \
151                 $(BLTDATA)

153 CLEANFILES +=   $(BLTFILES)
154 CLOBBERFILES += $(DYNLIB) $(LIBLINKS)

156 ROOTFS_DYNLIB = $(DYNLIB:%=$(ROOTFS_LIBDIR)/%)
```

```
**********************************************************
    3217 Mon Aug  5 08:37:50 2019
new/usr/src/cmd/svc/configd/Makefile
11506 smatch resync
**********************************************************
#
# 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 2015 RackTop Systems.
#
# Copyright 2019 Joyent, Inc.
#

MYPROG = svc.configd
MYOBJS = \
        backend.o \
        configd.o \
        client.o \
        file_object.o \
        maindoor.o \
        object.o \
        rc_node.o \
        snapshot.o

PROG = $(MYPROG)
OBJS = $(MYOBJS)

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

include ../../Makefile.cmd
include ../../Makefile.ctf

NATIVE_BUILD=$(POUND_SIGN)
$(NATIVE_BUILD)PROG = $(MYPROG:%=%-native)
$(NATIVE_BUILD)OBJS = $(MYOBJS:%.o=%-native.o)

ROOTCMDDIR=     $(ROOT)/lib/svc/bin

MYCPPFLAGS = -I. -I../common -I../../../common/svc \
        -I$(ROOT)/usr/include/sqlite-sys -D_REENTRANT
CPPFLAGS += $(MYCPPFLAGS)
CFLAGS  += $(CCVERBOSE)
CERRWARN += -_gcc=-Wno-parentheses
CERRWARN += -_gcc=-Wno-type-limits
CERRWARN += -_gcc=-Wno-unused-label
```



```
CERRWARN += -_gcc=-Wno-unused-variable
CERRWARN += -_gcc=-Wno-unused-function
CERRWARN += -_gcc=-Wno-uninitialized

# strange false positive
SMOFF += free

MYLDLIBS = -lumem -luutil
LDLIBS  += -lsecdb -lbsm $(MYLDLIBS)
LINTFLAGS += -errtags -erroff=E_BAD_FORMAT_ARG_TYPE2 -erroff=E_NAME_DEF_NOT_USED

CLOBBERFILES += $(MYPROG:%=%-native)

LIBUUTIL        = $(SRC)/lib/libuutil
LIBSCF          = $(SRC)/lib/libscf

SCRIPTFILE      = restore_repository
ROOTSCRIPTFILE  = $(ROOTCMDDIR)/$(SCRIPTFILE)

#
# Native variant (used in ../seed)
#
$(NATIVE_BUILD)CC =     $(NATIVECC)
$(NATIVE_BUILD)LD =     $(NATIVELD)
$(NATIVE_BUILD)CFLAGS = $(NATIVE_CFLAGS)
$(NATIVE_BUILD)CPPFLAGS = $(MYCPPFLAGS) -I$(LIBUUTIL)/common -I$(LIBSCF)/inc
$(NATIVE_BUILD)CPPFLAGS += -DNATIVE_BUILD
$(NATIVE_BUILD)LDFLAGS =
$(NATIVE_BUILD)LDLIBS = -L$(ADJUNCT_PROTO)/usr/lib -R$(ADJUNCT_PROTO)/usr/lib \
                        -L$(LIBUUTIL)/native -R $(LIBUUTIL)/native $(MYLDLIBS)

DIRMODE = 0755
FILEMODE = 0555

OBJSQLITE =
LIBSQLITE = -lsqlite-sys
$(NATIVE_BUILD)OBJSQLITE = $(ROOT)/lib/libsqlite-native.o
$(NATIVE_BUILD)LIBSQLITE =

OBJS += $(OBJSQLITE)
LDLIBS += $(LIBSQLITE)

install := TARGET = install
clobber := TARGET = clobber

.KEEP_STATE:
.PARALLEL: $(MYOBJS) $(MYOBJS:%.o=%-native.o)

all: $(PROG)

native: FRC
        @cd $(LIBUUTIL)/native; pwd; $(MAKE) $(MFLAGS) install
        @NATIVE_BUILD= $(MAKE) $(MFLAGS) all

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

%-native.o: %.c
        $(COMPILE.c) -o $@ $<
        $(POST_PROCESS_O)

$(ROOTCMDDIR)/%: %.sh
        $(INS.rename)

install: all $(ROOTCMD) $(ROOTVARSADMFILE) $(ROOTSCRIPTFILE)
```

```
128 clean: FRC
129         $(RM) $(MYOBJS) $(MYOBJS:%.o=%-native.o)

131 clobber:

128 lint:   lint_SRCS

130 lint_SRCS:

133 include ../../Makefile.targ

135 FRC:
```

```
**********************************************************
    1291 Mon Aug  5 08:37:50 2019
new/usr/src/cmd/syseventd/modules/sysevent_conf_mod/Makefile
11506 smatch resync
**********************************************************
```

```
  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 2008 Sun Microsystems, Inc.  All rights reserved.
 23 # Use is subject to license terms.
 24 #
 25 # Copyright 2019 Joyent, Inc.
 26 #

 28 LIBRARY =       sysevent_conf_mod

 30 include ../Makefile.com

 32 LDLIBS +=       -lnvpair
 33 CPPFLAGS +=     -I ../../daemons/syseventconfd

 35 CERRWARN +=     -_gcc=-Wno-uninitialized

 37 # strange smatch false positive
 38 SMOFF +=       allocating_enough_data

 40 .KEEP_STATE:

 42 all: $(DYNLIB)

 44 install: all                            \
 45         $(ROOTLIBSYSEVENTDIR)           \
 46         $(ROOTLIBDIR)                   \
 47         $(ROOTLIBS)

 49 include ../Makefile.targ
```

```
**********************************************************
   63652 Mon Aug  5 08:37:51 2019
new/usr/src/common/ficl/vm.c
11506 smatch resync
**********************************************************
```

```
/*
 * v m . c
 * Forth Inspired Command Language - virtual machine methods
 * Author: John Sadler (john_sadler@alum.mit.edu)
 * Created: 19 July 1997
 * $Id: vm.c,v 1.17 2010/09/13 18:43:04 asau Exp $
 */
/*
 * This file implements the virtual machine of Ficl. Each virtual
 * machine retains the state of an interpreter. A virtual machine
 * owns a pair of stacks for parameters and return addresses, as
 * well as a pile of state variables and the two dedicated registers
 * of the interpreter.
 */
/*
 * Copyright (c) 1997-2001 John Sadler (john_sadler@alum.mit.edu)
 * All rights reserved.
 *
 * Get the latest Ficl release at http://ficl.sourceforge.net
 *
 * I am interested in hearing from anyone who uses Ficl. If you have
 * a problem, a success story, a defect, an enhancement request, or
 * if you would like to contribute to the Ficl release, please
 * contact me by email at the address above.
 *
 * L I C E N S E  and  D I S C L A I M E R
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. 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.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR 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 AUTHOR 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.
 */

/*
 * Copyright 2019 Joyent, Inc.
 */

#include "ficl.h"

#if FICL_ROBUST >= 2
#define FICL_VM_CHECK(vm)       \
        FICL_VM_ASSERT(vm, (*(vm->ip - 1)) == vm->runningWord)
#else
#define FICL_VM_CHECK(vm)
#endif
```



```
/*
 * v m B r a n c h R e l a t i v e
 */
void
ficlVmBranchRelative(ficlVm *vm, int offset)
{
        vm->ip += offset;
}
```

*unchanged_portion_omitted_*

```
/*
 * v m G e t W o r d T o P a d
 * Does vmGetWord and copies the result to the pad as a NULL terminated
 * string. Returns the length of the string. If the string is too long
 * to fit in the pad, it is truncated.
 */
int
ficlVmGetWordToPad(ficlVm *vm)
{
        ficlString s;
        char *pad = (char *)vm->pad;
        s = ficlVmGetWord(vm);

        if (FICL_STRING_GET_LENGTH(s) >= FICL_PAD_SIZE)
                FICL_STRING_SET_LENGTH(s, FICL_PAD_SIZE - 1);
        if (FICL_STRING_GET_LENGTH(s) > FICL_PAD_SIZE)
                FICL_STRING_SET_LENGTH(s, FICL_PAD_SIZE);

        (void) strncpy(pad, FICL_STRING_GET_POINTER(s),
            FICL_STRING_GET_LENGTH(s));
        pad[FICL_STRING_GET_LENGTH(s)] = '\0';
        return ((int)(FICL_STRING_GET_LENGTH(s)));
}
```

*unchanged_portion_omitted_*

```
**********************************************************
    1330 Mon Aug  5 08:37:51 2019
new/usr/src/lib/libpctx/Makefile.com
11506 smatch resync
**********************************************************
```

```
   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 2006 Sun Microsystems, Inc.  All rights reserved.
  23 # Use is subject to license terms.
  24 #
  25 # Copyright 2019 Joyent, Inc.
  25 # ident "%Z%%M% %I%     %E% SMI"
  26 #

  28 LIBRARY = libpctx.a
  29 VERS = .1

  31 OBJECTS = libpctx.o

  33 # include library definitions
  34 include ../../Makefile.lib

  36 LIBS = $(DYNLIB)
  36 LIBS = $(DYNLIB) $(LINTLIB)
  37 $(LINTLIB) :=   SRCS = ../common/llib-lpctx
  37 LDLIBS +=       -lproc -lc

  39 SRCDIR =        ../common

  41 CFLAGS +=       $(CCVERBOSE)
  42 CPPFLAGS +=     -D_REENTRANT -I$(SRCDIR)

  44 # false positive: pctx_run() error: dereferencing freed memory 'pctx'
  45 SMOFF += free

  47 .KEEP_STATE:

  49 all: $(LIBS)

  49 # x86 and sparc have different alignment complaints (all LINTED).
  50 # Make lint shut up about suppression directive not used.
  51 lint := LINTFLAGS += -erroff=E_SUPPRESSION_DIRECTIVE_UNUSED
  52 lint := LINTFLAGS64 += -erroff=E_SUPPRESSION_DIRECTIVE_UNUSED

  54 lint: lintcheck

  51 # include library targets
```



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

```
**********************************************************
    4575 Mon Aug  5 08:37:52 2019
new/usr/src/lib/libumem/Makefile.com
11506 smatch resync
**********************************************************
#
# 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 2008 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#
# Copyright 2019 Joyent, Inc.
# Copyright (c) 2019, Joyent, Inc.
#

#
# The build process for libumem is sightly different from that used by other
# libraries, because libumem must be built in two flavors - as a standalone
# for use by kmdb and as a normal library.  We use $(CURTYPE) to indicate the
# current flavor being built.
#

LIBRARY = libumem.a
STANDLIBRARY = libstandumem.so
VERS = .1

# By default, we build the shared library.  Construction of the standalone
# is specifically requested by architecture-specific Makefiles.
TYPES = library
CURTYPE = library

# This would be much prettier if a) Makefile.lib didn't reqire both $(SRCS) and
# $(OBJECTS) to be set or b) make gave us a nice way to do basename in pattern
# replacement definitions.

# Files specific to the library version of libumem
OBJECTS_library = \
        init_lib.o \
        umem_agent_support.o \
        umem_fail.o \
        umem_fork.o \
        umem_genasm.o \
        umem_update_thread.o \
        vmem_mmap.o \
        vmem_sbrk.o

SRCS_common_library = \
        $(ISASRCDIR)/umem_genasm.c
```



```
SRCS_library = $(OBJECTS_library:%.o=../common/%.c) $(SRC_common_library)

# Files specific to the standalone version of libumem
OBJECTS_standalone = \
        init_stand.o \
        stub_stand.o \
        vmem_stand.o

SRCS_standalone = $(OBJECTS_standalone:%.o=../common/%.c)

# Architecture-dependent files common to both versions of libumem
OBJECTS_common_isadep = \
        asm_subr.o

SRCS_common_isadep = \
        $(ISASRCDIR)/asm_subr.s

# Architecture-independent files common to both versions  of libumem
OBJECTS_common_common = \
        envvar.o \
        getpcstack.o \
        malloc.o \
        misc.o \
        vmem_base.o \
        umem.o \
        vmem.o

SRCS_common_common = $(OBJECTS_common_common:%.o=../common/%.c)

OBJECTS = \
        $(OBJECTS_$(CURTYPE)) \
        $(OBJECTS_common_isadep) \
        $(OBJECTS_common_common)

include ../../Makefile.lib
include ../../Makefile.rootfs

SRCS = \
        $(SRCS_$(CURTYPE)) \
        $(SRCS_common_common)

SRCDIR = ../common

#
# Used to verify that the standalone doesn't have any unexpected external
# dependencies.
#
LINKTEST_OBJ = objs/linktest_stand.o

CLOBBERFILES_standalone = $(LINKTEST_OBJ)
CLOBBERFILES += $(CLOBBERFILES_$(CURTYPE))

LIBS_standalone = $(STANDLIBRARY)
LIBS_library = $(DYNLIB)
LIBS_library = $(DYNLIB) $(LINTLIB)
LIBS =  $(LIBS_$(CURTYPE))

MAPFILE_SUPPLEMENTAL_standalone = ../common/stand_mapfile
MAPFILE_SUPPLEMENTAL = $(MAPFILE_SUPPLEMENTAL_$(CURTYPE))

LDLIBS +=       -lc

LDFLAGS_standalone = $(ZNOVERSION) $(BREDUCE) -M../common/mapfile-vers \
        -M$(MAPFILE_SUPPLEMENTAL) -dy -r
LDFLAGS = $(LDFLAGS_$(CURTYPE))
```

```
ASFLAGS_standalone = -DUMEM_STANDALONE
ASFLAGS_library =
ASFLAGS += -P $(ASFLAGS_$(CURTYPE)) -D_ASM

$(LINTLIB) := SRCS = ../common/$(LINTSRC)

# We want the thread-specific errno in the library, but we don't want it in
# the standalone.  $(DTS_ERRNO) is designed to add -D_TS_ERRNO to $(CPPFLAGS),
# in order to enable this feature.  Conveniently, -D_REENTRANT does the same
# thing.  As such, we null out $(DTS_ERRNO) to ensure that the standalone
# doesn't get it.
DTS_ERRNO=

# We need to rename some standard functions so we can easily implement them
# in consumers.
STAND_RENAMED_FUNCS= \
        atomic_add_64 \
        atomic_add_32_nv \
        atomic_swap_64 \
        snprintf \
        vsnprintf

CPPFLAGS_standalone = -DUMEM_STANDALONE $(STAND_RENAMED_FUNCS:%=-D%=umem_%)
CPPFLAGS_library = -D_REENTRANT
CPPFLAGS +=     -I../common -I../../common/inc $(CPPFLAGS_$(CURTYPE))

CFLAGS_standalone = $(STAND_FLAGS_32)
CFLAGS_common =
CFLAGS += $(CFLAGS_$(CURTYPE)) $(CFLAGS_common)

CFLAGS64_standalone = $(STAND_FLAGS_64)
CFLAGS64 += $(CCVERBOSE) $(CFLAGS64_$(CURTYPE)) $(CFLAGS64_common)

# false positive for umem_alloc_sizes_add()
pics/umem.o := SMOFF += index_overflow
objs/umem.o := SMOFF += index_overflow
INSTALL_DEPS_library =          $(ROOTLINKS) $(ROOTLINT) $(ROOTLIBS)

INSTALL_DEPS_library =          $(ROOTLINKS) $(ROOTLIBS)
#
# turn off ptr-cast warnings, since we do them all the time
#
LINTFLAGS +=    -erroff=E_BAD_PTR_CAST_ALIGN
LINTFLAGS64 +=  -erroff=E_BAD_PTR_CAST_ALIGN

DYNFLAGS +=     $(ZINTERPOSE)

.KEEP_STATE:
```

```
**********************************************************
    5257 Mon Aug  5 08:37:52 2019
new/usr/src/tools/smatch/Makefile
11506 smatch resync
**********************************************************
```

```
#
# 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) 2019, Joyent, Inc.
#

#
# The src/ sub-directory is un-modified copy of
# https://github.com/illumos/smatch/tree/0.5.1-il-4
# https://github.com/illumos/smatch/tree/0.5.1-il-3
#
# This Makefile installs just enough for us to be able to run smatch
# locally.
#

PROG = smatch
SPARSE_VERSION = 0.5.1-il-4
SPARSE_VERSION = 0.5.1-il-3

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 -m64 -msave-args -D__sun -Wall -Wno-unknown-pragmas -std=gnu99 -node
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) $(DIRMODE) $@

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 smatch_bits.o smatch_integer_overflow.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 \
```

```
 124         mtag_map.schema

 126 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 .PARALLEL: $(OBJS)

 135 all: $(PROG)

 137 install: all .WAIT $(ROOTONBLDMACHPROG) $(ROOTONBLDDATAFILES)

 139 clean:
 140         rm -f $(OBJS) $(BUILT_HEADERS)

 142 $(ROOTONBLDDATAFILES): $(SMATCHDATADIR)/smatch_data/db

 144 $(SMATCHDATADIR)/smatch_data/%: src/smatch_data/%
 145         $(INS.file)

 147 $(SMATCHDATADIR)/smatch_data/db:
 148         $(INS.dir)

 150 $(SMATCHDATADIR)/smatch_data:
 151         $(INS.dir)

 153 $(PROG): $(OBJS)
 154         $(LINK.c) $(OBJS) -o $@ $(LDLIBS)
 155         $(POST_PROCESS)

 157 %.o: src/%.c $(BUILT_HEADERS)
 158         $(COMPILE.c) -o $@ $<

 160 %.o: src/cwchash/%.c
 161         $(COMPILE.c) -o $@ $<

 163 src/check_list_local.h:
 164         touch src/check_list_local.h

 166 src/version.h:
 167         echo '#define SPARSE_VERSION "$(SPARSE_VERSION)"' > src/version.h

 169 include ../Makefile.targ
```

****
3019 Mon Aug 5 08:37:52 2019
new/usr/src/tools/smatch/src/Documentation/sparse-README.txt
11506 smatch resync
****

sparse (spˆ/rs), adj,., spars-er, spars-est.
sparse (spärs), 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.

```
**********************************************************
   11978 Mon Aug  5 08:37:53 2019
new/usr/src/tools/smatch/src/Makefile
11506 smatch resync
**********************************************************
```

```
1 VERSION=0.5.1-il-4
1 VERSION=0.5.1

3 # Generating file version.h if current version has changed
4 SPARSE_VERSION:=$(shell git describe 2>/dev/null || echo '$(VERSION)')
5 VERSION_H := $(shell cat version.h 2>/dev/null)
6 ifneq ($(lastword $(VERSION_H)),"$(SPARSE_VERSION)")
7 $(info $(shell echo '     GEN      'version.h))
8 $(shell echo '#define SPARSE_VERSION "$(SPARSE_VERSION)"' > version.h)
9 endif

11 OS = linux

13 ifeq ($(CC),"")
14 CC = gcc
15 endif

17 CFLAGS += -O2 -finline-functions -fno-strict-aliasing -g
18 CFLAGS += -Wall -Wwrite-strings -Wno-switch
19 LDFLAGS += -g -lm -lsqlite3 -lssl -lcrypto
20 LD = gcc
21 AR = ar
22 PKG_CONFIG = pkg-config
23 COMMON_CFLAGS = -O2 -finline-functions -fno-strict-aliasing -g
24 COMMON_CFLAGS += -Wall -Wwrite-strings

26 ALL_CFLAGS = $(COMMON_CFLAGS) $(PKG_CFLAGS) $(CFLAGS)
27 #
28 # For debugging, put this in local.mk:
29 #
30 #     CFLAGS += -O0 -DDEBUG -g3 -gdwarf-2
31 #

33 HAVE_LIBXML:=$(shell $(PKG_CONFIG) --exists libxml-2.0 2>/dev/null && echo 'yes'
34 HAVE_GCC_DEP:=$(shell touch .gcc-test.c &&                                    \
35                 $(CC) -c -Wp,-MD,.gcc-test.d .gcc-test.c 2>/dev/null && \
36                 echo 'yes'; rm -f .gcc-test.d .gcc-test.o .gcc-test.c)

38 GTK_VERSION:=3.0
39 HAVE_GTK:=$(shell $(PKG_CONFIG) --exists gtk+-$(GTK_VERSION) 2>/dev/null && echo
40 ifneq ($(HAVE_GTK),yes)
41         GTK_VERSION:=2.0
42         HAVE_GTK:=$(shell $(PKG_CONFIG) --exists gtk+-$(GTK_VERSION) 2>/dev/null
43 endif

45 LLVM_CONFIG:=llvm-config
46 HAVE_LLVM:=$(shell $(LLVM_CONFIG) --version >/dev/null 2>&1 && echo 'yes')

48 GCC_BASE := $(shell $(CC) --print-file-name=)
49 COMMON_CFLAGS += -DGCC_BASE=\"$(GCC_BASE)\"

51 MULTIARCH_TRIPLET := $(shell $(CC) -print-multiarch 2>/dev/null)
52 COMMON_CFLAGS += -DMULTIARCH_TRIPLET=\"$(MULTIARCH_TRIPLET)\"

54 ifeq ($(HAVE_GCC_DEP),yes)
55 COMMON_CFLAGS += -Wp,-MD,$(@D)/.$(@F).d
56 endif

58 DESTDIR=
59 INSTALL_PREFIX ?=$(HOME)
60 BINDIR=$(INSTALL_PREFIX)/bin
```



```
61 LIBDIR=$(INSTALL_PREFIX)/lib
62 MANDIR=$(INSTALL_PREFIX)/share/man
63 MAN1DIR=$(MANDIR)/man1
64 INCLUDEDIR=$(INSTALL_PREFIX)/include
65 PKGCONFIGDIR=$(LIBDIR)/pkgconfig
66 SMATCHDATADIR=$(INSTALL_PREFIX)/share/smatch

68 SMATCH_FILES=smatch_flow.o smatch_conditions.o smatch_slist.o smatch_states.o \
69         smatch_helper.o smatch_type.o smatch_hooks.o smatch_function_hooks.o \
70         smatch_modification_hooks.o smatch_extra.o smatch_estate.o smatch_math.o
71         smatch_sval.o smatch_ranges.o smatch_implied.o smatch_ignore.o smatch_pr
72         smatch_var_sym.o smatch_tracker.o smatch_files.o smatch_expression_stack
73         smatch_equiv.o smatch_buf_size.o smatch_strlen.o smatch_capped.o smatch_
74         smatch_expressions.o smatch_returns.o smatch_parse_call_math.o \
75         smatch_param_limit.o smatch_param_filter.o \
76         smatch_param_set.o smatch_comparison.o smatch_param_compare_limit.o smat
77         smatch_function_ptrs.o smatch_annotate.o smatch_string_list.o \
78         smatch_param_cleared.o smatch_start_states.o \
79         smatch_recurse.o smatch_data_source.o smatch_type_val.o \
80         smatch_common_functions.o smatch_struct_assignment.o \
81         smatch_unknown_value.o smatch_stored_conditions.o avl.o \
82         smatch_function_info.o smatch_links.o smatch_auto_copy.o \
83         smatch_type_links.o smatch_untracked_param.o smatch_impossible.o \
84         smatch_strings.o smatch_param_used.o smatch_container_of.o smatch_addres
85         smatch_buf_comparison.o smatch_real_absolute.o smatch_scope.o \
86         smatch_imaginary_absolute.o smatch_parameter_names.o \
87         smatch_return_to_param.o smatch_passes_array_size.o \
88         smatch_constraints.o smatch_constraints_required.o \
89         smatch_fn_arg_link.o smatch_about_fn_ptr_arg.o smatch_mtag.o \
90         smatch_mtag_map.o smatch_mtag_data.o \
91         smatch_param_to_mtag_data.o smatch_mem_tracker.o smatch_array_values.o \
92         smatch_nul_terminator.o smatch_assigned_expr.o smatch_kernel_user_data.o
93         smatch_statement_count.o smatch_integer_overflow.o smatch_bits.o
93         smatch_statement_count.o

95 SMATCH_CHECKS=$(shell ls check_*.c | sed -e 's/\.c/.o/')
96 SMATCH_DATA=smatch_data/kernel.allocation_funcs \
97         smatch_data/kernel.frees_argument smatch_data/kernel.puts_argument \
98         smatch_data/kernel.dev_queue_xmit smatch_data/kernel.returns_err_ptr \
99         smatch_data/kernel.dma_funcs smatch_data/kernel.returns_held_funcs \
100        smatch_data/kernel.no_return_funcs

102 SMATCH_SCRIPTS=smatch_scripts/add_gfp_to_allocations.sh \
103        smatch_scripts/build_kernel_data.sh \
104        smatch_scripts/call_tree.pl smatch_scripts/filter_kernel_deref_check.sh
105        smatch_scripts/find_expanded_holes.pl smatch_scripts/find_null_params.sh
106        smatch_scripts/follow_params.pl smatch_scripts/gen_allocation_list.sh \
107        smatch_scripts/gen_bit_shifters.sh smatch_scripts/gen_dma_funcs.sh \
108        smatch_scripts/generisize.pl smatch_scripts/gen_err_ptr_list.sh \
109        smatch_scripts/gen_expects_err_ptr.sh smatch_scripts/gen_frees_list.sh \
110        smatch_scripts/gen_gfp_flags.sh smatch_scripts/gen_no_return_funcs.sh \
111        smatch_scripts/gen_puts_list.sh smatch_scripts/gen_returns_held.sh \
112        smatch_scripts/gen_rosenberg_funcs.sh smatch_scripts/gen_sizeof_param.sh
113        smatch_scripts/gen_unwind_functions.sh smatch_scripts/kchecker \
114        smatch_scripts/kpatch.sh smatch_scripts/new_bugs.sh \
115        smatch_scripts/show_errs.sh smatch_scripts/show_ifs.sh \
116        smatch_scripts/show_unreachable.sh smatch_scripts/strip_whitespace.pl \
117        smatch_scripts/summarize_errs.sh smatch_scripts/test_kernel.sh \
118        smatch_scripts/trace_params.pl smatch_scripts/unlocked_paths.pl \
119        smatch_scripts/whitespace_only.sh smatch_scripts/wine_checker.sh \

121 PROGRAMS=test-lexing test-parsing obfuscate compile graph sparse \
122          test-linearize example test-unssa test-dissect ctags
123 INST_PROGRAMS=smatch cgcc

125 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
```

```
 258 $(foreach p,$(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 {} \;
```

```
**********************************************************
    2466 Mon Aug  5 08:37:53 2019
new/usr/src/tools/smatch/src/check_64bit_shift.c
11506 smatch resync
**********************************************************
```

```
/*
 * 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 void match_shift_mask(struct expression *expr)
{
        struct expression *right, *shifter;
        struct range_list *rl;
        char *str;

        expr = strip_expr(expr);
        if (expr->type != EXPR_BINOP || expr->op != '&')
                return;

        if (get_type(expr->left) != &ullong_ctype)
                return;

        if (type_bits(get_type(expr->right)) == 64)
                return;

        right = strip_expr(expr->right);
        if (right->type != EXPR_BINOP || right->op != SPECIAL_LEFTSHIFT)
                return;

        shifter = strip_expr(right->right);
        get_real_absolute_rl(shifter, &rl);
        if (rl_max(rl).uvalue < 32)
                return;

        str = expr_to_str(expr->right);
        sm_warning("should '%s' be a 64 bit type?", str);
        free_string(str);
}

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);
        add_hook(&match_shift_mask, BINOP_HOOK);
}
```

*_____unchanged_portion_omitted_*

```
**********************************************************
    2927 Mon Aug  5 08:37:54 2019
new/usr/src/tools/smatch/src/check_buffer_too_small_for_struct.c
11506 smatch resync
**********************************************************
```

```
 1 /*
 2  * Copyright (C) 2014 Oracle.
 3  *
 4  * This program is free software; you can redistribute it and/or
 5  * modify it under the terms of the GNU General Public License
 6  * as published by the Free Software Foundation; either version 2
 7  * of the License, or (at your option) any later version.
 8  *
 9  * This program is distributed in the hope that it will be useful,
10  * but WITHOUT ANY WARRANTY; without even the implied warranty of
11  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
12  * GNU General Public License for more details.
13  *
14  * You should have received a copy of the GNU General Public License
15  * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
16  */

18 #include "smatch.h"

20 static int my_id;

22 STATE(too_small);

24 static void match_assign(struct expression *expr)
25 {
26         struct symbol *left_type, *right_type;
27         struct expression *size_expr;
28         sval_t min_size;
29         int limit_type;
30         int bytes;

32         left_type = get_type(expr->left);
33         if (!left_type || left_type->type != SYM_PTR)
34                 return;
35         left_type = get_real_base_type(left_type);
36         if (!left_type || left_type->type != SYM_STRUCT)
37                 return;

39         right_type = get_type(expr->right);
40         if (!right_type || right_type->type != SYM_PTR)
41                 return;
42         right_type = get_real_base_type(right_type);
43         if (!right_type)
44                 return;
45         if (right_type != &void_ctype && type_bits(right_type) != 8)
46                 return;

48         bytes = get_array_size_bytes(expr->right);
49         if (bytes >= type_bytes(left_type))
50                 return;

52         size_expr = get_size_variable(expr->right, &limit_type);
46         size_expr = get_size_variable(expr->right);
53         if (!size_expr)
54                 return;
55         if (limit_type != ELEM_COUNT)
56                 return;

58         get_absolute_min(size_expr, &min_size);
59         if (min_size.value >= type_bytes(left_type))
60                 return;
```



```
62         set_state_expr(my_id, expr->left, &too_small);
63 }

65 static void match_dereferences(struct expression *expr)
66 {
67         struct symbol *left_type;
68         struct expression *right;
69         struct smatch_state *state;
70         char *name;
71         struct expression *size_expr;
72         sval_t min_size;
73         int limit_type;

75         if (expr->type != EXPR_PREOP)
76                 return;

78         expr = strip_expr(expr->unop);
79         state = get_state_expr(my_id, expr);
80         if (state != &too_small)
81                 return;

83         left_type = get_type(expr);
84         if (!left_type || left_type->type != SYM_PTR)
85                 return;
86         left_type = get_real_base_type(left_type);
87         if (!left_type || left_type->type != SYM_STRUCT)
88                 return;

90         right = get_assigned_expr(expr);
91         size_expr = get_size_variable(right, &limit_type);
82         size_expr = get_size_variable(right);
92         if (!size_expr)
93                 return;
94         if (limit_type != ELEM_COUNT)
95                 return;

97         get_absolute_min(size_expr, &min_size);
98         if (min_size.value >= type_bytes(left_type))
99                 return;

101         name = expr_to_str(right);
102         sm_warning("is '%s' large enough for 'struct %s'? %s", name, left_type->
103         free_string(name);
104         set_state_expr(my_id, expr, &undefined);
105 }
______unchanged_portion_omitted_
```

```
**********************************************************
    1352 Mon Aug  5 08:37:55 2019
new/usr/src/tools/smatch/src/check_cmn_err.c
11506 smatch resync
**********************************************************
```

```
/*
 * 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
 *
 * Copyright 2019 Joyent, Inc.
 */

/*
 * Heavily borrowed from check_wine.c: what we're doing here is teaching smatch
 * that cmn_err(CE_PANIC, ...) is noreturn.
 */

#include "scope.h"
#include "smatch.h"
#include "smatch_extra.h"

#define CE_PANIC (3)

void match_cmn_err(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;

        if (sval.value == CE_PANIC)
                nullify_path();
}


void check_cmn_err(int id)
{
        if (option_project != PROJ_ILLUMOS_KERNEL)
                return;

        add_function_hook("cmn_err", &match_cmn_err, NULL);
}
```

```
**********************************************************
   21157 Mon Aug  5 08:37:55 2019
new/usr/src/tools/smatch/src/check_debug.c
11506 smatch resync
**********************************************************
```

```
   1 /*
   2  * Copyright (C) 2009 Dan Carpenter.
   3  *
   4  * This program is free software; you can redistribute it and/or
   5  * modify it under the terms of the GNU General Public License
   6  * as published by the Free Software Foundation; either version 2
   7  * of the License, or (at your option) any later version.
   8  *
   9  * This program is distributed in the hope that it will be useful,
  10  * but WITHOUT ANY WARRANTY; without even the implied warranty of
  11  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  12  * GNU General Public License for more details.
  13  *
  14  * You should have received a copy of the GNU General Public License
  15  * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
  16  */

  18 #include "smatch.h"
  19 #include "smatch_slist.h"
  20 #include "smatch_extra.h"

  22 void show_sname_alloc(void);
  23 void show_data_range_alloc(void);
  24 void show_ptrlist_alloc(void);
  25 void show_sm_state_alloc(void);

  27 int local_debug;
  28 static int my_id;
  29 char *trace_variable;

  31 static void match_all_values(const char *fn, struct expression *expr, void *info
  32 {
  33         struct stree *stree;

  35         stree = get_all_states_stree(SMATCH_EXTRA);
  36         __print_stree(stree);
  37         free_stree(&stree);
  38 }
```

*unchanged_portion_omitted_*

```
 203 static void match_user_rl(const char *fn, struct expression *expr, void *info)
 204 {
 205         struct expression *arg;
 206         struct range_list *rl = NULL;
 201         struct range_list *rl;
 207         char *name;

 209         arg = get_argument_from_call_expr(expr->args, 0);
 210         name = expr_to_str(arg);

 212         get_user_rl(arg, &rl);
 213         sm_msg("user rl: '%s' = '%s'", name, show_rl(rl));

 215         free_string(name);
 216 }
```

*unchanged_portion_omitted_*

```
 385 static void match_buf_size(const char *fn, struct expression *expr, void *info)
 386 {
 387         struct expression *arg, *comp;
 388         struct range_list *rl;
```



```
 389         int elements, bytes;
 390         char *name;
 391         char buf[256] = "";
 392         int limit_type;
 393         int n;
 394         sval_t sval;

 396         arg = get_argument_from_call_expr(expr->args, 0);

 398         elements = get_array_size(arg);
 399         bytes = get_array_size_bytes_max(arg);
 400         rl = get_array_size_bytes_rl(arg);
 401         comp = get_size_variable(arg, &limit_type);
 395         comp = get_size_variable(arg);

 403         name = expr_to_str(arg);
 404         n = snprintf(buf, sizeof(buf), "buf size: '%s' %d elements, %d bytes", n
 405         free_string(name);

 407         if (!rl_to_sval(rl, &sval))
 408                 n += snprintf(buf + n, sizeof(buf) - n, " (rl = %s)", show_rl(rl

 410         if (comp) {
 411                 name = expr_to_str(comp);
 412                 snprintf(buf + n, sizeof(buf) - n, "[size_var=%s %s]", limit_typ
 406                 snprintf(buf + n, sizeof(buf) - n, "[size_var=%s]", name);
 413                 free_string(name);
 414         }
 415         sm_msg("%s", buf);
 416 }
```

*unchanged_portion_omitted_*

```
 507 static void match_debug_implied_on(const char *fn, struct expression *expr, void
 508 {
 509         option_debug_implied = 1;
 510 }

 512 static void match_debug_implied_off(const char *fn, struct expression *expr, voi
 513 {
 514         option_debug_implied = 0;
 515 }

 513 static void match_about(const char *fn, struct expression *expr, void *info)
 514 {
 515         struct expression *arg;
 516         struct sm_state *sm;
 517         char *name;

 519         sm_msg("---- about ----");
 520         match_print_implied(fn, expr, NULL);
 521         match_buf_size(fn, expr, NULL);
 522         match_strlen(fn, expr, NULL);
 523         match_real_absolute(fn, expr, NULL);

 525         arg = get_argument_from_call_expr(expr->args, 0);
 526         name = expr_to_str(arg);
 527         if (!name) {
 528                 sm_msg("info: not a straight forward variable.");
 529                 return;
 530         }

 532         FOR_EACH_SM(__get_cur_stree(), sm) {
 533                 if (strcmp(sm->name, name) != 0)
 534                         continue;
 535                 sm_msg("%s", show_sm(sm));
 536         } END_FOR_EACH_SM(sm);
```

```
 537 }
_____unchanged_portion_omitted_

 638 static void match_mtag(const char *fn, struct expression *expr, void *info)
 639 {
 640         struct expression *arg;
 641         char *name;
 642         mtag_t tag = 0;
 643         int offset = 0;

 645         arg = get_argument_from_call_expr(expr->args, 0);
 646         name = expr_to_str(arg);
 647         expr_to_mtag_offset(arg, &tag, &offset);
 648         sm_msg("mtag: '%s' => tag: %llu %d", name, tag, offset);
 650         get_mtag(arg, &tag);
 651         sm_msg("mtag: '%s' => tag: %lld", name, tag);
 649         free_string(name);
 650 }
_____unchanged_portion_omitted_

 666 static void match_container(const char *fn, struct expression *expr, void *info)
 667 {
 668         struct expression *container, *x;
 669         char *cont, *name, *str;

 671         container = get_argument_from_call_expr(expr->args, 0);
 672         x = get_argument_from_call_expr(expr->args, 1);

 674         str = get_container_name(container, x);
 675         cont = expr_to_str(container);
 676         name = expr_to_str(x);
 677         sm_msg("container: '%s' vs '%s' --> '%s'", cont, name, str);
 678         free_string(cont);
 679         free_string(name);
 680 }

 682 static void match_state_count(const char *fn, struct expression *expr, void *inf
 683 {
 684         sm_msg("state_count = %d\n", sm_state_counter);
 685 }
_____unchanged_portion_omitted_

 736 void check_debug(int id)
 737 {
 738         my_id = id;
 739         add_function_hook("__smatch_about", &match_about, NULL);
 740         add_function_hook("__smatch_all_values", &match_all_values, NULL);
 741         add_function_hook("__smatch_state", &match_state, NULL);
 742         add_function_hook("__smatch_states", &match_states, NULL);
 743         add_function_hook("__smatch_value", &match_print_value, NULL);
 744         add_function_hook("__smatch_known", &match_print_known, NULL);
 745         add_function_hook("__smatch_implied", &match_print_implied, NULL);
 746         add_function_hook("__smatch_implied_min", &match_print_implied_min, NULL
 747         add_function_hook("__smatch_implied_max", &match_print_implied_max, NULL
 748         add_function_hook("__smatch_user_rl", &match_user_rl, NULL);
 749         add_function_hook("__smatch_capped", &match_capped, NULL);
 750         add_function_hook("__smatch_hard_max", &match_print_hard_max, NULL);
 751         add_function_hook("__smatch_fuzzy_max", &match_print_fuzzy_max, NULL);
 752         add_function_hook("__smatch_absolute", &match_print_absolute, NULL);
 753         add_function_hook("__smatch_absolute_min", &match_print_absolute_min, NU
 754         add_function_hook("__smatch_absolute_max", &match_print_absolute_max, NU
 755         add_function_hook("__smatch_real_absolute", &match_real_absolute, NULL);
 756         add_function_hook("__smatch_sval_info", &match_sval_info, NULL);
 757         add_function_hook("__smatch_member_name", &match_member_name, NULL);
 758         add_function_hook("__smatch_possible", &match_possible, NULL);
 759         add_function_hook("__smatch_cur_stree", &match_cur_stree, NULL);
```



```
 760         add_function_hook("__smatch_strlen", &match_strlen, NULL);
 761         add_function_hook("__smatch_buf_size", &match_buf_size, NULL);
 762         add_function_hook("__smatch_note", &match_note, NULL);
 763         add_function_hook("__smatch_dump_related", &match_dump_related, NULL);
 764         add_function_hook("__smatch_compare", &match_compare, NULL);
 765         add_function_hook("__smatch_debug_on", &match_debug_on, NULL);
 766         add_function_hook("__smatch_debug_check", &match_debug_check, NULL);
 767         add_function_hook("__smatch_debug_off", &match_debug_off, NULL);
 768         add_function_hook("__smatch_local_debug_on", &match_local_debug_on, NULL
 769         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
 770         add_function_hook("__smatch_intersection", &match_intersection, NULL);
 771         add_function_hook("__smatch_type", &match_type, NULL);
 772         add_implied_return_hook("__smatch_type_rl_helper", &match_type_rl_return
 773         add_function_hook("__smatch_merge_tree", &match_print_merge_tree, NULL);
 774         add_function_hook("__smatch_stree_id", &match_print_stree_id, NULL);
 775         add_function_hook("__smatch_mtag", &match_mtag, NULL);
 776         add_function_hook("__smatch_mtag_data", &match_mtag_data_offset, NULL);
 777         add_function_hook("__smatch_state_count", &match_state_count, NULL);
 778         add_function_hook("__smatch_mem", &match_mem, NULL);
 779         add_function_hook("__smatch_exit", &match_exit, NULL);
 780         add_function_hook("__smatch_container", &match_container, NULL);

 782         add_hook(free_old_stree, AFTER_FUNC_HOOK);
 783         add_hook(trace_var, STMT_HOOK_AFTER);
 784 }
_____unchanged_portion_omitted_
```

```
**********************************************************
    2890 Mon Aug  5 08:37:56 2019
new/usr/src/tools/smatch/src/check_debug.h
11506 smatch resync
**********************************************************
______unchanged_portion_omitted_
  60 #define __smatch_type_rl(type, fmt...) __smatch_type_rl_helper((type)0, fmt)
  61 #define __smatch_rl(fmt...) __smatch_type_rl(long long, fmt)

  63 static inline void __smatch_bit_info(long long expr){}

  65 static inline void __smatch_oops(unsigned long null_val){}

  67 static inline void __smatch_merge_tree(long long var){}

  69 static inline void __smatch_stree_id(void){}

  71 static inline void __smatch_mtag(void *p){}
  72 static inline void __smatch_mtag_data(long long arg){}
  73 static inline void __smatch_exit(void){}

  75 static inline void __smatch_state_count(void){}
  76 static inline void __smatch_mem(void){}

  78 static inline void __smatch_container(long long container, long long x){}
  79 #endif
```

****
2233 Mon Aug 5 08:37:56 2019
new/usr/src/tools/smatch/src/check_dma_mapping_error.c
11506 smatch resync
****
_____***unchanged_portion_omitted_***

```
 33 static void match_assign(const char *fn, struct expression *expr, void *unused)
 34 {
 35         struct range_list *rl;

 37         if (!get_implied_rl(expr->right, &rl))
 38                 return;
 39         if (rl_max(rl).value != 1)
 40                 return;
 41         set_state_expr(my_id, expr->left, &positive);
 42 }
```

_____***unchanged_portion_omitted_***

****
6668 Mon Aug 5 08:37:56 2019
new/usr/src/tools/smatch/src/check_err_ptr_deref.c
11506 smatch resync
****
_____***unchanged_portion_omitted_***

```
 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", -4095, -1, &match_err, 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 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    7059 Mon Aug  5 08:37:57 2019
new/usr/src/tools/smatch/src/check_free_strict.c
11506 smatch resync
**********************************************************
______unchanged_portion_omitted_

294 static void match_untracked(struct expression *call, int param)
295 {
296         struct state_list *slist = NULL;
297         struct expression *arg;
298         struct sm_state *sm;
299         char *name;
300         char buf[64];
301         int len;

303         arg = get_argument_from_call_expr(call->args, param);
304         if (!arg)
305                 return;

307         name = expr_to_var(arg);
308         if (!name)
309                 return;
310         snprintf(buf, sizeof(buf), "%s->", name);
311         free_string(name);
312         len = strlen(buf);

314         FOR_EACH_MY_SM(my_id, __get_cur_stree(), sm) {
315                 if (strncmp(sm->name, buf, len) == 0)
316                         add_ptr_list(&slist, sm);
317         } END_FOR_EACH_SM(sm);

319         FOR_EACH_PTR(slist, sm) {
320                 set_state(sm->owner, sm->name, sm->sym, &ok);
321         } END_FOR_EACH_PTR(sm);

323         free_slist(&slist);
324 }

326 void check_free_strict(int id)
327 {
328         my_id = id;

330         if (option_project != PROJ_KERNEL)
331                 return;

333         add_function_hook("kfree", &match_free, INT_PTR(0));
334         add_function_hook("kmem_cache_free", &match_free, INT_PTR(1));

336         if (option_spammy)
337                 add_hook(&match_symbol, SYM_HOOK);
338         add_hook(&match_dereferences, DEREF_HOOK);
339         add_hook(&match_call, FUNCTION_CALL_HOOK);
340         add_hook(&match_return, RETURN_HOOK);

342         add_modification_hook_late(my_id, &ok_to_use);
343         add_pre_merge_hook(my_id, &pre_merge_hook);

345         select_return_states_hook(PARAM_FREED, &set_param_freed);
346         add_untracked_param_hook(&match_untracked);
347 }
______unchanged_portion_omitted_
```

****
3457 Mon Aug 5 08:37:59 2019
new/usr/src/tools/smatch/src/check_held_dev.c
11506 smatch resync
****
_____***unchanged_portion_omitted_***

```
 101 static void register_returns_held_funcs(void)
 102 {
 103         struct token *token;
 104         const char *func;

 106         token = get_tokens_file("kernel.returns_held_funcs");
 107         if (!token)
 108                 return;
 109         if (token_type(token) != TOKEN_STREAMBEGIN)
 110                 return;
 111         token = token->next;
 112         while (token_type(token) != TOKEN_STREAMEND) {
 113                 if (token_type(token) != TOKEN_IDENT)
 114                         return;
 115                 func = show_ident(token->ident);
 116                 return_implies_state_sval(func, valid_ptr_min_sval, valid_ptr_ma
 116                 return_implies_state(func, valid_ptr_min, valid_ptr_max,
 117                                      &match_returns_held, NULL);
 118                 return_implies_state(func, 0, 0, &match_returns_null,
 119                                           NULL);
 120                 token = token->next;
 121         }
 122         clear_token_alloc();
 123 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   12812 Mon Aug  5 08:38:00 2019
new/usr/src/tools/smatch/src/check_kernel.c
11506 smatch resync
**********************************************************
```

```
/*
 * 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 sval_t err_ptr_min;
static sval_t err_ptr_max;
static sval_t null_ptr;

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(err_ptr_min, err_ptr_max);
                *rl = cast_rl(get_type(arg), *rl);
        }
        if (!get_implied_rl(arg, rl))
                *rl = alloc_rl(ll_to_sval(-4095), ll_to_sval(-1));
        return 1;
}
```

*_____unchanged_portion_omitted_*

```
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;
        struct range_list *rl;

        arg = get_argument_from_call_expr(call_expr->args, param);
        pre_state = get_state_expr(SMATCH_EXTRA, arg);
        if (estate_rl(pre_state)) {
                rl = estate_rl(pre_state);
                rl = remove_range(rl, null_ptr, null_ptr);
                rl = remove_range(rl, err_ptr_min, err_ptr_max);
        } else {
                rl = alloc_rl(valid_ptr_min_sval, valid_ptr_max_sval);
```



```
        }
        end_state = alloc_estate_rl(rl);
        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);
        call_results_to_rl(call_expr, &ptr_ctype, "0,(-4095)-(-1)", &rl);
        rl = alloc_rl(ll_to_sval(-4095), ll_to_sval(0));
        rl = rl_intersection(estate_rl(pre_state), rl);
        rl = cast_rl(get_type(arg), 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 range_list *rl;
        struct smatch_state *new_state;

        arg = get_argument_from_call_expr(call_expr->args, 0);
        pre_state = get_state_expr(SMATCH_EXTRA, arg);
        if (estate_rl(pre_state)) {
                rl = estate_rl(pre_state);
                rl = remove_range(rl, err_ptr_min, err_ptr_max);
        } else {
                rl = alloc_rl(valid_ptr_min_sval, valid_ptr_max_sval);
        }
        rl = cast_rl(get_type(arg), rl);
        set_extra_expr_nomod(arg, alloc_estate_rl(rl));
        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 range_list *rl;
        struct smatch_state *new_state;

        arg = get_argument_from_call_expr(call_expr->args, 0);
        pre_state = get_state_expr(SMATCH_EXTRA, arg);
        rl = estate_rl(pre_state);
        if (!rl)
                rl = alloc_rl(err_ptr_min, err_ptr_max);
        rl = rl_intersection(rl, alloc_rl(err_ptr_min, err_ptr_max));
        rl = cast_rl(get_type(arg), rl);
        set_extra_expr_nomod(arg, alloc_estate_rl(rl));
        new_state = estate_filter_range(pre_state, sval_type_min(&long_ctype), l
        new_state = estate_filter_range(new_state, ll_to_sval(0), sval_type_max(
```

```
130         set_extra_expr_nomod(arg, new_state);
154 }
______unchanged_portion_omitted_

405 bool is_ignored_kernel_data(const char *name)
406 {
407         if (option_project != PROJ_KERNEL)
408                 return false;

410         /*
411          * On the file I was looking at lockdep was 25% of the DB.
412          */
413         if (strstr(name, ".dep_map."))
414                 return true;
415         if (strstr(name, ".lockdep_map."))
416                 return true;
417         return false;
418 }

420 void check_kernel(int id)
421 {
422         if (option_project != PROJ_KERNEL)
423                 return;

425         err_ptr_min.type = &ptr_ctype;
426         err_ptr_min.value = -4095;
427         err_ptr_max.type = &ptr_ctype;
428         err_ptr_max.value = -1l;
429         null_ptr.type = &ptr_ctype;
430         null_ptr.value = 0;

432         err_ptr_min = sval_cast(&ptr_ctype, err_ptr_min);
433         err_ptr_max = sval_cast(&ptr_ctype, err_ptr_max);

435         add_implied_return_hook("ERR_PTR", &implied_err_cast_return, NULL);
436         add_implied_return_hook("ERR_CAST", &implied_err_cast_return, NULL);
437         add_implied_return_hook("PTR_ERR", &implied_err_cast_return, NULL);
438         add_hook(hack_ERR_PTR, AFTER_DEF_HOOK);
439         return_implies_state("IS_ERR_OR_NULL", 0, 0, &match_param_valid_ptr, (vo
440         return_implies_state("IS_ERR_OR_NULL", 1, 1, &match_param_err_or_null, (
441         return_implies_state("IS_ERR", 0, 0, &match_not_err, NULL);
442         return_implies_state("IS_ERR", 1, 1, &match_err, NULL);
443         return_implies_state("tomoyo_memory_ok", 1, 1, &match_param_valid_ptr, (

445         add_macro_assign_hook_extra("container_of", &match_container_of_macro, N
446         add_hook(match_container_of, ASSIGNMENT_HOOK);

448         add_implied_return_hook("find_next_bit", &match_next_bit, NULL);
449         add_implied_return_hook("find_next_zero_bit", &match_next_bit, NULL);
450         add_implied_return_hook("find_first_bit", &match_next_bit, NULL);
451         add_implied_return_hook("find_first_zero_bit", &match_next_bit, NULL);

453         add_implied_return_hook("fls", &match_fls, NULL);
454         add_implied_return_hook("fls64", &match_fls, NULL);

456         add_function_hook("__ftrace_bad_type", &__match_nullify_path_hook, NULL)
457         add_function_hook("__write_once_size", &match__write_once_size, NULL);

459         add_function_hook("__read_once_size", &match__read_once_size, NULL);
460         add_function_hook("__read_once_size_nocheck", &match__read_once_size, NU

462         if (option_info)
463                 add_hook(match_end_file, END_FILE_HOOK);
464 }
______unchanged_portion_omitted_
```

```
**********************************************************
   54325 Mon Aug  5 08:38:00 2019
new/usr/src/tools/smatch/src/check_kernel_printf.c
11506 smatch resync
**********************************************************
______unchanged_portion_omitted_

static void time_and_date(const char *fmt, struct symbol *type, struct symbol *b
{
        assert(tolower(fmt[0]) == 't');

        if (fmt[1] == 'R' && !is_struct_tag(basetype, "rtc_time"))
                sm_error("'%%ptR' expects argument of type struct 'rtc_time', ar
                         vaidx, type_to_str(type));
}

static void check_clock(const char *fmt, struct symbol *type, struct symbol *bas
{
        assert(fmt[0] == 'C');
        if (isalnum(fmt[1])) {
                if (!strchr("nr", fmt[1]))
                        sm_warning("'%%pC' can only be followed by one of [nr]")
                if (isalnum(fmt[2]))
                        sm_warning("'%%pC%c' cannot be followed by '%c'", fmt[1]
        }
        if (!is_struct_tag(basetype, "clk"))
                sm_error("'%%pC' expects argument of type 'struct clk*', argumen
                       vaidx, type_to_str(type));
}
______unchanged_portion_omitted_

static void
pointer(const char *fmt, struct expression *arg, int vaidx)
{
        struct symbol *type, *basetype;

        type = get_type(arg);
        if (!type) {
                sm_warning("could not determine type of argument %d", vaidx);
                return;
        }
        if (!is_ptr_type(type)) {
                sm_error("%%p expects pointer argument, but argument %d has type
                         vaidx, type_to_str(type));
                return;
        }
        /* Just plain %p, nothing to check. */
        if (!isalnum(*fmt))
                return;

        basetype = get_real_base_type(type);
        if (is_void_type(basetype))
                return;
        /*
         * Passing a pointer-to-array is harmless, but most likely one
         * meant to pass pointer-to-first-element. If basetype is
         * array type, we issue a notice and "dereference" the types
         * once more.
         */
        if (basetype->type == SYM_ARRAY) {
                spam("note: passing pointer-to-array; is the address-of redundan
                type = basetype;
                basetype = get_real_base_type(type);
        }

        /*
```



```
         * We pass both the type and the basetype to the helpers. If,
         * for example, the pointer is really a decayed array which is
         * passed to %pI4, we might want to check that it is in fact
         * an array of four bytes. But most are probably only
         * interested in whether the basetype makes sense. Also, the
         * pointer may carry some annotation such as __user which
         * might be worth checking in the handlers which actually
         * dereference the pointer.
         */

        switch (*fmt) {
        case 'b':
        case 'F':
        case 'f':
        case 'S':
        case 's':
        case 'B':
                /* Can we do anything sensible? Check that the arg is a function
                break;

        case 'R':
        case 'r':
                resource_string(fmt, type, basetype, vaidx);
                break;
        case 'M':
        case 'm':
                mac_address_string(fmt, type, basetype, vaidx);
                break;
        case 'I':
        case 'i':
                switch (fmt[1]) {
                case '4':
                        ip4(fmt, type, basetype, vaidx);
                        break;
                case '6':
                        ip6(fmt, type, basetype, vaidx);
                        break;
                case 'S':
                        ipS(fmt, type, basetype, vaidx);
                        break;
                default:
                        sm_warning("'%%p%c' must be followed by one of [46S]", f
                        break;
                }
                break;
       /*
        * %pE and %ph can handle any valid pointer. We still check
        * whether all the subsequent alphanumerics are valid for the
        * particular %pX conversion.
        */
        case 'E':
                escaped_string(fmt, type, basetype, vaidx);
                break;
        case 'h':
                hex_string(fmt, type, basetype, vaidx);
                break;
        case 'U': /* TODO */
                break;
        case 'V':
                va_format(fmt, type, basetype, vaidx);
                break;
        case 'K': /* TODO */
                break;
        case 'N':
                netdev_feature(fmt, type, basetype, vaidx);
                break;
```

```
761         case 'a':
762                 address_val(fmt, type, basetype, vaidx);
763                 break;
764         case 'D':
765         case 'd':
766                 dentry_file(fmt, type, basetype, vaidx);
767                 break;
768         case 't':
769                 time_and_date(fmt, type, basetype, vaidx);
770                 break;
771         case 'C':
772                 check_clock(fmt, type, basetype, vaidx);
773                 break;
774         case 'g':
775                 block_device(fmt, type, basetype, vaidx);
776                 break;
777         case 'G':
778                 flag_string(fmt, type, basetype, vaidx);
779                 break;
780         case 'O':
781                 device_node_string(fmt, type, basetype, vaidx);
782                 break;
783         case 'x':
784                 /* 'x' is for an unhashed pointer */
785                 break;
786         default:
787                 sm_error("unrecognized %%p extension '%c', treated as normal %%p
788         }
789 }
```

*unchanged_portion_omitted_*

```
**********************************************************
    5616 Mon Aug  5 08:38:00 2019
new/usr/src/tools/smatch/src/check_list.h
11506 smatch resync
**********************************************************
```

```
#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_integer_overflow)
CK(register_integer_overflow_links)
CK(register_real_absolute)
CK(register_imaginary_absolute)
CK(register_bits)
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_data)
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)
```

```
127 CK(check_platform_device_put)
128 CK(check_info_leak)
129 CK(check_return_enomem)
130 CK(check_get_user_overflow)
131 CK(check_get_user_overflow2)
132 CK(check_access_ok_math)
133 CK(check_container_of)
134 CK(check_input_free_device)
135 CK(check_select)
136 CK(check_memset)
137 CK(check_logical_instead_of_bitwise)
138 CK(check_kmalloc_wrong_size)
139 CK(check_pointer_math)
140 CK(check_bit_shift)
141 CK(check_macro_side_effects)
142 CK(check_sizeof)
143 CK(check_return_cast)
144 CK(check_or_vs_and)
145 CK(check_passes_sizeof)
146 CK(check_assign_vs_compare)
147 CK(check_missing_break)
148 CK(check_array_condition)
149 CK(check_struct_type)
150 CK(check_64bit_shift)
151 CK(check_wrong_size_arg)
152 CK(check_cast_assign)
153 CK(check_readl_infinite_loops)
154 CK(check_double_checking)
155 CK(check_shift_to_zero)
156 CK(check_indenting)
157 CK(check_unreachable)
158 CK(check_no_if_block)
159 CK(check_buffer_too_small_for_struct)
160 CK(check_uninitialized)
161 CK(check_signed_integer_overflow_check)
162 CK(check_continue_vs_break)
163 CK(check_impossible_mask)
164 CK(check_syscall_arg_type)
165 CK(check_trinity_generator)

167 /* <- your test goes here */
168 /* CK(register_template) */

170 /* kernel specific */
171 CK(check_kernel_printf)
172 CK(check_locking)
173 CK(check_puts_argument)
174 CK(check_err_ptr)
175 CK(check_err_ptr_deref)
176 CK(check_expects_err_ptr)
177 CK(check_held_dev)
178 CK(check_return_negative_var)
179 CK(check_rosenberg)
180 CK(check_rosenberg2)
181 CK(check_wait_for_common)
182 CK(check_bogus_irqrestore)
183 CK(check_zero_to_err_ptr)
184 CK(check_freeing_devm)
185 CK(check_off_by_one_relative)
186 CK(check_capable)
187 CK(check_ns_capable)
188 CK(check_test_bit)
189 CK(check_dma_mapping_error)
190 CK(check_nospec)
191 CK(check_nospec_barrier)
192 CK(check_spectre)
```



```
193 CK(check_spectre_second_half)
194 CK(check_implicit_dependencies)

196 /* wine specific stuff */
197 CK(check_wine_filehandles)
198 CK(check_wine_WtoA)

200 /* illumos specific */
201 CK(check_all_func_returns)
202 CK(check_cmn_err)

204 #include "check_list_local.h"

206 CK(register_scope)
207 CK(register_stored_conditions)
208 CK(register_stored_conditions_links)
209 CK(register_sval)
210 CK(register_buf_size_late)
211 CK(register_smatch_extra_late)
212 CK(register_assigned_expr) /* This is used by smatch_extra.c so it has to come r
213 CK(register_assigned_expr_links)
214 CK(register_modification_hooks_late)  /* has to come after smatch_extra */
215 CK(register_comparison_late) /* has to come after modification_hooks_late */
216 CK(register_function_hooks)
217 CK(check_kernel)  /* this is overwriting stuff from smatch_extra_late */
218 CK(check_wine)
219 CK(register_returns)

221 #ifdef __undo_CK_def
222 #undef CK
223 #undef __undo_CK_def
224 #endif
```

```
**********************************************************
   35369 Mon Aug  5 08:38:00 2019
new/usr/src/tools/smatch/src/check_locking.c
11506 smatch resync
**********************************************************
______unchanged_portion_omitted_

  48 enum return_type {
  49         ret_any,
  50         ret_non_zero,
  51         ret_zero,
  52         ret_one,
  53         ret_negative,
  54         ret_positive,
  55 };
______unchanged_portion_omitted_

 104 static struct lock_info kernel_lock_table[] = {
 105         {"lock_kernel",   LOCK,   "BKL", NO_ARG, ret_any},
 106         {"unlock_kernel", UNLOCK, "BKL", NO_ARG, ret_any},

 108         {"spin_lock",                   LOCK,   "spin_lock", 0, ret_any},
 109         {"spin_unlock",                 UNLOCK, "spin_lock", 0, ret_any},
 110         {"spin_lock_nested",            LOCK,   "spin_lock", 0, ret_any},
 111         {"_spin_lock",                  LOCK,   "spin_lock", 0, ret_any},
 112         {"_spin_unlock",                UNLOCK, "spin_lock", 0, ret_any},
 113         {"_spin_lock_nested",           LOCK,   "spin_lock", 0, ret_any},
 114         {"__spin_lock",                 LOCK,   "spin_lock", 0, ret_any},
 115         {"__spin_unlock",               UNLOCK, "spin_lock", 0, ret_any},
 116         {"__spin_lock_nested",          LOCK,   "spin_lock", 0, ret_any},
 117         {"raw_spin_lock",               LOCK,   "spin_lock", 0, ret_any},
 118         {"raw_spin_unlock",             UNLOCK, "spin_lock", 0, ret_any},
 119         {"_raw_spin_lock",              LOCK,   "spin_lock", 0, ret_any},
 120         {"_raw_spin_lock_nested",       LOCK,   "spin_lock", 0, ret_any},
 121         {"_raw_spin_unlock",            UNLOCK, "spin_lock", 0, ret_any},
 122         {"__raw_spin_lock",             LOCK,   "spin_lock", 0, ret_any},
 123         {"__raw_spin_unlock",           UNLOCK, "spin_lock", 0, ret_any},

 125         {"spin_lock_irq",               LOCK,   "spin_lock", 0, ret_any},
 126         {"spin_unlock_irq",             UNLOCK, "spin_lock", 0, ret_any},
 127         {"_spin_lock_irq",              LOCK,   "spin_lock", 0, ret_any},
 128         {"_spin_unlock_irq",            UNLOCK, "spin_lock", 0, ret_any},
 129         {"__spin_lock_irq",             LOCK,   "spin_lock", 0, ret_any},
 130         {"__spin_unlock_irq",           UNLOCK, "spin_lock", 0, ret_any},
 131         {"_raw_spin_lock_irq",          LOCK,   "spin_lock", 0, ret_any},
 132         {"_raw_spin_unlock_irq",        UNLOCK, "spin_lock", 0, ret_any},
 133         {"__raw_spin_unlock_irq",       UNLOCK, "spin_lock", 0, ret_any},
 134         {"spin_lock_irqsave",           LOCK,   "spin_lock", 0, ret_any},
 135         {"spin_unlock_irqrestore",      UNLOCK, "spin_lock", 0, ret_any},
 136         {"_spin_lock_irqsave",          LOCK,   "spin_lock", 0, ret_any},
 137         {"_spin_unlock_irqrestore",     UNLOCK, "spin_lock", 0, ret_any},
 138         {"__spin_lock_irqsave",         LOCK,   "spin_lock", 0, ret_any},
 139         {"__spin_unlock_irqrestore",    UNLOCK, "spin_lock", 0, ret_any},
 140         {"_raw_spin_lock_irqsave",      LOCK,   "spin_lock", 0, ret_any},
 141         {"_raw_spin_unlock_irqrestore", UNLOCK, "spin_lock", 0, ret_any},
 142         {"__raw_spin_lock_irqsave",     LOCK,   "spin_lock", 0, ret_any},
 143         {"__raw_spin_unlock_irqrestore", UNLOCK, "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         {"__spin_lock_irqsave_nested",  LOCK,   "spin_lock", 0, ret_any},
 147         {"_raw_spin_lock_irqsave_nested", LOCK, "spin_lock", 0, ret_any},
 148         {"spin_lock_bh",                LOCK,   "spin_lock", 0, ret_any},
 149         {"spin_unlock_bh",              UNLOCK, "spin_lock", 0, ret_any},
 150         {"_spin_lock_bh",               LOCK,   "spin_lock", 0, ret_any},
 151         {"_spin_unlock_bh",             UNLOCK, "spin_lock", 0, ret_any},
 152         {"__spin_lock_bh",              LOCK,   "spin_lock", 0, ret_any},
```



```
 153         {"__spin_unlock_bh",            UNLOCK, "spin_lock", 0, ret_any},

 155         {"spin_trylock",                LOCK,   "spin_lock", 0, ret_one},
 156         {"_spin_trylock",               LOCK,   "spin_lock", 0, ret_one},
 157         {"__spin_trylock",              LOCK,   "spin_lock", 0, ret_one},
 158         {"raw_spin_trylock",            LOCK,   "spin_lock", 0, ret_one},
 159         {"_raw_spin_trylock",           LOCK,   "spin_lock", 0, ret_one},
 160         {"spin_trylock_irq",            LOCK,   "spin_lock", 0, ret_one},
 161         {"spin_trylock_irqsave",        LOCK,   "spin_lock", 0, ret_one},
 162         {"spin_trylock_bh",             LOCK,   "spin_lock", 0, ret_one},
 163         {"_spin_trylock_bh",            LOCK,   "spin_lock", 0, ret_one},
 164         {"__spin_trylock_bh",           LOCK,   "spin_lock", 0, ret_one},
 165         {"__raw_spin_trylock",          LOCK,   "spin_lock", 0, ret_one},
 166         {"_atomic_dec_and_lock",        LOCK,   "spin_lock", 1, ret_one},
 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},

 168         {"read_lock",                   LOCK,   "read_lock", 0, ret_any},
 169         {"read_unlock",                 UNLOCK, "read_lock", 0, ret_any},
 170         {"_read_lock",                  LOCK,   "read_lock", 0, ret_any},
 171         {"_read_unlock",                UNLOCK, "read_lock", 0, ret_any},
 172         {"__read_lock",                 LOCK,   "read_lock", 0, ret_any},
 173         {"__read_unlock",               UNLOCK, "read_lock", 0, ret_any},
 174         {"_raw_read_lock",              LOCK,   "read_lock", 0, ret_any},
 175         {"_raw_read_unlock",            UNLOCK, "read_lock", 0, ret_any},
 176         {"__raw_read_lock",             LOCK,   "read_lock", 0, ret_any},
 177         {"__raw_read_unlock",           UNLOCK, "read_lock", 0, ret_any},
 178         {"read_lock_irq",               LOCK,   "read_lock", 0, ret_any},
 179         {"read_unlock_irq" ,            UNLOCK, "read_lock", 0, ret_any},
 180         {"_read_lock_irq",              LOCK,   "read_lock", 0, ret_any},
 181         {"_read_unlock_irq",            UNLOCK, "read_lock", 0, ret_any},
 182         {"__read_lock_irq",             LOCK,   "read_lock", 0, ret_any},
 183         {"__read_unlock_irq",           UNLOCK, "read_lock", 0, ret_any},
 184         {"read_lock_irqsave",           LOCK,   "read_lock", 0, ret_any},
 185         {"read_unlock_irqrestore",      UNLOCK, "read_lock", 0, ret_any},
 186         {"_read_lock_irqsave",          LOCK,   "read_lock", 0, ret_any},
 187         {"_read_unlock_irqrestore",     UNLOCK, "read_lock", 0, ret_any},
 188         {"__read_lock_irqsave",         LOCK,   "read_lock", 0, ret_any},
 189         {"__read_unlock_irqrestore",    UNLOCK, "read_lock", 0, ret_any},
 190         {"read_lock_bh",                LOCK,   "read_lock", 0, ret_any},
 191         {"read_unlock_bh",              UNLOCK, "read_lock", 0, ret_any},
 192         {"_read_lock_bh",               LOCK,   "read_lock", 0, ret_any},
 193         {"_read_unlock_bh",             UNLOCK, "read_lock", 0, ret_any},
 194         {"__read_lock_bh",              LOCK,   "read_lock", 0, ret_any},
 195         {"__read_unlock_bh",            UNLOCK, "read_lock", 0, ret_any},
 196         {"_raw_read_lock_bh",           LOCK,   "read_lock", 0, ret_any},
 197         {"_raw_read_unlock_bh",         UNLOCK, "read_lock", 0, ret_any},
 198         {"__raw_read_lock_bh",          LOCK,   "read_lock", 0, ret_any},
 199         {"__raw_read_unlock_bh",        UNLOCK, "read_lock", 0, ret_any},

 201         {"generic__raw_read_trylock", LOCK,   "read_lock", 0, ret_one},
 202         {"read_trylock",                LOCK,   "read_lock", 0, ret_one},
 203         {"_read_trylock",               LOCK,   "read_lock", 0, ret_one},
 204         {"raw_read_trylock",            LOCK,   "read_lock", 0, ret_one},
 205         {"_raw_read_trylock",           LOCK,   "read_lock", 0, ret_one},
 206         {"__raw_read_trylock",          LOCK,   "read_lock", 0, ret_one},
```

```
207         {"__read_trylock",              LOCK,   "read_lock", 0, ret_one},
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},

209         {"write_lock",                LOCK,   "write_lock", 0, ret_any},
210         {"write_unlock",              UNLOCK, "write_lock", 0, ret_any},
211         {"_write_lock",               LOCK,   "write_lock", 0, ret_any},
212         {"_write_unlock",             UNLOCK, "write_lock", 0, ret_any},
213         {"__write_lock",              LOCK,   "write_lock", 0, ret_any},
214         {"__write_unlock",            UNLOCK, "write_lock", 0, ret_any},
215         {"write_lock_irq",            LOCK,   "write_lock", 0, ret_any},
216         {"write_unlock_irq",          UNLOCK, "write_lock", 0, ret_any},
217         {"_write_lock_irq",           LOCK,   "write_lock", 0, ret_any},
218         {"_write_unlock_irq",         UNLOCK, "write_lock", 0, ret_any},
219         {"__write_lock_irq",          LOCK,   "write_lock", 0, ret_any},
220         {"__write_unlock_irq",        UNLOCK, "write_lock", 0, ret_any},
221         {"write_lock_irqsave",        LOCK,   "write_lock", 0, ret_any},
222         {"write_unlock_irqrestore",   UNLOCK, "write_lock", 0, ret_any},
223         {"_write_lock_irqsave",       LOCK,   "write_lock", 0, ret_any},
224         {"_write_unlock_irqrestore",  UNLOCK, "write_lock", 0, ret_any},
225         {"__write_lock_irqsave",      LOCK,   "write_lock", 0, ret_any},
226         {"__write_unlock_irqrestore", UNLOCK, "write_lock", 0, ret_any},
227         {"write_lock_bh",             LOCK,   "write_lock", 0, ret_any},
228         {"write_unlock_bh",           UNLOCK, "write_lock", 0, ret_any},
229         {"_write_lock_bh",            LOCK,   "write_lock", 0, ret_any},
230         {"_write_unlock_bh",          UNLOCK, "write_lock", 0, ret_any},
231         {"__write_lock_bh",           LOCK,   "write_lock", 0, ret_any},
232         {"__write_unlock_bh",         UNLOCK, "write_lock", 0, ret_any},
233         {"_raw_write_lock",           LOCK,   "write_lock", 0, ret_any},
234         {"__raw_write_lock",          LOCK,   "write_lock", 0, ret_any},
235         {"_raw_write_unlock",         UNLOCK, "write_lock", 0, ret_any},
236         {"__raw_write_unlock",        UNLOCK, "write_lock", 0, ret_any},

238         {"write_trylock",             LOCK,   "write_lock", 0, ret_one},
239         {"_write_trylock",            LOCK,   "write_lock", 0, ret_one},
240         {"raw_write_trylock",         LOCK,   "write_lock", 0, ret_one},
241         {"_raw_write_trylock",        LOCK,   "write_lock", 0, ret_one},
242         {"__write_trylock",           LOCK,   "write_lock", 0, ret_one},
243         {"__raw_write_trylock",       LOCK,   "write_lock", 0, ret_one},
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},

245         {"down",              LOCK,   "sem", 0, ret_any},
246         {"up",                UNLOCK, "sem", 0, ret_any},
247         {"down_trylock",      LOCK,   "sem", 0, ret_zero},
248         {"down_timeout",      LOCK,   "sem", 0, ret_zero},
249         {"down_interruptible", LOCK,  "sem", 0, ret_zero},


252         {"down_write",          LOCK,   "rw_sem", 0, ret_any},
253         {"downgrade_write",     UNLOCK, "rw_sem", 0, ret_any},
254         {"downgrade_write",     LOCK,   "read_sem", 0, ret_any},
255         {"up_write",            UNLOCK, "rw_sem", 0, ret_any},
256         {"down_write_trylock",  LOCK,   "rw_sem", 0, ret_one},
257         {"down_write_killable", LOCK,   "rw_sem", 0, ret_zero},
258         {"down_read",           LOCK,   "read_sem", 0, ret_any},
259         {"down_read_trylock",   LOCK,   "read_sem", 0, ret_one},
```



```
260         {"down_read_killable",  LOCK,   "read_sem", 0, ret_zero},
261         {"up_read",             UNLOCK, "read_sem", 0, ret_any},

263         {"mutex_lock",                      LOCK,   "mutex", 0, ret_any},
264         {"mutex_lock_io",                   LOCK,   "mutex", 0, ret_any},
265         {"mutex_unlock",                    UNLOCK, "mutex", 0, ret_any},
266         {"mutex_lock_nested",               LOCK,   "mutex", 0, ret_any},
267         {"mutex_lock_io_nested",            LOCK,   "mutex", 0, ret_any},

269         {"mutex_lock_interruptible",        LOCK,   "mutex", 0, ret_zero},
270         {"mutex_lock_interruptible_nested", LOCK,   "mutex", 0, ret_zero},
271         {"mutex_lock_killable",             LOCK,   "mutex", 0, ret_zero},
272         {"mutex_lock_killable_nested",      LOCK,   "mutex", 0, ret_zero},

274         {"mutex_trylock",                   LOCK,   "mutex", 0, ret_one},
259         {"mutex_trylock",                   LOCK,   "mutex", 0, ret_non_zero},

276         {"raw_local_irq_disable", LOCK,   "irq", NO_ARG, ret_any},
277         {"raw_local_irq_enable",  UNLOCK, "irq", NO_ARG, ret_any},
278         {"spin_lock_irq",         LOCK,   "irq", NO_ARG, ret_any},
279         {"spin_unlock_irq",       UNLOCK, "irq", NO_ARG, ret_any},
280         {"_spin_lock_irq",        LOCK,   "irq", NO_ARG, ret_any},
281         {"_spin_unlock_irq",      UNLOCK, "irq", NO_ARG, ret_any},
282         {"__spin_lock_irq",       LOCK,   "irq", NO_ARG, ret_any},
283         {"__spin_unlock_irq",     UNLOCK, "irq", NO_ARG, ret_any},
284         {"_raw_spin_lock_irq",    LOCK,   "irq", NO_ARG, ret_any},
285         {"_raw_spin_unlock_irq",  UNLOCK, "irq", NO_ARG, ret_any},
286         {"__raw_spin_unlock_irq", UNLOCK, "irq", NO_ARG, ret_any},
287         {"spin_trylock_irq",      LOCK,   "irq", NO_ARG, ret_one},
272         {"spin_trylock_irq",      LOCK,   "irq", NO_ARG, ret_non_zero},
288         {"read_lock_irq",         LOCK,   "irq", NO_ARG, ret_any},
289         {"read_unlock_irq",       UNLOCK, "irq", NO_ARG, ret_any},
290         {"_read_lock_irq",        LOCK,   "irq", NO_ARG, ret_any},
291         {"_read_unlock_irq",      UNLOCK, "irq", NO_ARG, ret_any},
292         {"__read_lock_irq",       LOCK,   "irq", NO_ARG, ret_any},
293         {"__read_unlock_irq",     UNLOCK, "irq", NO_ARG, ret_any},
294         {"write_lock_irq",        LOCK,   "irq", NO_ARG, ret_any},
295         {"write_unlock_irq",      UNLOCK, "irq", NO_ARG, ret_any},
296         {"_write_lock_irq",       LOCK,   "irq", NO_ARG, ret_any},
297         {"_write_unlock_irq",     UNLOCK, "irq", NO_ARG, ret_any},
298         {"__write_lock_irq",      LOCK,   "irq", NO_ARG, ret_any},
299         {"__write_unlock_irq",    UNLOCK, "irq", NO_ARG, ret_any},

301         {"arch_local_irq_save",         LOCK,   "irqsave", RETURN_VAL, ret_any},
302         {"arch_local_irq_restore",      UNLOCK, "irqsave", 0, ret_any},
303         {"__raw_local_irq_save",        LOCK,   "irqsave", RETURN_VAL, ret_any},
304         {"raw_local_irq_restore",       UNLOCK, "irqsave", 0, ret_any},
305         {"spin_lock_irqsave_nested",    LOCK,   "irqsave", RETURN_VAL, ret_any},
306         {"spin_lock_irqsave",           LOCK,   "irqsave", RETURN_VAL, ret_any},
307         {"spin_lock_irqsave",           LOCK,   "irqsave", 1, ret_any},
308         {"spin_unlock_irqrestore",      UNLOCK, "irqsave", 1, ret_any},
309         {"_spin_lock_irqsave_nested",   LOCK,   "irqsave", RETURN_VAL, ret_any},
310         {"_spin_lock_irqsave",          LOCK,   "irqsave", RETURN_VAL, ret_any},
311         {"_spin_lock_irqsave",          LOCK,   "irqsave", 1, ret_any},
312         {"_spin_unlock_irqrestore",     UNLOCK, "irqsave", 1, ret_any},
313         {"__spin_lock_irqsave_nested", LOCK,   "irqsave", 1, ret_any},
314         {"__spin_lock_irqsave",         LOCK,   "irqsave", 1, ret_any},
315         {"__spin_unlock_irqrestore",    UNLOCK, "irqsave", 1, ret_any},
316         {"_raw_spin_lock_irqsave",      LOCK,   "irqsave", RETURN_VAL, ret_any},
317         {"_raw_spin_lock_irqsave",      LOCK,   "irqsave", 1, ret_any},
318         {"_raw_spin_unlock_irqrestore",UNLOCK, "irqsave", 1, ret_any},
319         {"__raw_spin_lock_irqsave",     LOCK,   "irqsave", RETURN_VAL, ret_any},
320         {"__raw_spin_unlock_irqrestore",UNLOCK, "irqsave", 1, ret_any},
321         {"_raw_spin_lock_irqsave_nested", LOCK, "irqsave", RETURN_VAL, ret_any},
322         {"spin_trylock_irqsave",        LOCK,   "irqsave", 1, ret_one},
307         {"spin_trylock_irqsave",        LOCK,   "irqsave", 1, ret_non_zero},
```

```
323         {"read_lock_irqsave",           LOCK,   "irqsave", RETURN_VAL, ret_any},
324         {"read_lock_irqsave",           LOCK,   "irqsave", 1, ret_any},
325         {"read_unlock_irqrestore",      UNLOCK, "irqsave", 1, ret_any},
326         {"_read_lock_irqsave",          LOCK,   "irqsave", RETURN_VAL, ret_any},
327         {"_read_lock_irqsave",          LOCK,   "irqsave", 1, ret_any},
328         {"_read_unlock_irqrestore",     UNLOCK, "irqsave", 1, ret_any},
329         {"__read_lock_irqsave",         LOCK,   "irqsave", RETURN_VAL, ret_any},
330         {"__read_unlock_irqrestore",    UNLOCK, "irqsave", 1, ret_any},
331         {"write_lock_irqsave",          LOCK,   "irqsave", RETURN_VAL, ret_any},
332         {"write_lock_irqsave",          LOCK,   "irqsave", 1, ret_any},
333         {"write_unlock_irqrestore",     UNLOCK, "irqsave", 1, ret_any},
334         {"_write_lock_irqsave",         LOCK,   "irqsave", RETURN_VAL, ret_any},
335         {"_write_lock_irqsave",         LOCK,   "irqsave", 1, ret_any},
336         {"_write_unlock_irqrestore",    UNLOCK, "irqsave", 1, ret_any},
337         {"__write_lock_irqsave",        LOCK,   "irqsave", RETURN_VAL, ret_any},
338         {"__write_unlock_irqrestore",   UNLOCK, "irqsave", 1, ret_any},

340         {"local_bh_disable",    LOCK,   "bottom_half", NO_ARG, ret_any},
341         {"_local_bh_disable",   LOCK,   "bottom_half", NO_ARG, ret_any},
342         {"__local_bh_disable",  LOCK,   "bottom_half", NO_ARG, ret_any},
343         {"local_bh_enable",     UNLOCK, "bottom_half", NO_ARG, ret_any},
344         {"_local_bh_enable",    UNLOCK, "bottom_half", NO_ARG, ret_any},
345         {"__local_bh_enable",   UNLOCK, "bottom_half", NO_ARG, ret_any},
346         {"spin_lock_bh",        LOCK,   "bottom_half", NO_ARG, ret_any},
347         {"spin_unlock_bh",      UNLOCK, "bottom_half", NO_ARG, ret_any},
348         {"_spin_lock_bh",       LOCK,   "bottom_half", NO_ARG, ret_any},
349         {"_spin_unlock_bh",     UNLOCK, "bottom_half", NO_ARG, ret_any},
350         {"__spin_lock_bh",      LOCK,   "bottom_half", NO_ARG, ret_any},
351         {"__spin_unlock_bh",    UNLOCK, "bottom_half", NO_ARG, ret_any},
352         {"read_lock_bh",        LOCK,   "bottom_half", NO_ARG, ret_any},
353         {"read_unlock_bh",      UNLOCK, "bottom_half", NO_ARG, ret_any},
354         {"_read_lock_bh",       LOCK,   "bottom_half", NO_ARG, ret_any},
355         {"_read_unlock_bh",     UNLOCK, "bottom_half", NO_ARG, ret_any},
356         {"__read_lock_bh",      LOCK,   "bottom_half", NO_ARG, ret_any},
357         {"__read_unlock_bh",    UNLOCK, "bottom_half", NO_ARG, ret_any},
358         {"_raw_read_lock_bh",   LOCK,   "bottom_half", NO_ARG, ret_any},
359         {"_raw_read_unlock_bh", UNLOCK, "bottom_half", NO_ARG, ret_any},
360         {"write_lock_bh",       LOCK,   "bottom_half", NO_ARG, ret_any},
361         {"write_unlock_bh",     UNLOCK, "bottom_half", NO_ARG, ret_any},
362         {"_write_lock_bh",      LOCK,   "bottom_half", NO_ARG, ret_any},
363         {"_write_unlock_bh",    UNLOCK, "bottom_half", NO_ARG, ret_any},
364         {"__write_lock_bh",     LOCK,   "bottom_half", NO_ARG, ret_any},
365         {"__write_unlock_bh",   UNLOCK, "bottom_half", NO_ARG, ret_any},
366         {"spin_trylock_bh",     LOCK,   "bottom_half", NO_ARG, ret_one},
367         {"_spin_trylock_bh",    LOCK,   "bottom_half", NO_ARG, ret_one},
368         {"__spin_trylock_bh",   LOCK,   "bottom_half", NO_ARG, ret_one},
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},

370         {"ffs_mutex_lock",        LOCK,   "mutex", 0, ret_zero},
371 };
```

*unchanged_portion_omitted_*

```
465 static bool nestable(const char *name)
466 {
467         if (strstr(name, "read_sem:"))
468                 return true;
469         if (strcmp(name, "bottom_half:") == 0)
470                 return true;
471         return false;
472 }

474 static void do_lock(const char *name)
475 {
476         struct sm_state *sm;
```



```
478         if (__inline_fn)
479                 return;

481         sm = get_sm_state(my_id, name, NULL);
482         if (!sm)
483                 add_tracker(&starts_unlocked, my_id, name, NULL);
484         if (sm && slist_has_state(sm->possible, &locked) && !nestable(name))
460         if (sm && slist_has_state(sm->possible, &locked) &&
461                         strcmp(name, "bottom_half:") != 0)
485                 sm_error("double lock '%s'", name);
486         if (sm)
487                 func_has_transition = TRUE;
488         set_state(my_id, name, NULL, &locked);
489 }
```

*unchanged_portion_omitted_*

```
767 static int matches_return_type(struct range_list *rl, enum return_type type)
768 {
769         sval_t zero_sval = ll_to_sval(0);
770         sval_t one_sval = ll_to_sval(1);

772         /* All these double negatives are super ugly!  */

774         switch (type) {
775         case ret_zero:
776                 return !possibly_true_rl(rl, SPECIAL_NOTEQUAL, alloc_rl(zero_sva
777         case ret_one:
778                 return !possibly_true_rl(rl, SPECIAL_NOTEQUAL, alloc_rl(one_sval
779         case ret_non_zero:
780                 return !possibly_true_rl(rl, SPECIAL_EQUAL, alloc_rl(zero_sval,
781         case ret_negative:
782                 return !possibly_true_rl(rl, SPECIAL_GTE, alloc_rl(zero_sval, ze
783         case ret_positive:
784                 return !possibly_true_rl(rl, '<', alloc_rl(zero_sval, zero_sval)
785         case ret_any:
786         default:
787                 return 1;
788         }
789 }
```

*unchanged_portion_omitted_*

```
929 static void register_lock(int index)
930 {
931         struct lock_info *lock = &lock_table[index];
932         void *idx = INT_PTR(index);

934         if (lock->return_type == ret_non_zero) {
935                 return_implies_state(lock->function, 1, INT_MAX, &match_lock_hel
909                 return_implies_state(lock->function, valid_ptr_min, valid_ptr_ma
936                 return_implies_state(lock->function, 0, 0, &match_lock_failed, i
937         } else if (lock->return_type == ret_any && lock->arg == RETURN_VAL) {
938                 add_function_assign_hook(lock->function, &match_returns_locked,
939         } else if (lock->return_type == ret_any) {
940                 add_function_hook(lock->function, &match_lock_unlock, idx);
941         } else if (lock->return_type == ret_zero) {
942                 return_implies_state(lock->function, 0, 0, &match_lock_held, idx
943                 return_implies_state(lock->function, -4095, -1, &match_lock_fail
944         } else if (lock->return_type == ret_one) {
945                 return_implies_state(lock->function, 1, 1, &match_lock_held, idx
946                 return_implies_state(lock->function, 0, 0, &match_lock_failed, i
947         }
948 }
```

*unchanged_portion_omitted_*

```
**********************************************************
    3706 Mon Aug  5 08:38:01 2019
new/usr/src/tools/smatch/src/check_macro_side_effects.c
11506 smatch resync
**********************************************************
```

```
   1 /*
   2  * Copyright (C) 2012 Oracle.
   3  *
   4  * This program is free software; you can redistribute it and/or
   5  * modify it under the terms of the GNU General Public License
   6  * as published by the Free Software Foundation; either version 2
   7  * of the License, or (at your option) any later version.
   8  *
   9  * This program is distributed in the hope that it will be useful,
  10  * but WITHOUT ANY WARRANTY; without even the implied warranty of
  11  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  12  * GNU General Public License for more details.
  13  *
  14  * You should have received a copy of the GNU General Public License
  15  * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
  16  */

  18 #include "scope.h"
  19 #include "smatch.h"
  20 #include "smatch_slist.h"
  21 #include "smatch_expression_stacks.h"

  23 static int my_id;

  25 static struct string_list *ignored_macros;
  26 static struct position old_pos;

  28 static struct smatch_state *alloc_my_state(struct expression *expr)
  29 {
  30         struct smatch_state *state;
  31         char *name;

  33         state = __alloc_smatch_state(0);
  33         expr = strip_expr(expr);
  34         name = expr_to_str(expr);
  35         if (!name)
  36                 return NULL;

  38         state = __alloc_smatch_state(0);
  39         state->name = alloc_sname(name);
  40         free_string(name);
  41         state->data = expr;
  42         return state;
  43 }
_____unchanged_portion_omitted_

 159 void check_macro_side_effects(int id)
 160 {
 161         my_id = id;

 163         if (!option_spammy)
 164                 return;

 166         set_dynamic_states(my_id);
 167         add_hook(&match_unop, OP_HOOK);
 168         add_hook(&match_stmt, STMT_HOOK);
 169         register_ignored_macros();
 170 }
_____unchanged_portion_omitted_
```

****
4313 Mon Aug 5 08:38:01 2019
new/usr/src/tools/smatch/src/check_missing_break.c
11506 smatch resync
****
_____***unchanged_portion_omitted_***

```
 173 void check_missing_break(int id)
 174 {
 175         my_id = id;

 177         if (!option_spammy)
 178                 return;

 180         set_dynamic_states(my_id);
 181         add_unmatched_state_hook(my_id, &unmatched_state);
 182         add_merge_hook(my_id, &merge_hook);

 184         add_hook(&match_assign, ASSIGNMENT_HOOK);
 185         add_hook(&match_symbol, SYM_HOOK);
 186         add_hook(&match_stmt, STMT_HOOK);
 187         add_hook(&match_switch, STMT_HOOK);
 188         add_hook(&match_switch_end, STMT_HOOK_AFTER);
 189 }
```

_____***unchanged_portion_omitted_***

****
1292 Mon Aug 5 08:38:02 2019
new/usr/src/tools/smatch/src/check_no_return.c
11506 smatch resync
****

```
 1 /*
 2  * Copyright (C) 2010 Dan Carpenter.
 3  *
 4  * This program is free software; you can redistribute it and/or
 5  * modify it under the terms of the GNU General Public License
 6  * as published by the Free Software Foundation; either version 2
 7  * of the License, or (at your option) any later version.
 8  *
 9  * This program is distributed in the hope that it will be useful,
10  * but WITHOUT ANY WARRANTY; without even the implied warranty of
11  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
12  * GNU General Public License for more details.
13  *
14  * You should have received a copy of the GNU General Public License
15  * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
16  */

18 #include "smatch.h"
19 #include "smatch_slist.h"

21 static int my_id;
22 static int returned;

24 static void match_return(struct expression *ret_value)
25 {
26         if (__inline_fn)
27                 return;
28         if (is_reachable())
29                 returned = 1;
30 }

32 static void match_func_end(struct symbol *sym)
33 {
34         if (__inline_fn)
35                 return;
36         if (out_of_memory() || taking_too_long())
37                 return;
38         if (!is_reachable() && !returned)
39                 sm_info("info: add to no_return_funcs");
40         returned = 0;
41 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    6705 Mon Aug  5 08:38:02 2019
new/usr/src/tools/smatch/src/check_nospec.c
11506 smatch resync
**********************************************************
______unchanged_portion_omitted_

  97 static void returned_struct_members(int return_id, char *return_ranges, struct e
  98 {
  99         struct stree *start_states = get_start_states();
 100         struct symbol *returned_sym;
 101         struct sm_state *sm;
 102         const char *param_name;
 103         struct range_list *rl;
 104         int param;

 106         returned_sym = expr_to_sym(expr);

 108         FOR_EACH_MY_SM(my_id, __get_cur_stree(), sm) {
 109                 if (get_state_stree(start_states, my_id, sm->name, sm->sym) == s
 110                         continue;
 111                 param = get_param_num_from_sym(sm->sym);
 112                 if (param < 0) {
 113                         if (!returned_sym || returned_sym != sm->sym)
 114                                 continue;
 115                         param = -1;
 116                 }

 118                 param_name = get_param_name(sm);
 119                 if (!param_name)
 120                         continue;
 121                 if (param != -1 && strcmp(param_name, "$") == 0)
 122                         continue;

 124                 if (!get_user_rl_var_sym(sm->name, sm->sym, &rl))
 125                         continue;

 127                 sql_insert_return_states(return_id, return_ranges, NOSPEC, param
 128         } END_FOR_EACH_SM(sm);

 130         if (is_nospec(expr) && get_user_rl(expr, &rl))
 131                 sql_insert_return_states(return_id, return_ranges, NOSPEC, -1, "

 133         if (get_state(barrier_id, "barrier", NULL) == &nospec)
 134                 sql_insert_return_states(return_id, return_ranges, NOSPEC_WB, -1
 135 }
______unchanged_portion_omitted_

 219 static void match_barrier(struct statement *stmt)
 220 {
 221         char *macro;

 223         macro = get_macro_name(stmt->pos);
 224         if (!macro)
 225                 return;
 226         if (strcmp(macro, "rmb") != 0 &&
 227             strcmp(macro, "smp_rmb") != 0 &&
 228             strcmp(macro, "barrier_nospec") != 0 &&
 229             strcmp(macro, "preempt_disable") != 0)
 225             strcmp(macro, "barrier_nospec") != 0)
 230                 return;

 232         set_state(barrier_id, "barrier", NULL, &nospec);
 233         mark_user_data_as_nospec();
 234 }
______unchanged_portion_omitted_
```



```
 241 static void select_return_stmt_cnt(struct expression *expr, int param, char *key
 242 {
 243         int cnt;

 245         cnt = atoi(value);
 246         if (cnt > 400)
 247                 mark_user_data_as_nospec();
 248 }

 250 void check_nospec(int id)
 251 {
 252         my_id = id;

 254         add_hook(&nospec_assign, ASSIGNMENT_HOOK);

 256         select_caller_info_hook(set_param_nospec, NOSPEC);
 257         add_unmatched_state_hook(my_id, &unmatched_state);

 259         add_hook(&match_call_info, FUNCTION_CALL_HOOK);
 260         add_member_info_callback(my_id, struct_member_callback);
 261         add_split_return_callback(&returned_struct_members);
 262         select_return_states_hook(NOSPEC, &db_returns_nospec);
 263         select_return_states_hook(NOSPEC_WB, &db_returns_barrier);
 264         select_return_states_hook(STMT_CNT, &select_return_stmt_cnt);

 266         add_hook(&match_asm, ASM_HOOK);
 267         add_hook(&match_after_nospec_asm, STMT_HOOK_AFTER);
 268 }
______unchanged_portion_omitted_
```

```
**********************************************************
    3372 Mon Aug  5 08:38:02 2019
new/usr/src/tools/smatch/src/check_off_by_one_relative.c
11506 smatch resync
**********************************************************
```

```
 1 /*
 2  * Copyright (C) 2015 Oracle.
 3  *
 4  * This program is free software; you can redistribute it and/or
 5  * modify it under the terms of the GNU General Public License
 6  * as published by the Free Software Foundation; either version 2
 7  * of the License, or (at your option) any later version.
 8  *
 9  * This program is distributed in the hope that it will be useful,
10  * but WITHOUT ANY WARRANTY; without even the implied warranty of
11  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
12  * GNU General Public License for more details.
13  *
14  * You should have received a copy of the GNU General Public License
15  * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
16  */

18 /*
19  * The point here is to store that a buffer has x bytes even if we don't know
20  * the value of x.
21  *
22  */

24 #include "smatch.h"
25 #include "smatch_slist.h"
26 #include "smatch_extra.h"

28 static int my_id;

30 static void array_check(struct expression *expr)
31 {
32         struct expression *array;
33         struct expression *size;
34         struct expression *offset;
35         char *array_str, *offset_str;
36         int limit_type;

38         expr = strip_expr(expr);
39         if (!is_array(expr))
40                 return;

42         array = get_array_base(expr);
43         size = get_size_variable(array, &limit_type);
44         if (!size || limit_type != ELEM_COUNT)
42         size = get_size_variable(array);
43         if (!size)
45                 return;
46         offset = get_array_offset(expr);
47         if (!possible_comparison(size, SPECIAL_EQUAL, offset))
48                 return;

50         if (buf_comparison_index_ok(expr))
51                 return;

53         array_str = expr_to_str(array);
54         offset_str = expr_to_str(offset);
55         sm_warning("potentially one past the end of array '%s[%s]'", array_str,
56         free_string(array_str);
57         free_string(offset_str);
58 }
```



```
56 static int known_access_ok_comparison(struct expression *expr)
57 {
58         struct expression *array;
59         struct expression *size;
60         struct expression *offset;
61         int comparison;

63         array = get_array_base(expr);
64         size = get_size_variable(array);
65         if (!size)
66                 return 0;
67         offset = get_array_offset(expr);
68         comparison = get_comparison(size, offset);
69         if (comparison == '>' || comparison == SPECIAL_UNSIGNED_GT)
70                 return 1;

72         return 0;
73 }
```

```
60 static int known_access_ok_numbers(struct expression *expr)
61 {
62         struct expression *array;
63         struct expression *offset;
64         sval_t max;
65         int size;

67         array = get_array_base(expr);
68         offset = get_array_offset(expr);

70         size = get_array_size(array);
71         if (size <= 0)
72                 return 0;

74         get_absolute_max(offset, &max);
75         if (max.uvalue < size)
76                 return 1;
77         return 0;
78 }

80 static void array_check_data_info(struct expression *expr)
81 {
82         struct expression *array;
83         struct expression *offset;
84         struct state_list *slist;
85         struct sm_state *sm;
86         struct compare_data *comp;
87         char *offset_name;
88         const char *equal_name = NULL;

90         expr = strip_expr(expr);
91         if (!is_array(expr))
92                 return;

94         if (known_access_ok_numbers(expr))
95                 return;
96         if (buf_comparison_index_ok(expr))
111        if (known_access_ok_comparison(expr))
97                 return;

99         array = get_array_base(expr);
100        offset = get_array_offset(expr);
101        offset_name = expr_to_var(offset);
102        if (!offset_name)
103                return;
104        slist = get_all_possible_equal_comparisons(offset);
105        if (!slist)
```

```
 106                 goto free;

 108         FOR_EACH_PTR(slist, sm) {
 109                 comp = sm->state->data;
 110                 if (strcmp(comp->left_var, offset_name) == 0) {
 111                         if (db_var_is_array_limit(array, comp->right_var, comp->
 112                                 equal_name = comp->right_var;
 113                                 break;
 114                         }
 115                 } else if (strcmp(comp->right_var, offset_name) == 0) {
 116                         if (db_var_is_array_limit(array, comp->left_var, comp->l
 117                                 equal_name = comp->left_var;
 118                                 break;
 119                         }
 120                 }
 121         } END_FOR_EACH_PTR(sm);

 123         if (equal_name) {
 124                 char *array_name = expr_to_str(array);

 126                 sm_warning("potential off by one '%s[]' limit '%s'", array_name,
 127                 free_string(array_name);
 128         }

 130 free:
 131         free_slist(&slist);
 132         free_string(offset_name);
 133 }
```

*unchanged_portion_omitted_*

****
3526 Mon Aug 5 08:38:03 2019
new/usr/src/tools/smatch/src/check_precedence.c
11506 smatch resync
****
_____***unchanged_portion_omitted_***

```
123 static void match_mask_compare(struct expression *expr)
124 {
125         if (expr->op != '&')
126                 return;
127         if (expr->right->type != EXPR_COMPARE)
128                 return;
129
130         sm_warning("compare has higher precedence than mask");
131 }
132
133 static void match_subtract_shift(struct expression *expr)
134 {
135         if (expr->op != SPECIAL_LEFTSHIFT)
136                 return;
137         if (expr->right->type != EXPR_BINOP)
138                 return;
139         if (expr->right->op != '-')
140                 return;
141         sm_warning("subtract is higher precedence than shift");
142 }
143
144 void check_precedence(int id)
145 {
146         my_id = id;
147
148         add_hook(&match_condition, CONDITION_HOOK);
149         add_hook(&match_binop, BINOP_HOOK);
150         add_hook(&match_mask, BINOP_HOOK);
151         add_hook(&match_mask_compare, BINOP_HOOK);
152         add_hook(&match_subtract_shift, BINOP_HOOK);
153 }
```

_____***unchanged_portion_omitted_***

****
1418 Mon Aug 5 08:38:03 2019
new/usr/src/tools/smatch/src/check_return_cast.c
11506 smatch resync
****

```
/*
 * 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 || func_type->type != SYM_FN)
                return;
        func_type = get_real_base_type(func_type);
        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));
}
```

*_____unchanged_portion_omitted_*

****
9033 Mon Aug 5 08:38:04 2019
new/usr/src/tools/smatch/src/check_rosenberg.c
11506 smatch resync
****
_____***unchanged_portion_omitted_***

```
 379 void check_rosenberg2(int id)
 380 {
 381         if (option_project != PROJ_KERNEL)
 382                 return;

 384         my_member_id = id;
 385         set_dynamic_states(my_member_id);
 386         add_extra_mod_hook(&extra_mod_hook);
 387 }
```

_____***unchanged_portion_omitted_***

****
2212 Mon Aug 5 08:38:04 2019
new/usr/src/tools/smatch/src/check_shift_to_zero.c
11506 smatch resync
****

```
/*
 * 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;
        if (is_ignored_expr(my_id, expr))
                return;
        sm_warning("right shifting more than type allows %d vs %lld", type_bits(
}
```

_____***unchanged_portion_omitted_***

****
2292 Mon Aug 5 08:38:05 2019
new/usr/src/tools/smatch/src/check_snprintf.c
11506 smatch resync
****
_____***unchanged_portion_omitted_***

```
 74 void check_snprintf(int id)
 75 {
 76         if (option_project != PROJ_KERNEL)
 77                 return;
 78         if (!option_spammy)
 79                 return;

 81         my_id = id;
 82         set_dynamic_states(my_id);
 83         add_hook(&match_call, FUNCTION_CALL_HOOK);
 84         add_function_assign_hook("snprintf", &match_snprintf, NULL);
 85         add_modification_hook(my_id, &ok_to_use);
 86 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    4707 Mon Aug  5 08:38:06 2019
new/usr/src/tools/smatch/src/check_spectre.c
11506 smatch resync
**********************************************************
```

```
   1 /*
   2  * Copyright (C) 2018 Oracle.
   3  *
   4  * This program is free software; you can redistribute it and/or
   5  * modify it under the terms of the GNU General Public License
   6  * as published by the Free Software Foundation; either version 2
   7  * of the License, or (at your option) any later version.
   8  *
   9  * This program is distributed in the hope that it will be useful,
  10  * but WITHOUT ANY WARRANTY; without even the implied warranty of
  11  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  12  * GNU General Public License for more details.
  13  *
  14  * You should have received a copy of the GNU General Public License
  15  * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
  16  */

  18 #include "smatch.h"
  19 #include "smatch_extra.h"

  21 static int my_id;
  22 extern int second_half_id;
  23 extern void set_spectre_first_half(struct expression *expr);

  25 static int suppress_multiple = 1;

  27 static int is_write(struct expression *expr)
  28 {
  29         return 0;
  30 }
______unchanged_portion_omitted_

 152 static void array_check(struct expression *expr)
 153 {
 154         struct expression_list *conditions;
 155         struct expression *array_expr, *offset;
 156         unsigned long long mask;
 157         int array_size;
 158         char *name;

 160         expr = strip_expr(expr);
 161         if (!is_array(expr))
 162                 return;

 164         if (is_impossible_path())
 165                 return;
 166         if (is_harmless(expr))
 167                 return;

 169         array_expr = get_array_base(expr);
 170         if (suppress_multiple && is_ignored_expr(my_id, array_expr)) {
 171                 set_spectre_first_half(expr);
 168         if (suppress_multiple && is_ignored_expr(my_id, array_expr))
 172                 return;
 173         }

 175         offset = get_array_offset(expr);
 176         if (!is_user_rl(offset))
 177                 return;
 178         if (is_nospec(offset))
 179                 return;
```



```
 181         array_size = get_array_size(array_expr);
 182         if (array_size > 0 && get_max_by_type(offset) < array_size)
 183                 return;
 184 //      binfo = get_bit_info(offset);
 185 //      if (array_size > 0 && binfo && binfo->possible < array_size)
 186 //              return;

 188         mask = get_mask(offset);
 189         if (mask <= array_size)
 190                 return;

 192         conditions = get_conditions(offset);

 194         name = expr_to_str(array_expr);
 195         sm_warning("potential spectre issue '%s' [%s]%s",
 196                 name,
 197                 is_read(expr) ? "r" : "w",
 198                 conditions ? " (local cap)" : "");

 200         set_spectre_first_half(expr);
 201         if (suppress_multiple)
 202                 add_ignore_expr(my_id, array_expr);
 203         free_string(name);
 204 }
______unchanged_portion_omitted_
```

```
**********************************************************
    2926 Mon Aug  5 08:38:06 2019
new/usr/src/tools/smatch/src/check_spectre_second_half.c
11506 smatch resync
**********************************************************
```

```
/*
 * 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"
#include "smatch_slist.h"

/* New chips will probably be able to speculate further ahead */
#define MAX_SPEC_STMT 200

static int my_id;

struct stree *first_halfs;

struct expression *recently_set;

void set_spectre_first_half(struct expression *expr)
{
        char buf[64];
        char *name;

        name = expr_to_str(expr);
        snprintf(buf, sizeof(buf), "%p %s", expr, name);
        free_string(name);

        set_state_stree(&first_halfs, my_id, buf, NULL, alloc_state_num(get_stmt
}

void clear_spectre_second_halfs(void)
{
        struct sm_state *sm;

        FOR_EACH_MY_SM(my_id, __get_cur_stree(), sm) {
                set_state(my_id, sm->name, sm->sym, alloc_state_num(-MAX_SPEC_ST
        } END_FOR_EACH_SM(sm);
}

static struct smatch_state *get_spectre_first_half(struct expression *expr)
{
        char buf[64];
        char *name;

        name = expr_to_str(expr);
        snprintf(buf, sizeof(buf), "%p %s", expr, name);
        free_string(name);

        return get_state_stree(first_halfs, my_id, buf, NULL);
```



```
}

static void match_assign(struct expression *expr)
{
        struct smatch_state *state;

        if (expr->op == SPECIAL_AND_ASSIGN)
                return;

        state = get_spectre_first_half(expr->right);
        if (state) {
                set_state_expr(my_id, expr->left, state);
                recently_set = expr->left;
                return;
        }
        state = get_state_expr(my_id, expr->right);
        if (!state)
                return;
        set_state_expr(my_id, expr->left, state);
        recently_set = expr->left;
}

static void match_done(struct expression *expr)
{
        struct smatch_state *state;
        char *name;

        if (expr == recently_set)
                return;

        state = get_state_expr(my_id, expr);
        if (!state)
                return;

        if (get_stmt_cnt() - (long)state->data > MAX_SPEC_STMT)
                return;

        name = expr_to_str(expr);
        sm_msg("warn: possible spectre second half.  '%s'", name);
        free_string(name);

        set_state_expr(my_id, expr, alloc_state_num(-MAX_SPEC_STMT));
}

static void match_end_func(struct symbol *sym)
{
        if (__inline_fn)
                return;
        free_stree(&first_halfs);
}

void check_spectre_second_half(int id)
{
        my_id = id;

        if (option_project != PROJ_KERNEL)
                return;
        set_dynamic_states(my_id);
        add_hook(&match_assign, ASSIGNMENT_HOOK);
        add_hook(&match_done, SYM_HOOK);
        add_hook(&match_done, DEREF_HOOK);

        add_hook(&match_end_func, END_FUNC_HOOK);
}
```

```
**********************************************************
    5290 Mon Aug  5 08:38:06 2019
new/usr/src/tools/smatch/src/check_string_len.c
11506 smatch resync
**********************************************************
______unchanged_portion_omitted_

 36 struct param_info zero_one = {0, 1};

 38 static int handle_format(struct expression *call, char **pp, int *arg_nr, bool u
 38 static int handle_format(struct expression *call, char **pp, int *arg_nr)
 39 {
 40         struct expression *arg;
 41         char *p = *pp;
 42         int ret = 1;
 43         char buf[256];
 44         sval_t sval;
 44         sval_t max;

 46         p++;  /* we passed it with *p == '%' */

 48         if (*p == '%') {
 49                 p++;
 50                 ret = 1;
 51                 goto out_no_arg;
 52         }
 53         if (*p == 'c') {
 54                 p++;
 55                 ret = 1;
 56                 goto out;
 57         }


 60         if (isdigit(*p) || *p == '.') {
 61                 unsigned long num;

 63                 if (*p == '.')
 64                         p++;

 66                 num = strtoul(p, &p, 10);
 67                 ret = num;

 69                 while (*p == 'l')
 70                         p++;
 71                 p++; /* eat the 'd' char */
 72                 goto out;
 73         }

 75         if (*p == 'l') {
 76                 p++;
 77                 if (*p == 'l')
 78                         p++;
 79         }

 81         if (option_project == PROJ_KERNEL && *p == 'z')
 82                 p++;

 84         if (option_project == PROJ_KERNEL && *p == 'p') {
 85                 if (*(p + 1) == 'I' || *(p + 1) == 'i') {
 86                         char *eye;

 88                         eye = p + 1;
 89                         p += 2;
 90                         if (*p == 'h' || *p == 'n' || *p == 'b' || *p == 'l')
 91                                 p++;
 92                         if (*p == '4') {
```



```
 93                                 p++;
 94                                 ret = 15;
 95                                 goto out;
 96                         }
 97                         if (*p == '6') {
 98                                 p++;
 99                                 if (*p == 'c')
100                                         p++;
101                                 if (*eye == 'I')
102                                         ret = 39;
103                                 if (*eye == 'i')
104                                         ret = 32;
105                                 goto out;
106                         }
107                 }
108                 if (*(p + 1) == 'M') {
109                         p += 2;
110                         if (*p == 'R' || *p == 'F')
111                                 p++;
112                         ret = 17;
113                         goto out;
114                 }
115                 if (*(p + 1) == 'm') {
116                         p += 2;
117                         if (*p == 'R')
118                                 p++;
119                         ret = 12;
120                         goto out;
121                 }
122         }

124         arg = get_argument_from_call_expr(call->args, *arg_nr);
125         if (!arg)
126                 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         }

138         if (*p != 'd' && *p != 'i' && *p != 'x' && *p != 'X' && *p != 'u' && *p
139                 ret = 1;
140                 p++;
141                 goto out;
142         }

144         if (use_max) {
145                 get_absolute_max(arg, &sval);
146         } else {
147                 get_absolute_min(arg, &sval);
148                 if (sval_is_negative(sval))
149                         sval.value = 0;
150         }
144         get_absolute_max(arg, &max);


153         if (*p == 'x' || *p == 'X' || *p == 'p') {
154                 ret = snprintf(buf, sizeof(buf), "%llx", sval.uvalue);
147                 ret = snprintf(buf, sizeof(buf), "%llx", max.uvalue);
155         } else if (*p == 'u') {
156                 ret = snprintf(buf, sizeof(buf), "%llu", sval.uvalue);
```

```
149                 ret = snprintf(buf, sizeof(buf), "%llu", max.uvalue);
157         } else if (!expr_unsigned(arg)) {
158                 sval_t min;
159                 int tmp;

161                 ret = snprintf(buf, sizeof(buf), "%lld", sval.value);
154                 ret = snprintf(buf, sizeof(buf), "%lld", max.value);
162                 get_absolute_min(arg, &min);
163                 tmp = snprintf(buf, sizeof(buf), "%lld", min.value);
164                 if (tmp > ret)
165                         ret = tmp;
166         } else {
167                 ret = snprintf(buf, sizeof(buf), "%lld", sval.value);
160                 ret = snprintf(buf, sizeof(buf), "%lld", max.value);
168         }
169         p++;

171 out:
172         (*arg_nr)++;
173 out_no_arg:
174         *pp = p;
175         return ret;
176 }

178 int get_formatted_string_size_helper(struct expression *call, int arg, bool use_
171 int get_formatted_string_size(struct expression *call, int arg)
179 {
180         struct expression *expr;
181         char *p;
182         int count;

184         expr = get_argument_from_call_expr(call->args, arg);
185         if (!expr || expr->type != EXPR_STRING)
186                 return -1;

188         arg++;
189         count = 0;
190         p = expr->string->data;
191         while (*p) {

193                 if (*p == '%') {
194                         count += handle_format(call, &p, &arg, use_max);
187                         count += handle_format(call, &p, &arg);
195                 } else if (*p == '\\') {
196                         p++;
197                 }else {
198                         p++;
199                         count++;
200                 }
201         }

196         count++; /* count the NUL terminator */
203         return count;
204 }

206 int get_formatted_string_size(struct expression *call, int arg)
207 {
208         return get_formatted_string_size_helper(call, arg, true);
209 }

211 int get_formatted_string_min_size(struct expression *call, int arg)
212 {
213         return get_formatted_string_size_helper(call, arg, false);
214 }

216 static void match_not_limited(const char *fn, struct expression *call, void *inf
```



```
217 {
218         struct param_info *params = info;
219         struct range_list *rl;
220         struct expression *dest;
221         struct expression *arg;
222         int buf_size, size;
223         int user = 0;
224         int i;
225         int offset = 0;

227         dest = get_argument_from_call_expr(call->args, params->buf_or_limit);
228         dest = strip_expr(dest);
229         if (dest->type == EXPR_BINOP && dest->op == '+') {
230                 sval_t max;

232                 if (get_hard_max(dest->right, &max))
233                         offset = max.value;
234                 dest = dest->left;
235         }


238         buf_size = get_array_size_bytes(dest);
239         if (buf_size <= 0)
240                 return;

242         size = get_formatted_string_size(call, params->string);
243         if (size < 0)
227         if (size <= 0)
244                 return;
245         if (size < offset)
246                 size -= offset;
247         size++; /* add the NULL terminator */
248         if (size <= buf_size)
249                 return;

251         i = 0;
252         FOR_EACH_PTR(call->args, arg) {
253                 if (i++ <= params->string)
254                         continue;
255                 if (get_user_rl(arg, &rl))
256                         user = 1;
257         } END_FOR_EACH_PTR(arg);

259         sm_error("format string overflow. buf_size: %d length: %d%s",
260                 buf_size, size, user ? " [user data]": "");
261 }
______unchanged_portion_omitted_
```

****
3586 Mon Aug 5 08:38:07 2019
new/usr/src/tools/smatch/src/check_syscall_arg_type.c
11506 smatch resync
****
_____***unchanged_portion_omitted_***

```
 158 void check_syscall_arg_type(int id)
 159 {
 160         my_id = id;
 161         if (option_project != PROJ_KERNEL)
 162                 return;

 164         set_dynamic_states(my_id);
 165         add_merge_hook(my_id, &merge_states);
 166         add_function_hook("fdget", &match_fdget, NULL);
 167 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    3027 Mon Aug  5 08:38:07 2019
new/usr/src/tools/smatch/src/check_testing_index_after_use.c
11506 smatch resync
**********************************************************
______unchanged_portion_omitted_

 38 static int get_the_max(struct expression *expr, sval_t *sval)
 39 {
 40         struct range_list *rl;

 42         if (get_hard_max(expr, sval))
 43                 return 1;
 44         if (!option_spammy)
 45                 return 0;
 46         if (get_fuzzy_max(expr, sval))
 47                 return 1;
 48         if (get_user_rl(expr, &rl)) {
 49                 *sval = rl_max(rl);
 50                 return 1;
 51         }
 52         return 0;
 53 }

 38 static void array_check(struct expression *expr)
 39 {
 40         struct expression *array_expr;
 41         int array_size;
 42         struct expression *offset;
 43         struct range_list *rl;
 60         sval_t max;

 45         expr = strip_expr(expr);
 46         if (!is_array(expr))
 47                 return;

 49         array_expr = get_array_base(expr);
 50         array_size = get_array_size(array_expr);
 51         if (!array_size || array_size == 1)
 52                 return;

 54         offset = get_array_offset(expr);
 55         get_absolute_rl(offset, &rl);
 56         if (rl_max(rl).uvalue < array_size)
 57                 return;
 58         if (buf_comparison_index_ok(expr))
 59                 return;

 72         if (!get_the_max(offset, &max)) {
 61         if (getting_address())
 62                 return;
 63         if (is_capped(offset))
 64                 return;
 65         set_state_expr(my_used_id, offset, alloc_state_num(array_size));
 78         }
 66 }
______unchanged_portion_omitted_

108 void check_testing_index_after_use(int id)
109 {
110         my_used_id = id;
111         set_dynamic_states(my_used_id);
112         add_hook(&array_check, OP_HOOK);
113         add_hook(&match_condition, CONDITION_HOOK);
114         add_modification_hook(my_used_id, &delete);
115 }
```

```
**********************************************************
    9286 Mon Aug  5 08:38:09 2019
new/usr/src/tools/smatch/src/check_uninitialized.c
11506 smatch resync
**********************************************************
______unchanged_portion_omitted_
```

```
 98 static void match_negative_comparison(struct expression *expr)
 99 {
100         struct expression *success;
101         struct sm_state *sm;
102         sval_t max;

104         /*
105          * In the kernel, people don't use "if (ret) {" and "if (ret < 0) {"
106          * consistently.  Ideally Smatch would know the return but often it
107          * doesn't.
108          *
109          */

111         if (option_project != PROJ_KERNEL)
112                 return;

114         if (expr->type != EXPR_COMPARE || expr->op != '<')
115                 return;
116         if (!is_zero(expr->right))
117                 return;
118         if (get_implied_max(expr->left, &max) && max.value == 0)
119                 return;

121         success = compare_expression(expr->left, SPECIAL_EQUAL, expr->right);
122         if (!assume(success))
123                 return;

125         FOR_EACH_MY_SM(my_id, __get_cur_stree(), sm) {
126                 if (sm->state == &initialized)
127                         set_true_false_states(my_id, sm->name, sm->sym, NULL, &i
128         } END_FOR_EACH_SM(sm);

130         end_assume();
131 }

133 static int is_initialized(struct expression *expr)
134 {
135         struct sm_state *sm;

137         expr = strip_expr(expr);
138         if (expr->type != EXPR_SYMBOL)
139                 return 1;
140         sm = get_sm_state_expr(my_id, expr);
141         if (!sm)
142                 return 1;
143         if (!slist_has_state(sm->possible, &uninitialized))
144                 return 1;
145         return 0;
146 }

148 static void match_dereferences(struct expression *expr)
149 {
150         char *name;

152         if (implications_off || parse_error)
117         if (parse_error)
153                 return;

155         if (expr->type != EXPR_PREOP)
```



```
156                 return;
157         if (is_impossible_path())
158                 return;
159         if (is_initialized(expr->unop))
160                 return;

162         name = expr_to_str(expr->unop);
163         sm_error("potentially dereferencing uninitialized '%s'.", name);
164         free_string(name);

166         set_state_expr(my_id, expr->unop, &initialized);
167 }

169 static void match_condition(struct expression *expr)
170 {
171         char *name;

173         if (implications_off || parse_error)
138         if (parse_error)
174                 return;

176         if (is_impossible_path())
177                 return;

179         if (is_initialized(expr))
180                 return;

182         name = expr_to_str(expr);
183         sm_error("potentially using uninitialized '%s'.", name);
184         free_string(name);

186         set_state_expr(my_id, expr, &initialized);
187 }
______unchanged_portion_omitted_

299 static void match_symbol(struct expression *expr)
300 {
301         char *name;

303         if (implications_off || parse_error)
268         if (parse_error)
304                 return;

306         if (is_impossible_path())
307                 return;

309         if (is_initialized(expr))
310                 return;

312         if (is_being_modified(expr))
313                 return;

315         name = expr_to_str(expr);
316         sm_error("uninitialized symbol '%s'.", name);
317         free_string(name);

319         set_state_expr(my_id, expr, &initialized);
320 }
______unchanged_portion_omitted_

383 void check_uninitialized(int id)
384 {
385         my_id = id;

387         add_hook(&match_declarations, DECLARATION_HOOK);
388         add_extra_mod_hook(&extra_mod_hook);
```

```
 389         add_hook(&match_assign, ASSIGNMENT_HOOK);
 390         add_hook(&match_negative_comparison, CONDITION_HOOK);
 391         add_untracked_param_hook(&match_untracked);
 392         add_pre_merge_hook(my_id, &pre_merge_hook);

 394         add_hook(&match_dereferences, DEREF_HOOK);
 395         add_hook(&match_condition, CONDITION_HOOK);
 396         add_hook(&match_call, FUNCTION_CALL_HOOK);
 397         add_hook(&match_call_struct_members, FUNCTION_CALL_HOOK);
 398         add_hook(&match_symbol, SYM_HOOK);

 400         register_ignored_params_from_file();
 401 }
```

*unchanged_portion_omitted_*

```
**********************************************************
    6905 Mon Aug  5 08:38:10 2019
new/usr/src/tools/smatch/src/check_unwind.c
11506 smatch resync
**********************************************************
______unchanged_portion_omitted_

 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_sval("__request_region", valid_ptr_min_sval, valid_
 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_sval("ioremap", valid_ptr_min_sval, valid_ptr_max_s
 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_sval("pci_iomap", valid_ptr_min_sval, valid_ptr_max
 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_sval("__create_workqueue_key", valid_ptr_min_sval,
 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");


 231         add_hook(&match_return, RETURN_HOOK);
 232 }
______unchanged_portion_omitted_
```

```
**********************************************************
    1976 Mon Aug  5 08:38:10 2019
new/usr/src/tools/smatch/src/check_wine_WtoA.c
11506 smatch resync
**********************************************************
```

```
/*
 * 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
 * Idea from Michael Stefaniuc and Vincent BØron'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;
}
_____unchanged_portion_omitted_
```

```
**********************************************************
    4250 Mon Aug  5 08:38:11 2019
new/usr/src/tools/smatch/src/check_zero_to_err_ptr.c
11506 smatch resync
**********************************************************
______unchanged_portion_omitted_
```

```
 77 static int is_non_zero_int(struct range_list *rl)
 78 {
 79         struct data_range *tmp;
 80         int cnt = -1;

 82         FOR_EACH_PTR(rl, tmp) {
 83                 cnt++;

 85                 if (cnt == 0) {
 86                         if (tmp->min.value == INT_MIN &&
 87                             tmp->max.value == -1)
 88                                 continue;
 89                 } else if (cnt == 1) {
 90                         if (tmp->min.value == 1 &&
 91                             tmp->max.value == INT_MAX)
 92                                 return 1;
 93                 }
 94                 return 0;
 95         } END_FOR_EACH_PTR(tmp);
 96         return 0;
 97 }

 99 static int is_valid_ptr(sval_t sval)
100 {
101         if (sval.value == INT_MIN || sval.value == INT_MAX)
 79         if (sval.type == &int_ctype &&
 80             (sval.value == INT_MIN || sval.value == INT_MAX))
102                 return 0;

104         if (sval_cmp(valid_ptr_min_sval, sval) <= 0 &&
105             sval_cmp(valid_ptr_max_sval, sval) >= 0) {
 84             sval_cmp(valid_ptr_max_sval, sval) >= 0)
106                 return 1;
107         }
108         return 0;
109 }

111 static int has_distinct_zero(struct range_list *rl)
112 {
113         struct data_range *tmp;

115         FOR_EACH_PTR(rl, tmp) {
116                 if (tmp->min.value == 0 || tmp->max.value == 0)
117                         return 1;
118         } END_FOR_EACH_PTR(tmp);
119         return 0;
120 }

122 static void match_err_ptr(const char *fn, struct expression *expr, void *data)
123 {
124         struct expression *arg_expr;
125         struct sm_state *sm, *tmp;
 93         sval_t sval;

127         if (is_impossible_path())
128                 return;

130         arg_expr = get_argument_from_call_expr(expr->args, 0);
131         sm = get_sm_state_expr(SMATCH_EXTRA, arg_expr);
```



```
132         if (!sm)
133                 return;

135         if (is_comparison_call(expr))
136                 return;

138         if (next_line_checks_IS_ERR(expr, arg_expr))
139                 return;
140         if (strcmp(fn, "ERR_PTR") == 0 &&
141             next_line_is_if(arg_expr))
142                 return;

144         FOR_EACH_PTR(sm->possible, tmp) {
145                 if (!estate_rl(tmp->state))
146                         continue;
147                 if (is_non_zero_int(estate_rl(tmp->state)))
148                         continue;
149                 if (has_distinct_zero(estate_rl(tmp->state))) {
150                         sm_warning("passing zero to '%s'", fn);
151                         return;
152                 }
153                 if (strcmp(fn, "PTR_ERR") != 0)
154                         continue;
155                 if (is_valid_ptr(estate_min(tmp->state)) &&
156                     is_valid_ptr(estate_max(tmp->state))) {
157                         sm_warning("passing a valid pointer to '%s'", fn);
158                         return;
159                 }
117                 if (!rl_to_sval(estate_rl(tmp->state), &sval))
118                         continue;
119                 if (sval.value != 0)
120                         continue;
121                 sm_warning("passing zero to '%s'", fn);
122                 return;
160         } END_FOR_EACH_PTR(tmp);
161 }
```

```
______unchanged_portion_omitted_
```

```
**********************************************************
   92239 Mon Aug  5 08:38:11 2019
new/usr/src/tools/smatch/src/evaluate.c
11506 smatch resync
**********************************************************
______unchanged_portion_omitted_

1949 struct symbol *find_identifier(struct ident *ident, struct symbol_list *_list, i
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 }
______unchanged_portion_omitted_
```

```
**********************************************************
    8448 Mon Aug  5 08:38:11 2019
new/usr/src/tools/smatch/src/expression.h
11506 smatch resync
**********************************************************
______unchanged_portion_omitted_

 247 /* Constant expression values */
 248 int is_zero_constant(struct expression *);
 249 int expr_truth_value(struct expression *expr);
 250 long long get_expression_value(struct expression *);
 251 long long const_expression_value(struct expression *);
 252 long long get_expression_value_silent(struct expression *expr);

 254 /* Expression parsing */
 255 struct token *parse_expression(struct token *token, struct expression **tree);
 256 struct token *conditional_expression(struct token *token, struct expression **tr
 257 struct token *primary_expression(struct token *token, struct expression **tree);
 258 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 *);
 264 struct symbol *find_identifier(struct ident *ident, struct symbol_list *_list, i

 266 extern int expand_symbol(struct symbol *);

 268 static inline struct expression *alloc_expression(struct position pos, int type)
 269 {
 270         struct expression *expr = __alloc_expression(0);
 271         expr->type = type;
 272         expr->pos = pos;
 273         expr->flags = CEF_NONE;
 274         return expr;
 275 }
______unchanged_portion_omitted_
```

```
**********************************************************
    5766 Mon Aug  5 08:38:11 2019
new/usr/src/tools/smatch/src/graph.c
11506 smatch resync
**********************************************************
```

```
  1 /* Copyright ´' International Business Machines Corp., 2006
  1 /* Copyright ' International Business Machines Corp., 2006
  2  *              Adelard LLP, 2007
  3  *
  4  * Author: Josh Triplett <josh@freedesktop.org>
  5  *         Dan Sheridan <djs@adelard.com>
  6  *
  7  * Permission is hereby granted, free of charge, to any person obtaining a copy
  8  * of this software and associated documentation files (the "Software"), to deal
  9  * in the Software without restriction, including without limitation the rights
 10  * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 11  * copies of the Software, and to permit persons to whom the Software is
 12  * furnished to do so, subject to the following conditions:
 13  *
 14  * The above copyright notice and this permission notice shall be included in
 15  * all copies or substantial portions of the Software.
 16  *
 17  * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 18  * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 19  * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 20  * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 21  * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 22  * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
 23  * THE SOFTWARE.
 24  */
 25 #include <stdarg.h>
 26 #include <stdlib.h>
 27 #include <stdio.h>
 28 #include <string.h>
 29 #include <ctype.h>
 30 #include <unistd.h>
 31 #include <fcntl.h>

 33 #include "lib.h"
 34 #include "allocate.h"
 35 #include "token.h"
 36 #include "parse.h"
 37 #include "symbol.h"
 38 #include "expression.h"
 39 #include "linearize.h"


 42 /* Draw the subgraph for a given entrypoint. Includes details of loads
 43  * and stores for globals, and marks return bbs */
 44 static void graph_ep(struct entrypoint *ep)
 45 {
 46         struct basic_block *bb;
 47         struct instruction *insn;

 49         const char *fname, *sname;

 51         fname = show_ident(ep->name->ident);
 52         sname = stream_name(ep->entry->bb->pos.stream);

 54         printf("subgraph cluster%p {\n"
 55                "    color=blue;\n"
 56                "    label=<<TABLE BORDER=\"0\" CELLBORDER=\"0\">\n"
 57                "              <TR><TD>%s</TD></TR>\n"
 58                "              <TR><TD><FONT POINT-SIZE=\"21\">%s()</FONT></TD></T
 59                "           </TABLE>>;\n"
 60                "    file=\"%s\";\n"
```



```
 61                "    fun=\"%s\";\n"
 62                "    ep=bb%p;\n",
 63                ep, sname, fname, sname, fname, ep->entry->bb);

 65         FOR_EACH_PTR(ep->bbs, bb) {
 66                 struct basic_block *child;
 67                 int ret = 0;
 68                 const char * s = ", ls=\"[";

 70                 /* Node for the bb */
 71                 printf("    bb%p [shape=ellipse,label=%d,line=%d,col=%d",
 72                        bb, bb->pos.line, bb->pos.line, bb->pos.pos);


 75                 /* List loads and stores */
 76                 FOR_EACH_PTR(bb->insns, insn) {
 77                         switch(insn->opcode) {
 78                         case OP_STORE:
 79                                 if (insn->symbol->type == PSEUDO_SYM) {
 80                                   printf("%s store(%s)", s, show_ident(insn->sym
 81                                   s = ",";
 82                                 }
 83                                 break;

 85                         case OP_LOAD:
 86                                 if (insn->symbol->type == PSEUDO_SYM) {
 87                                   printf("%s load(%s)", s, show_ident(insn->symb
 88                                   s = ",";
 89                                 }
 90                                 break;

 92                         case OP_RET:
 93                                 ret = 1;
 94                                 break;

 96                         }
 97                 } END_FOR_EACH_PTR(insn);
 98                 if (s[1] == 0)
 99                         printf("]\"");
100                 if (ret)
101                         printf(",op=ret");
102                 printf("];\n");

104                 /* Edges between bbs; lower weight for upward edges */
105                 FOR_EACH_PTR(bb->children, child) {
106                         printf("    bb%p -> bb%p [op=br, %s];\n", bb, child,
107                                (bb->pos.line > child->pos.line) ? "weight=5" : "
108                 } END_FOR_EACH_PTR(child);
109         } END_FOR_EACH_PTR(bb);

111         printf("}\n");
112 }
```

*_____unchanged_portion_omitted_*

```
**********************************************************
    8798 Mon Aug  5 08:38:12 2019
new/usr/src/tools/smatch/src/smatch.c
11506 smatch resync
**********************************************************
```

```
/*
 * 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
 *
 * Copyright 2019 Joyent, Inc.
 */

#include <stdio.h>
#include <unistd.h>
#include <libgen.h>
#include "smatch.h"
#include "smatch_slist.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;
int option_timeout = 60;

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[] = {
______unchanged_portion_omitted_

#define OPTION(_x) do {                                 \
        if (match_option((*argvp)[1], #_x)) {           \
        if (match_option((*argvp)[i], #_x)) {           \
                option_##_x = 1;                        \
        }                                               \
} while (0)

void parse_args(int *argcp, char ***argvp)
{
        int i;

        for (i = 1 ; i < *argcp; i++) {
                if (!strcmp((*argvp)[i], "--help"))
                        help();

                if (!strcmp((*argvp)[i], "--show-checks"))
                        show_checks();

                if (!strncmp((*argvp)[i], "--project=", 10))
                        option_project_str = (*argvp)[i] + 10;

                if (!strncmp((*argvp)[i], "-p=", 3))
                        option_project_str = (*argvp)[i] + 3;

                if (!strncmp((*argvp)[i], "--db-file=", 10))
                        option_db_file = (*argvp)[i] + 10;

                if (!strncmp((*argvp)[i], "--data=", 7))
                        option_datadir_str = (*argvp)[i] + 7;

                if (!strncmp((*argvp)[i], "--debug=", 8))
                        option_debug_check = (*argvp)[i] + 8;

                if (strncmp((*argvp)[i], "--trace=", 8) == 0)
                        trace_variable = (*argvp)[i] + 8;

                if (strncmp((*argvp)[i], "--enable=", 9) == 0) {
                        enable_disable_checks((*argvp)[i] + 9, 1);
                        option_enable = 1;
                }

                if (strncmp((*argvp)[i], "--disable=", 10) == 0) {
                        enable_disable_checks((*argvp)[i] + 10, 0);
                        option_enable = 1;
                        option_disable = 1;
                }

                if (!strncmp((*argvp)[i], "--timeout=", 10)) {
                        if (sscanf((*argvp)[i] + 10, "%d",
                            &option_timeout) != 1)
                                sm_fatal("invalid option %s", (*argvp)[i]);
                }

                OPTION(fatal_checks);
                OPTION(spammy);
                OPTION(info);
                OPTION(debug);
```

```
234                 OPTION(debug_implied);
235                 OPTION(debug_related);
234                 OPTION(assume_loops);
235                 OPTION(no_data);
236                 OPTION(two_passes);
237                 OPTION(full_path);
238                 OPTION(param_mapper);
239                 OPTION(call_tree);
240                 OPTION(file_output);
241                 OPTION(time);
242                 OPTION(mem);
243                 OPTION(no_db);
244                 OPTION(succeed);
245         }

247         if (strcmp(option_project_str, "smatch_generic") != 0)
248                 option_project = PROJ_UNKNOWN;

250         if (strcmp(option_project_str, "kernel") == 0)
251                 option_project = PROJ_KERNEL;
252         else if (strcmp(option_project_str, "wine") == 0)
253                 option_project = PROJ_WINE;
254         else if (strcmp(option_project_str, "illumos_kernel") == 0)
255                 option_project = PROJ_ILLUMOS_KERNEL;
256         else if (strcmp(option_project_str, "illumos_user") == 0)
257                 option_project = PROJ_ILLUMOS_USER;
258 }
```

_____unchanged_portion_omitted_

```
321 int main(int argc, char **argv)
322 {
323         struct string_list *filelist = NULL;
324         int i;
325         reg_func func;

327         sm_outfd = stdout;
328         sql_outfd = stdout;
329         caller_info_fd = stdout;

331         progname = argv[0];

333         parse_args(&argc, &argv);

335         if (argc < 2)
336                 help();

338         /* this gets set back to zero when we parse the first function */
339         final_pass = 1;

341         bin_dir = get_bin_dir(argv[0]);
342         data_dir = get_data_dir(argv[0]);

344         allocate_hook_memory();
345         allocate_dynamic_states_array(num_checks);
346         create_function_hook_hash();
347         open_smatch_db(option_db_file);
348         sparse_initialize(argc, argv, &filelist);
349         alloc_valid_ptr_rl();

351         for (i = 1; i < ARRAY_SIZE(reg_funcs); i++) {
352                 func = reg_funcs[i].func;
353                 /* The script IDs start at 1.
354                    0 is used for internal stuff. */
355                 if (!option_enable || reg_funcs[i].enabled == 1 ||
356                     (option_disable && reg_funcs[i].enabled != -1) ||
357                     strncmp(reg_funcs[i].name, "register_", 9) == 0)
```



```
358                         func(i);
359         }

361         smatch(filelist);
358         smatch(argc, argv);
362         free_string(data_dir);

364         if (option_succeed)
365                 return 0;
366         if (sm_nr_errors > 0)
367                 return 1;
368         if (sm_nr_checks > 0 && option_fatal_checks)
369                 return 1;
370         return 0;
371 }
```

_____unchanged_portion_omitted_

```
**********************************************************
   47415 Mon Aug  5 08:38:12 2019
new/usr/src/tools/smatch/src/smatch.h
11506 smatch resync
**********************************************************
______unchanged_portion_omitted_
  72 DECLARE_ALLOCATOR(tracker);
  73 DECLARE_PTR_LIST(tracker_list, struct tracker);
  74 DECLARE_PTR_LIST(stree_stack, struct stree);

  76 /* The first 3 struct members must match struct tracker */
  77 struct sm_state {
  78         const char *name;
  79         struct symbol *sym;
  80         unsigned short owner;
  81         unsigned short merged:1;
  82         unsigned short skip_implications:1;
  83         unsigned int nr_children;
  82         unsigned int line;
  83         struct smatch_state *state;
  84         struct stree *pool;
  85         struct sm_state *left;
  86         struct sm_state *right;
  87         struct state_list *possible;
  88 };
______unchanged_portion_omitted_
 101 DECLARE_PTR_LIST(constraint_list, struct constraint);

 103 struct bit_info {
 104         unsigned long long set;
 105         unsigned long long possible;
 106 };

 108 enum hook_type {
 109         EXPR_HOOK,
 110         STMT_HOOK,
 111         STMT_HOOK_AFTER,
 112         SYM_HOOK,
 113         STRING_HOOK,
 114         DECLARATION_HOOK,
 115         ASSIGNMENT_HOOK,
 116         ASSIGNMENT_HOOK_AFTER,
 117         RAW_ASSIGNMENT_HOOK,
 118         GLOBAL_ASSIGNMENT_HOOK,
 119         LOGIC_HOOK,
 120         CONDITION_HOOK,
 121         PRELOOP_HOOK,
 122         SELECT_HOOK,
 123         WHOLE_CONDITION_HOOK,
 124         FUNCTION_CALL_HOOK_BEFORE,
 125         FUNCTION_CALL_HOOK,
 126         CALL_HOOK_AFTER_INLINE,
 127         FUNCTION_CALL_HOOK_AFTER_DB,
 128         CALL_ASSIGNMENT_HOOK,
 129         MACRO_ASSIGNMENT_HOOK,
 130         BINOP_HOOK,
 131         OP_HOOK,
 132         DEREF_HOOK,
 133         CASE_HOOK,
 134         ASM_HOOK,
 135         CAST_HOOK,
 136         SIZEOF_HOOK,
 137         BASE_HOOK,
 138         FUNC_DEF_HOOK,
 139         AFTER_DEF_HOOK,
 140         END_FUNC_HOOK,
```



```
 141         AFTER_FUNC_HOOK,
 142         RETURN_HOOK,
 143         INLINE_FN_START,
 144         INLINE_FN_END,
 145         END_FILE_HOOK,
 146         NUM_HOOKS,
 147 };

 149 #define TRUE 1
 150 #define FALSE 0

 152 struct range_list;

 154 void add_hook(void *func, enum hook_type type);
 155 typedef struct smatch_state *(merge_func_t)(struct smatch_state *s1, struct smat
 156 typedef struct smatch_state *(unmatched_func_t)(struct sm_state *state);
 157 void add_merge_hook(int client_id, merge_func_t *func);
 158 void add_unmatched_state_hook(int client_id, unmatched_func_t *func);
 159 void add_pre_merge_hook(int client_id, void (*hook)(struct sm_state *sm));
 160 typedef void (scope_hook)(void *data);
 161 void add_scope_hook(scope_hook *hook, void *data);
 162 typedef void (func_hook)(const char *fn, struct expression *expr, void *data);
 163 typedef void (implication_hook)(const char *fn, struct expression *call_expr,
 164                                 struct expression *assign_expr, void *data);
 165 typedef void (return_implies_hook)(struct expression *call_expr,
 166                                    int param, char *key, char *value);
 167 typedef int (implied_return_hook)(struct expression *call_expr, void *info, stru
 168 void add_function_hook(const char *look_for, func_hook *call_back, void *data);

 170 void add_function_assign_hook(const char *look_for, func_hook *call_back,
 171                               void *info);
 172 void add_implied_return_hook(const char *look_for,
 173                              implied_return_hook *call_back,
 174                              void *info);
 175 void add_macro_assign_hook(const char *look_for, func_hook *call_back,
 176                               void *info);
 177 void add_macro_assign_hook_extra(const char *look_for, func_hook *call_back,
 178                               void *info);
 179 void return_implies_state(const char *look_for, long long start, long long end,
 180                           implication_hook *call_back, void *info);
 181 void return_implies_state_sval(const char *look_for, sval_t start, sval_t end,
 182                           implication_hook *call_back, void *info);
 183 void select_return_states_hook(int type, return_implies_hook *callback);
 184 void select_return_states_before(void (*fn)(void));
 185 void select_return_states_after(void (*fn)(void));
 186 int get_implied_return(struct expression *expr, struct range_list **rl);
 187 void allocate_hook_memory(void);

 189 struct modification_data {
 190         struct smatch_state *prev;
 191         struct expression *cur;
 192 };

 194 typedef void (modification_hook)(struct sm_state *sm, struct expression *mod_exp
 195 void add_modification_hook(int owner, modification_hook *call_back);
 196 void add_modification_hook_late(int owner, modification_hook *call_back);
 197 struct smatch_state *get_modification_state(struct expression *expr);

 199 int outside_of_function(void);
 200 const char *get_filename(void);
 201 const char *get_base_file(void);
 202 char *get_function(void);
 203 int get_lineno(void);
 204 extern int final_pass;
 205 extern struct symbol *cur_func_sym;
 206 extern int option_debug;
```

```
207 extern int local_debug;
208 extern int option_info;
209 extern int option_spammy;
210 extern int option_timeout;
211 extern char *trace_variable;
212 extern struct stree *global_states;
213 int is_skipped_function(void);
214 int is_silenced_function(void);
215 extern bool implications_off;

217 /* smatch_impossible.c */
218 int is_impossible_path(void);
219 void set_path_impossible(void);

221 extern FILE *sm_outfd;
222 extern FILE *sql_outfd;
223 extern FILE *caller_info_fd;
224 extern int sm_nr_checks;
225 extern int sm_nr_errors;
226 extern const char *progname;

228 /*
229  * How to use these routines:
230  *
231  * sm_fatal(): an internal error of some kind that should immediately exit
232  * sm_ierror(): an internal error
233  * sm_perror(): an internal error from parsing input source
234  * sm_error(): an error from input source
235  * sm_warning(): a warning from input source
236  * sm_info(): info message (from option_info)
237  * sm_debug(): debug message
238  * sm_msg(): other message (please avoid using this)
239  */

241 #define sm_printf(msg...) do { if (final_pass || option_debug || local_debug) fp

243 static inline void sm_prefix(void)
244 {
245         sm_printf("%s: %s:%d %s() ", progname, get_filename(), get_lineno(), get
246 }
_____unchanged_portion_omitted_
339 #define ALIGN(x, a) (((x) + (a) - 1) & ~((a) - 1))

341 struct smatch_state *__get_state(int owner, const char *name, struct symbol *sym
342 struct smatch_state *get_state(int owner, const char *name, struct symbol *sym);
343 struct smatch_state *get_state_expr(int owner, struct expression *expr);
344 struct state_list *get_possible_states(int owner, const char *name,
345                                        struct symbol *sym);
346 struct state_list *get_possible_states_expr(int owner, struct expression *expr);
347 struct sm_state *set_state(int owner, const char *name, struct symbol *sym,
348                 struct smatch_state *state);
349 struct sm_state *set_state_expr(int owner, struct expression *expr,
350                  struct smatch_state *state);
351 void delete_state(int owner, const char *name, struct symbol *sym);
352 void delete_state_expr(int owner, struct expression *expr);
353 void __delete_all_states_sym(struct symbol *sym);
354 void set_true_false_states(int owner, const char *name, struct symbol *sym,
355                            struct smatch_state *true_state,
356                            struct smatch_state *false_state);
357 void set_true_false_states_expr(int owner, struct expression *expr,
358                            struct smatch_state *true_state,
359                            struct smatch_state *false_state);

361 struct stree *get_all_states_from_stree(int owner, struct stree *source);
362 struct stree *get_all_states_stree(int id);
363 struct stree *__get_cur_stree(void);
```



```
364 int is_reachable(void);
365 void add_get_state_hook(void (*fn)(int owner, const char *name, struct symbol *s

367 /* smatch_helper.c */
368 DECLARE_PTR_LIST(int_stack, int);
369 char *alloc_string(const char *str);
370 void free_string(char *str);
371 void append(char *dest, const char *data, int buff_len);
372 void remove_parens(char *str);
373 struct smatch_state *alloc_state_num(int num);
374 struct smatch_state *alloc_state_str(const char *name);
375 struct smatch_state *merge_str_state(struct smatch_state *s1, struct smatch_stat
376 struct smatch_state *alloc_state_expr(struct expression *expr);
377 struct expression *get_argument_from_call_expr(struct expression_list *args,
378                                                int num);

380 char *expr_to_var(struct expression *expr);
381 struct symbol *expr_to_sym(struct expression *expr);
382 char *expr_to_str(struct expression *expr);
383 char *expr_to_str_sym(struct expression *expr,
384                                         struct symbol **sym_ptr);
385 char *expr_to_var_sym(struct expression *expr,
386                                  struct symbol **sym_ptr);
387 char *expr_to_known_chunk_sym(struct expression *expr, struct symbol **sym);
388 char *expr_to_chunk_sym_vsl(struct expression *expr, struct symbol **sym, struct
389 int get_complication_score(struct expression *expr);

391 int sym_name_is(const char *name, struct expression *expr);
392 int get_const_value(struct expression *expr, sval_t *sval);
393 int get_value(struct expression *expr, sval_t *val);
394 int get_implied_value(struct expression *expr, sval_t *val);
395 int get_implied_min(struct expression *expr, sval_t *sval);
396 int get_implied_max(struct expression *expr, sval_t *val);
397 int get_hard_max(struct expression *expr, sval_t *sval);
398 int get_fuzzy_min(struct expression *expr, sval_t *min);
399 int get_fuzzy_max(struct expression *expr, sval_t *max);
400 int get_absolute_min(struct expression *expr, sval_t *sval);
401 int get_absolute_max(struct expression *expr, sval_t *sval);
402 int parse_call_math(struct expression *expr, char *math, sval_t *val);
403 int parse_call_math_rl(struct expression *call, const char *math, struct range_l
396 int parse_call_math_rl(struct expression *call, char *math, struct range_list **
404 char *get_value_in_terms_of_parameter_math(struct expression *expr);
405 char *get_value_in_terms_of_parameter_math_var_sym(const char *var, struct symbo
406 int is_zero(struct expression *expr);
407 int known_condition_true(struct expression *expr);
408 int known_condition_false(struct expression *expr);
409 int implied_condition_true(struct expression *expr);
410 int implied_condition_false(struct expression *expr);
411 int can_integer_overflow(struct symbol *type, struct expression *expr);
412 void clear_math_cache(void);

414 int is_array(struct expression *expr);
415 struct expression *get_array_base(struct expression *expr);
416 struct expression *get_array_offset(struct expression *expr);
417 const char *show_state(struct smatch_state *state);
418 struct statement *get_expression_statement(struct expression *expr);
419 struct expression *strip_parens(struct expression *expr);
420 struct expression *strip_expr(struct expression *expr);
421 struct expression *strip_expr_set_parent(struct expression *expr);
422 void scoped_state(int my_id, const char *name, struct symbol *sym);
423 int is_error_return(struct expression *expr);
424 int getting_address(void);
425 int get_struct_and_member(struct expression *expr, const char **type, const char
426 char *get_member_name(struct expression *expr);
427 char *get_fnptr_name(struct expression *expr);
428 int cmp_pos(struct position pos1, struct position pos2);
```

```
429 int positions_eq(struct position pos1, struct position pos2);
430 struct statement *get_current_statement(void);
431 struct statement *get_prev_statement(void);
432 struct expression *get_last_expr_from_expression_stmt(struct expression *expr);
433 int get_param_num_from_sym(struct symbol *sym);
434 int get_param_num(struct expression *expr);
435 int ms_since(struct timeval *start);
436 int parent_is_gone_var_sym(const char *name, struct symbol *sym);
437 int parent_is_gone(struct expression *expr);
438 int invert_op(int op);
439 int op_remove_assign(int op);
440 int expr_equiv(struct expression *one, struct expression *two);
441 void push_int(struct int_stack **stack, int num);
442 int pop_int(struct int_stack **stack);

444 /* smatch_type.c */
445 struct symbol *get_real_base_type(struct symbol *sym);
446 int type_bytes(struct symbol *type);
447 int array_bytes(struct symbol *type);
448 struct symbol *get_pointer_type(struct expression *expr);
449 struct symbol *get_type(struct expression *expr);
450 struct symbol *get_final_type(struct expression *expr);
451 struct symbol *get_promoted_type(struct symbol *left, struct symbol *right);
452 int type_signed(struct symbol *base_type);
453 int expr_unsigned(struct expression *expr);
454 int expr_signed(struct expression *expr);
455 int returns_unsigned(struct symbol *base_type);
456 int is_pointer(struct expression *expr);
457 int returns_pointer(struct symbol *base_type);
458 sval_t sval_type_max(struct symbol *base_type);
459 sval_t sval_type_min(struct symbol *base_type);
460 int nr_bits(struct expression *expr);
461 int is_void_pointer(struct expression *expr);
462 int is_char_pointer(struct expression *expr);
463 int is_string(struct expression *expr);
464 int is_static(struct expression *expr);
465 int is_local_variable(struct expression *expr);
466 int types_equiv(struct symbol *one, struct symbol *two);
467 int fn_static(void);
468 const char *global_static();
469 struct symbol *cur_func_return_type(void);
470 struct symbol *get_arg_type(struct expression *fn, int arg);
471 struct symbol *get_member_type_from_key(struct expression *expr, const char *key
472 struct symbol *get_arg_type_from_key(struct expression *fn, int param, struct ex
473 int is_struct(struct expression *expr);
474 char *type_to_str(struct symbol *type);

476 /* smatch_ignore.c */
477 void add_ignore(int owner, const char *name, struct symbol *sym);
478 int is_ignored(int owner, const char *name, struct symbol *sym);
479 void add_ignore_expr(int owner, struct expression *expr);
480 int is_ignored_expr(int owner, struct expression *expr);

482 /* smatch_var_sym */
483 struct var_sym *alloc_var_sym(const char *var, struct symbol *sym);
484 struct var_sym_list *expr_to_vsl(struct expression *expr);
485 void add_var_sym(struct var_sym_list **list, const char *var, struct symbol *sym
486 void add_var_sym_expr(struct var_sym_list **list, struct expression *expr);
487 void del_var_sym(struct var_sym_list **list, const char *var, struct symbol *sym
488 int in_var_sym_list(struct var_sym_list *list, const char *var, struct symbol *s
489 struct var_sym_list *clone_var_sym_list(struct var_sym_list *from_vsl);
490 void merge_var_sym_list(struct var_sym_list **dest, struct var_sym_list *src);
491 struct var_sym_list *combine_var_sym_lists(struct var_sym_list *one, struct var_
492 int var_sym_lists_equiv(struct var_sym_list *one, struct var_sym_list *two);
493 void free_var_sym_list(struct var_sym_list **list);
494 void free_var_syms_and_list(struct var_sym_list **list);
```



```
496 /* smatch_tracker */
497 struct tracker *alloc_tracker(int owner, const char *name, struct symbol *sym);
498 void add_tracker(struct tracker_list **list, int owner, const char *name,
499                  struct symbol *sym);
500 void add_tracker_expr(struct tracker_list **list, int owner, struct expression *
501 void del_tracker(struct tracker_list **list, int owner, const char *name,
502                  struct symbol *sym);
503 int in_tracker_list(struct tracker_list *list, int owner, const char *name,
504                  struct symbol *sym);
505 void free_tracker_list(struct tracker_list **list);
506 void free_trackers_and_list(struct tracker_list **list);

508 /* smatch_conditions */
509 int in_condition(void);

511 /* smatch_flow.c */

513 extern int __in_fake_assign;
514 extern int __in_fake_parameter_assign;
515 extern int __in_fake_struct_assign;
516 extern int in_fake_env;
517 void smatch (struct string_list *filelist);
509 void smatch (int argc, char **argv);
518 int inside_loop(void);
519 int definitely_inside_loop(void);
520 struct expression *get_switch_expr(void);
521 int in_expression_statement(void);
522 void __process_post_op_stack(void);
523 void __split_expr(struct expression *expr);
524 void __split_label_stmt(struct statement *stmt);
525 void __split_stmt(struct statement *stmt);
526 extern int __in_function_def;
527 extern int option_assume_loops;
528 extern int option_two_passes;
529 extern int option_no_db;
530 extern int option_file_output;
531 extern int option_time;
532 extern struct expression_list *big_expression_stack;
533 extern struct expression_list *big_condition_stack;
534 extern struct statement_list *big_statement_stack;
535 int is_assigned_call(struct expression *expr);
536 int inlinable(struct expression *expr);
537 extern int __inline_call;
538 extern struct expression *__inline_fn;
539 extern int __in_pre_condition;
540 extern int __bail_on_rest_of_function;
541 extern struct statement *__prev_stmt;
542 extern struct statement *__cur_stmt;
543 extern struct statement *__next_stmt;
544 void init_fake_env(void);
545 void end_fake_env(void);
546 int time_parsing_function(void);
547 bool taking_too_long(void);

549 /* smatch_struct_assignment.c */
550 struct expression *get_faked_expression(void);
551 void __fake_struct_member_assignments(struct expression *expr);

553 /* smatch_project.c */
554 int is_no_inline_function(const char *function);

556 /* smatch_conditions */
557 void __split_whole_condition(struct expression *expr);
558 void __handle_logic(struct expression *expr);
559 int is_condition(struct expression *expr);
```

```
int __handle_condition_assigns(struct expression *expr);
int __handle_select_assigns(struct expression *expr);
int __handle_expr_statement_assigns(struct expression *expr);

/* smatch_implied.c */
extern int option_debug_implied;
extern int option_debug_related;
struct range_list_stack;
void param_limit_implications(struct expression *expr, int param, char *key, cha
struct stree *__implied_case_stree(struct expression *switch_expr,
                                   struct range_list *case_rl,
                                   struct range_list_stack **remaining_cases,
                                   struct stree **raw_stree);
void overwrite_states_using_pool(struct sm_state *gate_sm, struct sm_state *pool
int assume(struct expression *expr);
void end_assume(void);
int impossible_assumption(struct expression *left, int op, sval_t sval);

/* smatch_slist.h */
bool has_dynamic_states(unsigned short owner);
void set_dynamic_states(unsigned short owner);

/* smatch_extras.c */
int in_warn_on_macro(void);
#define SMATCH_EXTRA 5 /* this is my_id from smatch extra set in smatch.c */
extern int RETURN_ID;

struct data_range {
        sval_t min;
        sval_t max;
};

#define MTAG_ALIAS_BIT (1ULL << 63)
#define MTAG_OFFSET_MASK 0xfffULL
#define MTAG_SEED 0xdead << 12

const extern unsigned long valid_ptr_min;
extern unsigned long valid_ptr_max;
extern const sval_t valid_ptr_min_sval;
extern sval_t valid_ptr_max_sval;
extern long long valid_ptr_min, valid_ptr_max;
extern sval_t valid_ptr_min_sval, valid_ptr_max_sval;
extern struct range_list *valid_ptr_rl;
void alloc_valid_ptr_rl(void);

static const sval_t array_min_sval = {
        .type = &ptr_ctype,
        {.value = 100000},
};
static const sval_t array_max_sval = {
        .type = &ptr_ctype,
        {.value = ULONG_MAX - 4095},
        {.value = 199999},
};
static const sval_t text_seg_min = {
        .type = &ptr_ctype,
        {.value = 4096},
        {.value = 100000000},
};
static const sval_t text_seg_max = {
        .type = &ptr_ctype,
        {.value = ULONG_MAX - 4095},
        {.value = 177777777},
};
static const sval_t data_seg_min = {
        .type = &ptr_ctype,
```



```
        {.value = 4096},
        {.value = 200000000},
};
static const sval_t data_seg_max = {
        .type = &ptr_ctype,
        {.value = ULONG_MAX - 4095},
        {.value = 277777777},
};
static const sval_t bss_seg_min = {
        .type = &ptr_ctype,
        {.value = 4096},
        {.value = 300000000},
};
static const sval_t bss_seg_max = {
        .type = &ptr_ctype,
        {.value = ULONG_MAX - 4095},
        {.value = 377777777},
};
static const sval_t stack_seg_min = {
        .type = &ptr_ctype,
        {.value = 4096},
        {.value = 400000000},
};
static const sval_t stack_seg_max = {
        .type = &ptr_ctype,
        {.value = ULONG_MAX - 4095},
        {.value = 477777777},
};
static const sval_t kmalloc_seg_min = {
        .type = &ptr_ctype,
        {.value = 4096},
        {.value = 500000000},
};
static const sval_t kmalloc_seg_max = {
        .type = &ptr_ctype,
        {.value = ULONG_MAX - 4095},
        {.value = 577777777},
};
static const sval_t vmalloc_seg_min = {
        .type = &ptr_ctype,
        {.value = 4096},
        {.value = 600000000},
};
static const sval_t vmalloc_seg_max = {
        .type = &ptr_ctype,
        {.value = ULONG_MAX - 4095},
        {.value = 677777777},
};
static const sval_t fn_ptr_min = {
        .type = &ptr_ctype,
        {.value = 4096},
        {.value = 700000000},
};
static const sval_t fn_ptr_max = {
        .type = &ptr_ctype,
        {.value = ULONG_MAX - 4095},
        {.value = 777777777},
};

char *get_other_name_sym(const char *name, struct symbol *sym, struct symbol **n
char *map_call_to_other_name_sym(const char *name, struct symbol *sym, struct sy
char *map_long_to_short_name_sym(const char *name, struct symbol *sym, struct sy
char *map_long_to_short_name_sym(const char *name, struct symbol *sym, struct sy
char *map_long_to_short_name_sym_nostack(const char *name, struct symbol *sym, s

#define STRLEN_MAX_RET 1010101
```

```
672 /* smatch_absolute.c */
673 int get_absolute_min_helper(struct expression *expr, sval_t *sval);
674 int get_absolute_max_helper(struct expression *expr, sval_t *sval);

676 /* smatch_local_values.c */
677 int get_local_rl(struct expression *expr, struct range_list **rl);
678 int get_local_max_helper(struct expression *expr, sval_t *sval);
679 int get_local_min_helper(struct expression *expr, sval_t *sval);

681 /* smatch_type_value.c */
682 int get_db_type_rl(struct expression *expr, struct range_list **rl);
683 /* smatch_data_val.c */
684 int get_mtag_rl(struct expression *expr, struct range_list **rl);
685 /* smatch_array_values.c */
686 int get_array_rl(struct expression *expr, struct range_list **rl);

688 /* smatch_states.c */
689 void __swap_cur_stree(struct stree *stree);
690 void __push_fake_cur_stree();
691 struct stree *__pop_fake_cur_stree();
692 void __free_fake_cur_stree();
693 void __set_fake_cur_stree_fast(struct stree *stree);
694 void __pop_fake_cur_stree_fast(void);
695 void __merge_stree_into_cur(struct stree *stree);

697 int unreachable(void);
698 void __set_sm(struct sm_state *sm);
699 void __set_sm_cur_stree(struct sm_state *sm);
700 void __set_sm_fake_stree(struct sm_state *sm);
701 void __set_true_false_sm(struct sm_state *true_state,
702                          struct sm_state *false_state);
703 void nullify_path(void);
704 void __match_nullify_path_hook(const char *fn, struct expression *expr,
705                                void *unused);
706 void __unnullify_path(void);
707 int __path_is_null(void);
708 void save_all_states(void);
709 void restore_all_states(void);
710 void free_goto_stack(void);
711 void clear_all_states(void);

713 struct sm_state *get_sm_state(int owner, const char *name,
714                               struct symbol *sym);
715 struct sm_state *get_sm_state_expr(int owner, struct expression *expr);
716 void __push_true_states(void);
717 void __use_false_states(void);
718 void __discard_false_states(void);
719 void __merge_false_states(void);
720 void __merge_true_states(void);

722 void __negate_cond_stacks(void);
723 void __use_pre_cond_states(void);
724 void __use_cond_true_states(void);
725 void __use_cond_false_states(void);
726 void __push_cond_stacks(void);
727 void __fold_in_set_states(void);
728 void __free_set_states(void);
729 struct stree *__copy_cond_true_states(void);
730 struct stree *__copy_cond_false_states(void);
731 struct stree *__pop_cond_true_stack(void);
732 struct stree *__pop_cond_false_stack(void);
733 void __and_cond_states(void);
734 void __or_cond_states(void);
735 void __save_pre_cond_states(void);
736 void __discard_pre_cond_states(void);
```



```
737 struct stree *__get_true_states(void);
738 struct stree *__get_false_states(void);
739 void __use_cond_states(void);
740 extern struct state_list *__last_base_slist;

742 void __push_continues(void);
743 void __discard_continues(void);
744 void __process_continues(void);
745 void __merge_continues(void);

747 void __push_breaks(void);
748 void __process_breaks(void);
749 int __has_breaks(void);
750 void __merge_breaks(void);
751 void __use_breaks(void);

753 void __save_switch_states(struct expression *switch_expr);
754 void __discard_switches(void);
755 int have_remaining_cases(void);
756 void __merge_switches(struct expression *switch_expr, struct range_list *case_rl
757 void __push_default(void);
758 void __set_default(void);
759 int __pop_default(void);

761 void __push_conditions(void);
762 void __discard_conditions(void);

764 void __save_gotos(const char *name, struct symbol *sym);
765 void __merge_gotos(const char *name, struct symbol *sym);

767 void __print_cur_stree(void);

769 /* smatch_hooks.c */
770 void __pass_to_client(void *data, enum hook_type type);
771 void __pass_to_client_no_data(enum hook_type type);
772 void __pass_case_to_client(struct expression *switch_expr,
773                            struct range_list *rl);
774 int __has_merge_function(int client_id);
775 struct smatch_state *__client_merge_function(int owner,
776                                              struct smatch_state *s1,
777                                              struct smatch_state *s2);
778 struct smatch_state *__client_unmatched_state_function(struct sm_state *sm);
779 void call_pre_merge_hook(struct sm_state *sm);
780 void __push_scope_hooks(void);
781 void __call_scope_hooks(void);

783 /* smatch_function_hooks.c */
784 void create_function_hook_hash(void);
785 void __match_initializer_call(struct symbol *sym);

787 /* smatch_db.c */
788 enum info_type {
789         INTERNAL        = 0,
790         /*
791          * Changing these numbers is a pain.  Don't do it.  If you ever use a
792          * number it can't be re-used right away so there may be gaps.
793          * We select these in order by type so if the order matters, then give
794          * it a number below 100-999,9000-9999 ranges. */

796         PARAM_CLEARED   = 101,
797         PARAM_LIMIT     = 103,
798         PARAM_FILTER    = 104,

800         PARAM_VALUE     = 1001,
801         BUF_SIZE        = 1002,
786         USER_DATA       = 1003,
```

```
802         CAPPED_DATA     = 1004,
803         RETURN_VALUE    = 1005,
804         DEREFERENCE     = 1006,
805         RANGE_CAP       = 1007,
806         LOCK_HELD       = 1008,
807         LOCK_RELEASED   = 1009,
808         ABSOLUTE_LIMITS = 1010,
809         PARAM_ADD       = 1012,
810         PARAM_FREED     = 1013,
811         DATA_SOURCE     = 1014,
812         FUZZY_MAX       = 1015,
813         HARD_MAX        = 2015,
814         STR_LEN         = 1016,
815         ARRAY_LEN       = 1017,
816         CAPABLE         = 1018,
817         NS_CAPABLE      = 1019,
818         CONTAINER       = 1020,
819         CASTED_CALL     = 1021,
820         TYPE_LINK       = 1022,
821         UNTRACKED_PARAM = 1023,
822         LOST_PARAM      = 2023,
823         CULL_PATH       = 1024,
824         PARAM_SET       = 1025,
825         PARAM_USED      = 1026,
826         BYTE_UNITS      = 1027,
827         COMPARE_LIMIT   = 1028,
828         PARAM_COMPARE   = 1029,
829         CONSTRAINT      = 1031,
830         PASSES_TYPE     = 1032,
831         CONSTRAINT_REQUIRED = 1033,
832         BIT_INFO        = 1034,
833         NOSPEC          = 1035,
834         NOSPEC_WB       = 1036,
835         STMT_CNT        = 1037,
836         TERMINATED      = 1038,

838         /* put random temporary stuff in the 7000-7999 range for testing */
839         USER_DATA       = 8017,
840         USER_DATA_SET   = 9017,
821         USER_DATA3      = 8017,
822         USER_DATA3_SET  = 9017,
841         NO_OVERFLOW     = 8018,
842         NO_OVERFLOW_SIMPLE = 8019,
843         LOCKED          = 8020,
844         UNLOCKED        = 8021,
845         SET_FS          = 8022,
846         ATOMIC_INC      = 8023,
847         ATOMIC_DEC      = 8024,
848         NO_SIDE_EFFECT  = 8025,
849         FN_ARG_LINK     = 8028,
850         DATA_VALUE      = 8029,
851         ARRAYSIZE_ARG   = 8033,
852         SIZEOF_ARG      = 8034,
853         MEMORY_TAG      = 8036,
854         MTAG_ASSIGN     = 8035,
855         STRING_VALUE    = 8041,

857         BYTE_COUNT      = 8050,
858         ELEM_COUNT      = 8051,
859         ELEM_LAST       = 8052,
860         USED_LAST       = 8053,
861         USED_COUNT      = 8054,
862 };

864 extern struct sqlite3 *smatch_db;
865 extern struct sqlite3 *mem_db;
```



```
866 extern struct sqlite3 *cache_db;

868 void db_ignore_states(int id);
869 void select_caller_info_hook(void (*callback)(const char *name, struct symbol *s
870 void add_member_info_callback(int owner, void (*callback)(struct expression *cal
871 void add_split_return_callback(void (*fn)(int return_id, char *return_ranges, st
872 void add_returned_member_callback(int owner, void (*callback)(int return_id, cha
873 void select_call_implies_hook(int type, void (*callback)(struct expression *call
874 void select_return_implies_hook(int type, void (*callback)(struct expression *ca
875 struct range_list *db_return_vals(struct expression *expr);
876 struct range_list *db_return_vals_from_str(const char *fn_name);
877 char *return_state_to_var_sym(struct expression *expr, int param, const char *ke
878 char *get_chunk_from_key(struct expression *arg, char *key, struct symbol **sym,
879 char *get_variable_from_key(struct expression *arg, const char *key, struct symb
880 const char *state_name_to_param_name(const char *state_name, const char *param_n
881 const char *get_param_name_var_sym(const char *name, struct symbol *sym);
882 const char *get_param_name(struct sm_state *sm);
883 const char *get_mtag_name_var_sym(const char *state_name, struct symbol *sym);
884 const char *get_mtag_name_expr(struct expression *expr);
885 char *get_data_info_name(struct expression *expr);
886 int is_recursive_member(const char *param_name);

888 char *escape_newlines(const char *str);
889 void sql_exec(struct sqlite3 *db, int (*callback)(void*, int, char**, char**), v

891 #define sql_helper(db, call_back, data, sql...)
892 do {
893         char sql_txt[1024];
894
895         sqlite3_snprintf(sizeof(sql_txt), sql_txt, sql);
896         sm_debug("debug: %s\n", sql_txt);
897         sql_exec(db, call_back, data, sql_txt);
898 } while (0)
______unchanged_portion_omitted_

951 #define sql_insert(table, values...) sql_insert_helper(table, 0, 0, 0, values);
952 #define sql_insert_or_ignore(table, values...) sql_insert_helper(table, 0, 1, 0,
953 #define sql_insert_late(table, values...) sql_insert_helper(table, 0, 0, 1, valu
954 #define sql_insert_cache(table, values...) sql_insert_helper(table, cache_db, 1,

956 char *get_static_filter(struct symbol *sym);

958 void sql_insert_return_states(int return_id, const char *return_ranges,
959                 int type, int param, const char *key, const char *value);
960 void sql_insert_caller_info(struct expression *call, int type, int param,
961                 const char *key, const char *value);
962 void sql_insert_function_ptr(const char *fn, const char *struct_name);
963 void sql_insert_return_values(const char *return_values);
964 void sql_insert_return_implies(int type, int param, const char *key, const char
965 void sql_insert_function_type_size(const char *member, const char *ranges);
966 void sql_insert_function_type_info(int type, const char *struct_type, const char
967 void sql_insert_type_info(int type, const char *member, const char *value);
968 void sql_insert_local_values(const char *name, const char *value);
969 void sql_insert_function_type_value(const char *type, const char *value);
970 void sql_insert_function_type(int param, const char *value);
971 void sql_insert_parameter_name(int param, const char *value);
972 void sql_insert_data_info(struct expression *data, int type, const char *value);
973 void sql_insert_data_info_var_sym(const char *var, struct symbol *sym, int type,
974 void sql_save_constraint(const char *con);
975 void sql_save_constraint_required(const char *data, int op, const char *limit);
976 void sql_copy_constraint_required(const char *new_limit, const char *old_limit);
977 void sql_insert_fn_ptr_data_link(const char *ptr, const char *data);
978 void sql_insert_fn_data_link(struct expression *fn, int type, int param, const c
979 void sql_insert_mtag_about(mtag_t tag, const char *left_name, const char *right_
955 void insert_mtag_data(sval_t sval, struct range_list *rl);
980 void sql_insert_mtag_map(mtag_t tag, int offset, mtag_t container);
```

```
 981 void sql_insert_mtag_alias(mtag_t orig, mtag_t alias);
 982 int mtag_map_select_container(mtag_t tag, int offset, mtag_t *container);
 983 int mtag_map_select_tag(mtag_t container, int offset, mtag_t *tag);
 984 struct smatch_state *swap_mtag_return(struct expression *expr, struct smatch_sta
 985 struct range_list *swap_mtag_seed(struct expression *expr, struct range_list *rl

 987 void sql_select_return_states(const char *cols, struct expression *call,
 988         int (*callback)(void*, int, char**, char**), void *info);
 989 void sql_select_call_implies(const char *cols, struct expression *call,
 990         int (*callback)(void*, int, char**, char**));

 992 void open_smatch_db(char *db_file);

 994 /* smatch_files.c */
 995 int open_data_file(const char *filename);
 996 int open_schema_file(const char *schema);
 997 struct token *get_tokens_file(const char *filename);

 999 /* smatch.c */
1000 extern char *option_debug_check;
1001 extern char *option_project_str;
1002 extern char *bin_dir;
1003 extern char *data_dir;
1004 extern int option_no_data;
1005 extern int option_full_path;
1006 extern int option_param_mapper;
1007 extern int option_call_tree;
1008 extern int num_checks;

1010 enum project_type {
1011         PROJ_NONE,
1012         PROJ_KERNEL,
1013         PROJ_WINE,
1014         PROJ_ILLUMOS_KERNEL,
1015         PROJ_ILLUMOS_USER,
1016         PROJ_UNKNOWN,
1017 };
1018 extern enum project_type option_project;
1019 const char *check_name(unsigned short id);
1020 int id_from_name(const char *name);


1023 /* smatch_buf_size.c */
1024 int get_array_size(struct expression *expr);
1025 int get_array_size_bytes(struct expression *expr);
1026 int get_array_size_bytes_min(struct expression *expr);
1027 int get_array_size_bytes_max(struct expression *expr);
1028 struct range_list *get_array_size_bytes_rl(struct expression *expr);
1029 int get_real_array_size(struct expression *expr);
1030 int last_member_is_resizable(struct symbol *type);
1031 /* smatch_strlen.c */
1032 int get_implied_strlen(struct expression *expr, struct range_list **rl);
1033 int get_size_from_strlen(struct expression *expr);

1035 /* smatch_capped.c */
1036 int is_capped(struct expression *expr);
1037 int is_capped_var_sym(const char *name, struct symbol *sym);

1039 /* check_user_data.c */
1040 int is_user_macro(struct expression *expr);
1041 int is_user_data(struct expression *expr);
1042 int is_capped_user_data(struct expression *expr);
1043 int implied_user_data(struct expression *expr, struct range_list **rl);
1044 struct stree *get_user_stree(void);
1045 int get_user_rl(struct expression *expr, struct range_list **rl);
1046 int get_user_rl_spammy(struct expression *expr, struct range_list **rl);
```



```
1045 int is_user_rl(struct expression *expr);
1046 int get_user_rl_var_sym(const char *name, struct symbol *sym, struct range_list
1047 bool user_rl_capped(struct expression *expr);
1048 struct range_list *var_user_rl(struct expression *expr);

1050 /* check_locking.c */
1051 void print_held_locks();

1053 /* check_assigned_expr.c */
1054 struct expression *get_assigned_expr(struct expression *expr);
1055 struct expression *get_assigned_expr_name_sym(const char *name, struct symbol *s
1056 /* smatch_return_to_param.c */
1057 void __add_return_to_param_mapping(struct expression *assign, const char *return
1058 char *map_call_to_param_name_sym(struct expression *expr, struct symbol **sym);

1060 /* smatch_comparison.c */
1061 struct compare_data {
1062         /* The ->left and ->right expression pointers might be NULL (I'm lazy) *
1063         struct expression *left;
1064         const char *left_var;
1065         struct var_sym_list *left_vsl;
1066         int comparison;
1067         struct expression *right;
1068         const char *right_var;
1069         struct var_sym_list *right_vsl;
1070 };
1071 DECLARE_ALLOCATOR(compare_data);
1072 struct smatch_state *alloc_compare_state(
1073                 struct expression *left,
1074                 const char *left_var, struct var_sym_list *left_vsl,
1075                 int comparison,
1076                 struct expression *right,
1077                 const char *right_var, struct var_sym_list *right_vsl);
1078 int filter_comparison(int orig, int op);
1079 int merge_comparisons(int one, int two);
1080 int combine_comparisons(int left_compare, int right_compare);
1081 int state_to_comparison(struct smatch_state *state);
1082 struct smatch_state *merge_compare_states(struct smatch_state *s1, struct smatch
1083 int get_comparison(struct expression *left, struct expression *right);
1084 int get_comparison_no_extra(struct expression *a, struct expression *b);
1085 int get_comparison_strings(const char *one, const char *two);
1086 int possible_comparison(struct expression *a, int comparison, struct expression
1087 struct state_list *get_all_comparisons(struct expression *expr);
1088 struct state_list *get_all_possible_equal_comparisons(struct expression *expr);
1089 void __add_return_comparison(struct expression *call, const char *range);
1090 void __add_comparison_info(struct expression *expr, struct expression *call, con
1091 char *get_printed_param_name(struct expression *call, const char *param_name, st
1092 char *name_sym_to_param_comparison(const char *name, struct symbol *sym);
1093 char *expr_equal_to_param(struct expression *expr, int ignore);
1094 char *expr_lte_to_param(struct expression *expr, int ignore);
1095 char *expr_param_comparison(struct expression *expr, int ignore);
1096 int flip_comparison(int op);
1097 int negate_comparison(int op);
1098 int remove_unsigned_from_comparison(int op);
1099 int param_compare_limit_is_impossible(struct expression *expr, int left_param, c
1100 void filter_by_comparison(struct range_list **rl, int comparison, struct range_l
1101 struct sm_state *comparison_implication_hook(struct expression *expr,
1102                         struct state_list **true_stack,
1103                         struct state_list **false_stack);
1104 void __compare_param_limit_hook(struct expression *left_expr, struct expression
1105                                 const char *state_name,
1106                                 struct smatch_state *true_state, struct smatch_s
1107 int impossibly_high_comparison(struct expression *expr);

1109 /* smatch_sval.c */
1110 sval_t *sval_alloc(sval_t sval);
```

```
sval_t *sval_alloc_permanent(sval_t sval);
sval_t sval_blank(struct expression *expr);
sval_t sval_type_val(struct symbol *type, long long val);
sval_t sval_from_val(struct expression *expr, long long val);
int sval_is_ptr(sval_t sval);
int sval_unsigned(sval_t sval);
int sval_signed(sval_t sval);
int sval_bits(sval_t sval);
int sval_bits_used(sval_t sval);
int sval_is_negative(sval_t sval);
int sval_is_positive(sval_t sval);
int sval_is_min(sval_t sval);
int sval_is_max(sval_t sval);
int sval_is_a_min(sval_t sval);
int sval_is_a_max(sval_t sval);
int sval_is_negative_min(sval_t sval);
int sval_cmp_t(struct symbol *type, sval_t one, sval_t two);
int sval_cmp_val(sval_t one, long long val);
sval_t sval_min(sval_t one, sval_t two);
sval_t sval_max(sval_t one, sval_t two);
int sval_too_low(struct symbol *type, sval_t sval);
int sval_too_high(struct symbol *type, sval_t sval);
int sval_fits(struct symbol *type, sval_t sval);
sval_t sval_cast(struct symbol *type, sval_t sval);
sval_t sval_preop(sval_t sval, int op);
sval_t sval_binop(sval_t left, int op, sval_t right);
int sval_binop_overflows(sval_t left, int op, sval_t right);
int sval_binop_overflows_no_sign(sval_t left, int op, sval_t right);
int find_first_zero_bit(unsigned long long uvalue);
int sm_fls64(unsigned long long uvalue);
unsigned long long fls_mask(unsigned long long uvalue);
unsigned long long sval_fls_mask(sval_t sval);
const char *sval_to_str(sval_t sval);
const char *sval_to_str_or_err_ptr(sval_t sval);
const char *sval_to_numstr(sval_t sval);
sval_t ll_to_sval(long long val);

/* smatch_string_list.c */
int list_has_string(struct string_list *str_list, const char *str);
int insert_string(struct string_list **str_list, const char *str);
void insert_string(struct string_list **str_list, const char *str);
struct string_list *clone_str_list(struct string_list *orig);
struct string_list *combine_string_lists(struct string_list *one, struct string_

/* smatch_start_states.c */
struct stree *get_start_states(void);

/* smatch_recurse.c */
int has_symbol(struct expression *expr, struct symbol *sym);
int has_variable(struct expression *expr, struct expression *var);
int has_inc_dec(struct expression *expr);

/* smatch_stored_conditions.c */
struct smatch_state *get_stored_condition(struct expression *expr);
struct expression_list *get_conditions(struct expression *expr);
struct sm_state *stored_condition_implication_hook(struct expression *expr,
                        struct state_list **true_stack,
                        struct state_list **false_stack);

/* check_string_len.c */
int get_formatted_string_size(struct expression *call, int arg);
int get_formatted_string_min_size(struct expression *call, int arg);

/* smatch_param_set.c */
int param_was_set(struct expression *expr);
int param_was_set_var_sym(const char *name, struct symbol *sym);
```



```
/* smatch_param_filter.c */
int param_has_filter_data(struct sm_state *sm);

/* smatch_links.c */
void set_up_link_functions(int id, int linkid);
struct smatch_state *merge_link_states(struct smatch_state *s1, struct smatch_st
void store_link(int link_id, const char *name, struct symbol *sym, const char *l

/* smatch_auto_copy.c */
void set_auto_copy(int owner);

/* check_buf_comparison */
const char *limit_type_str(unsigned int limit_type);
struct expression *get_size_variable(struct expression *buf, int *limit_type);
struct expression *get_size_variable(struct expression *buf);
struct expression *get_array_variable(struct expression *size);
int buf_comparison_index_ok(struct expression *expr);

/* smatch_untracked_param.c */
void mark_untracked(struct expression *expr, int param, const char *key, const c
void add_untracked_param_hook(void (func)(struct expression *call, int param));
void add_lost_param_hook(void (func)(struct expression *call, int param));
void mark_all_params_untracked(int return_id, char *return_ranges, struct expres

/* smatch_strings.c */
struct state_list *get_strings(struct expression *expr);
struct expression *fake_string_from_mtag(mtag_t tag);

/* smatch_estate.c */
int estate_get_single_value(struct smatch_state *state, sval_t *sval);

/* smatch_address.c */
int get_address_rl(struct expression *expr, struct range_list **rl);
int get_member_offset(struct symbol *type, const char *member_name);
int get_member_offset_from_deref(struct expression *expr);

/* for now this is in smatch_used_parameter.c */
void __get_state_hook(int owner, const char *name, struct symbol *sym);

/* smatch_buf_comparison.c */
int db_var_is_array_limit(struct expression *array, const char *name, struct var

struct stree *get_all_return_states(void);
struct stree_stack *get_all_return_strees(void);
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);
char *get_container_name(struct expression *container, 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_bit_info.c */
struct bit_info *get_bit_info(struct expression *expr);
struct bit_info *get_bit_info_var_sym(const char *name, struct symbol *sym);
/* smatch_mem_tracker.c */
extern int option_mem;
unsigned long get_mem_kb(void);
unsigned long get_max_memory(void);

/* check_is_nospec.c */
bool is_nospec(struct expression *expr);
long get_stmt_cnt(void);

/* smatch_nul_terminator.c */
bool is_nul_terminated(struct expression *expr);
/* check_kernel.c  */
bool is_ignored_kernel_data(const char *name);

static inline bool type_is_ptr(struct symbol *type)
{
        return type &&
               (type->type == SYM_PTR ||
                type->type == SYM_ARRAY ||
                type->type == SYM_FN);
}

static inline int type_bits(struct symbol *type)
{
        if (!type)
                return 0;
        if (type_is_ptr(type))
        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 (is_ptr_type(base_type))
                return 1;
        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 (is_ptr_type(type))
```



```
                return bits_in_pointer;
        if (type->type == SYM_ARRAY)
                return bits_in_pointer - 1;
        if (type_unsigned(type))
                return type_bits(type);
        return type_bits(type) - 1;
}
```

*unchanged_portion_omitted_*

****
5351 Mon Aug 5 08:38:14 2019
new/usr/src/tools/smatch/src/smatch_about_fn_ptr_arg.c
11506 smatch resync
****
_____***unchanged_portion_omitted_***

```
 109 static char *get_data_member(char *fn_member, struct expression *expr, struct sy
 110 {
 111         struct symbol *tmp_sym;
 112         char *fn_str;
 113         char *arg_ptr = NULL;
 114         char *end_type;
 115         int len_ptr, len_str;
 116         char buf[128];

 118         *sym = NULL;
 119         run_sql(get_arg_ptr, &arg_ptr,
 120                 "select data from fn_ptr_data_link where fn_ptr = '%s';", fn_mem
 121         if (!arg_ptr)
 122                 return NULL;
 123         end_type = strchr(arg_ptr, '>');
 124         if (!end_type)
 125                 return NULL;
 126         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, "%s", end_type);
 142         snprintf(buf + len_str, sizeof(buf) - len_str, end_type);
 143         *sym = tmp_sym;
 144         return alloc_string(buf);
 145 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    7086 Mon Aug  5 08:38:17 2019
new/usr/src/tools/smatch/src/smatch_address.c
11506 smatch resync
**********************************************************
______unchanged_portion_omitted_

 65 static bool matches_anonymous_union(struct symbol *sym, const char *member_name)
 66 {
 67         struct symbol *type, *tmp;

 69         if (sym->ident)
 70                 return false;
 71         type = get_real_base_type(sym);
 72         if (!type || type->type != SYM_UNION)
 73                 return false;

 75         FOR_EACH_PTR(type->symbol_list, tmp) {
 76                 if (tmp->ident &&
 77                     strcmp(member_name, tmp->ident->name) == 0) {
 78                         return true;
 79                 }
 80         } END_FOR_EACH_PTR(tmp);

 82         return false;
 83 }

 85 int get_member_offset(struct symbol *type, const char *member_name)
 86 {
 87         struct symbol *tmp;
 88         int offset;
 89         int bits;

 91         if (!type || type->type != SYM_STRUCT)
 92                 return -1;

 94         bits = 0;
 95         offset = 0;
 96         FOR_EACH_PTR(type->symbol_list, tmp) {
 97                 if (bits_to_bytes(bits + type_bits(tmp)) > tmp->ctype.alignment)
 98                         offset += bits_to_bytes(bits);
 99                         bits = 0;
100                 }
101                 offset = ALIGN(offset, tmp->ctype.alignment);
102                 if (tmp->ident &&
103                     strcmp(member_name, tmp->ident->name) == 0) {
104                         return offset;
105                 }
106                 if (matches_anonymous_union(tmp, member_name))
107                         return offset;
108                 if (!(type_bits(tmp) % 8) && type_bits(tmp) / 8 == type_bytes(tm
109                         offset += type_bytes(tmp);
110                 else
111                         bits += type_bits(tmp);
112         } END_FOR_EACH_PTR(tmp);
113         return -1;
114 }

116 int get_member_offset_from_deref(struct expression *expr)
117 {
118         struct symbol *type;
119         struct ident *member;
120         int offset;

122         if (expr->type != EXPR_DEREF)  /* hopefully, this doesn't happen */
123                 return -1;
```



```
125         if (expr->member_offset >= 0)
126                 return expr->member_offset;

128         member = expr->member;
129         if (!member)
130                 return -1;

132         type = get_type(expr->deref);
133         if (type_is_ptr(type))
134                 type = get_real_base_type(type);
135         if (!type || type->type != SYM_STRUCT)
136                 return -1;

138         offset = get_member_offset(type, member->name);
139         if (offset >= 0)
140                 expr->member_offset = offset;
141         return offset;
142 }

111 static struct range_list *filter_unknown_negatives(struct range_list *rl)
112 {
113         struct data_range *first;
114         struct range_list *filter = NULL;

116         first = first_ptr_list((struct ptr_list *)rl);

118         if (sval_is_min(first->min) &&
119             sval_is_negative(first->max) &&
120             first->max.value == -1) {
121                 add_ptr_list(&filter, first);
122                 return rl_filter(rl, filter);
123         }

125         return rl;
126 }

144 static void add_offset_to_pointer(struct range_list **rl, int offset)
145 {
146         sval_t min, max, remove, sval;
147         struct range_list *orig = *rl;

149         /*
150          * Ha ha.  Treating zero as a special case means I'm correct at least a
151          * tiny fraction of the time.  Which is better than nothing.
152          *
153          */
154         if (offset == 0)
155                 return;

157         if (is_unknown_ptr(orig))
158                 return;

160         /*
161          * This function doesn't necessarily work how you might expect...
162          *
163          * Say you have s64min-(-1),1-s64max and you add 8 then I guess what
164          * we want to say is maybe something like 9-s64max.  This shows that the
165          * min it could be is 9 which is potentially useful information.  But
166          * if we start with (-12),5000000-57777777 and we add 8 then we'd want
167          * the result to be (-4),5000008-57777777 but (-4),5000000-57777777 is
168          * also probably acceptable.  If you start with s64min-s64max then the
169          * result should be 8-s64max.
170          *
171          */
```

```
173         /* We do the math on void pointer type, because this isn't "&v + 16" it
174          * is &v->sixteenth_byte.
175          */
176         orig = cast_rl(&ptr_ctype, orig);
177         min = sval_type_min(&ptr_ctype);
178         min.value = offset;
179         max = sval_type_max(&ptr_ctype);

181         if (!orig || is_whole_rl(orig)) {
182                 *rl = alloc_rl(min, max);
183                 return;
184         }

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;

186         /* no wrap around */
187         max.uvalue = rl_max(orig).uvalue;
188         if (max.uvalue > sval_type_max(&ptr_ctype).uvalue - offset) {
189                 remove = sval_type_max(&ptr_ctype);
190                 remove.uvalue -= offset;
191                 orig = remove_range(orig, remove, max);
192         }

194         sval.type = &int_ctype;
195         sval.value = offset;

197         *rl = rl_binop(orig, '+', alloc_rl(sval, sval));
198 }

200 static struct range_list *where_allocated_rl(struct symbol *sym)
201 {
202         if (!sym)
203                 return NULL;

205         return alloc_rl(valid_ptr_min_sval, valid_ptr_max_sval);
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);
206 }

208 int get_address_rl(struct expression *expr, struct range_list **rl)
209 {
210         struct expression *unop;

212         expr = strip_expr(expr);
213         if (!expr)
214                 return 0;

216         if (expr->type == EXPR_STRING) {
217                 *rl = alloc_rl(valid_ptr_min_sval, valid_ptr_max_sval);
212                 *rl = alloc_rl(text_seg_min, text_seg_max);
218                 return 1;
219         }
```



```
221         if (expr->type == EXPR_PREOP && expr->op == '&')
222                 expr = strip_expr(expr->unop);
223         else {
224                 struct symbol *type;
216         if (expr->type == EXPR_PREOP && expr->op == '&') {
217                 struct expression *unop;

226                 type = get_type(expr);
227                 if (!type || type->type != SYM_ARRAY)
228                         return 0;
229         }

231         if (expr->type == EXPR_SYMBOL) {
232                 *rl = where_allocated_rl(expr->symbol);
219                 unop = strip_expr(expr->unop);
220                 if (unop->type == EXPR_SYMBOL) {
221                         *rl = where_allocated_rl(unop->symbol);
233                 return 1;
234         }

236         if (is_array(expr)) {
237                 struct expression *array;
238                 struct expression *offset_expr;
239                 struct range_list *array_rl, *offset_rl, *bytes_rl, *res;
240                 struct symbol *type;
241                 sval_t bytes;
225                 if (unop->type == EXPR_DEREF) {
226                         int offset = get_member_offset_from_deref(unop);

243                 array = get_array_base(expr);
244                 offset_expr = get_array_offset(expr);

246                 type = get_type(array);
247                 type = get_real_base_type(type);
248                 bytes.type = ssize_t_ctype;
249                 bytes.uvalue = type_bytes(type);
250                 bytes_rl = alloc_rl(bytes, bytes);

252                 get_absolute_rl(array, &array_rl);
253                 get_absolute_rl(offset_expr, &offset_rl);

255                 if (type_bytes(type)) {
256                         res = rl_binop(offset_rl, '*', bytes_rl);
257                         res = rl_binop(res, '+', array_rl);
258                         *rl = res;
259                         return true;
260                 }

262                 if (implied_not_equal(array, 0) ||
263                     implied_not_equal(offset_expr, 0)) {
264                         *rl = alloc_rl(valid_ptr_min_sval, valid_ptr_max_sval);
265                         return 1;
266                 }

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 {
268                 return 0;
269         }

271         if (expr->type == EXPR_DEREF && expr->member) {
272                 struct range_list *unop_rl;
```

```
273                 int offset;
274
275                 offset = get_member_offset_from_deref(expr);
276                 unop = strip_expr(expr->unop);
277                 if (unop->type == EXPR_PREOP && unop->op == '*')
278                         unop = strip_expr(unop->unop);
279
280                 if (offset >= 0 &&
281                     get_implied_rl(unop, &unop_rl) &&
282                     !is_whole_rl(unop_rl)) {
283                         *rl = unop_rl;
284                         add_offset_to_pointer(rl, offset);
285                         return 1;
286                 }
287
288                 if (implied_not_equal(unop, 0) || offset > 0) {
289                         *rl = alloc_rl(valid_ptr_min_sval, valid_ptr_max_sval);
290                         return 1;
291                 }
292
293                 return 0;
294         }
295
296         if (is_non_null_array(expr)) {
297                 *rl = alloc_rl(array_min_sval, array_max_sval);
298                 return 1;
299         }
300
301         return 0;
302 }
```

_____*unchanged_portion_omitted_*

****
4541 Mon Aug 5 08:38:19 2019
new/usr/src/tools/smatch/src/smatch_array_values.c
11506 smatch resync
****
_____unchanged_portion_omitted_

```
 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;

 166         type = get_type(expr->left);
 166         type = get_type(expr->right);
 167         if (!type || type->type != SYM_BASETYPE)
 168                 return;

 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;

 178         if (expr->op != '=') {
 179                 rl = alloc_whole_rl(get_type(expr->right));
 180                 rl = cast_rl(type, rl);
 179                 rl = alloc_whole_rl(type);
 181         } else {
 182                 get_absolute_rl(expr->right, &rl);
 183                 rl = cast_rl(type, rl);
 184                 orig_rl = get_saved_rl(type, name);
 185                 rl = rl_union(orig_rl, rl);
 186         }

 188         update_cache(name, is_file_local(array), rl);
 189 }
```

_____unchanged_portion_omitted_

```
**********************************************************
    3775 Mon Aug  5 08:38:19 2019
new/usr/src/tools/smatch/src/smatch_assigned_expr.c
11506 smatch resync
**********************************************************
```

```
/*
 * 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 struct expression *skip_mod;

static void undef(struct sm_state *sm, struct expression *mod_expr)
{
        if (mod_expr == skip_mod)
                return;
        set_state(my_id, sm->name, sm->sym, &undefined);
}
```

*unchanged_portion_omitted_*

```
static void match_assignment(struct expression *expr)
{
        static struct expression *ignored_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;
        }

        if (expr->left == ignored_expr)
```



```
                return;
        ignored_expr = NULL;
        if (__in_fake_parameter_assign)
                ignored_expr = expr->left;

        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;

        skip_mod = expr;
        set_state(my_id, name, sym, alloc_state_expr(right));
free:
        free_string(name);
}

void register_assigned_expr(int id)
{
        my_id = check_assigned_expr_id = id;
        set_dynamic_states(check_assigned_expr_id);
        add_hook(&match_assignment, ASSIGNMENT_HOOK_AFTER);
        add_modification_hook(my_id, &undef);
        select_return_states_hook(PARAM_SET, &record_param_assignment);
}
```

```
147 void register_assigned_expr_links(int id)
148 {
149         link_id = id;
150         set_dynamic_states(link_id);
151         db_ignore_states(link_id);
152         set_up_link_functions(my_id, link_id);
153 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   10625 Mon Aug  5 08:38:20 2019
new/usr/src/tools/smatch/src/smatch_bits.c
11506 smatch resync
**********************************************************
```

```
/*
 * 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
 */

/*
 * This is to track when variables are masked away.
 *
 */

#include "smatch.h"
#include "smatch_extra.h"
#include "smatch_slist.h"

static int my_id;

static const struct bit_info unknown_bit_info = {
        .possible = -1ULL,
};

ALLOCATOR(bit_info, "bit data");
static struct bit_info *alloc_bit_info(unsigned long long set, unsigned long lon
{
        struct bit_info *bit_info = __alloc_bit_info(0);

        bit_info->set = set;
        bit_info->possible = possible;

        return bit_info;
}

static struct smatch_state *alloc_bstate(unsigned long long set, unsigned long l
{
        struct smatch_state *state;
        char buf[64];

        state = __alloc_smatch_state(0);
        snprintf(buf, sizeof(buf), "0x%llx + 0x%llx", set, possible);
        state->name = alloc_sname(buf);
        state->data = alloc_bit_info(set, possible);

        return state;
}

static struct bit_info *rl_to_binfo(struct range_list *rl)
{
        struct bit_info *ret = __alloc_bit_info(0);
        sval_t sval;
```



```
        if (rl_to_sval(rl, &sval)) {
                ret->set = sval.uvalue;
                ret->possible = sval.uvalue;

                return ret;
        }

        ret->set = 0;
        ret->possible = sval_fls_mask(rl_max(rl));
        // FIXME: what about negatives?

        return ret;
}

static int is_unknown_binfo(struct symbol *type, struct bit_info *binfo)
{
        if (!type)
                type = &ullong_ctype;

        if (binfo->set != 0)
                return 0;
        if (binfo->possible < (-1ULL >> (64 - type_bits(type))))
                return 0;

        return 1;
}

static struct smatch_state *unmatched_state(struct sm_state *sm)
{
        struct smatch_state *estate;
        struct symbol *type;
        unsigned long long possible;
        struct bit_info *p;

        estate = get_state(SMATCH_EXTRA, sm->name, sm->sym);
        if (estate_rl(estate)) {
                p = rl_to_binfo(estate_rl(estate));
                return alloc_bstate(p->set, p->possible);
        }

        type = estate_type(estate);
        if (!type)
                return alloc_bstate(0, -1ULL);

        if (type_bits(type) == 64)
                possible = -1ULL;
        else
                possible = (1ULL << type_bits(type)) - 1;

        return alloc_bstate(0, possible);
}

static void match_modify(struct sm_state *sm, struct expression *mod_expr)
{
        // FIXME: we really need to store the type

        set_state(my_id, sm->name, sm->sym, alloc_bstate(0, -1ULL));
}

static int binfo_equiv(struct bit_info *one, struct bit_info *two)
{
        if (one->set == two->set &&
            one->possible == two->possible)
                return 1;
        return 0;
}
```

```
129 static struct smatch_state *merge_bstates(struct smatch_state *one_state, struct
130 {
131         struct bit_info *one, *two;

133         one = one_state->data;
134         two = two_state->data;

136         if (binfo_equiv(one, two))
137                 return one_state;

139         return alloc_bstate(one->set & two->set, one->possible | two->possible);
140 }

142 /*
143  * The combine_bit_info() takes two bit_infos and takes creates the most
144  * accurate picture we can assuming both are true.  Or it returns unknown if
145  * the information is logically impossible.
146  *
147  * Which means that it takes the | of the ->set bits and the & of the possibly
148  * set bits, which is the opposite of what merge_bstates() does.
149  *
150  */
151 static struct bit_info *combine_bit_info(struct bit_info *one, struct bit_info *
152 {
153         struct bit_info *ret = __alloc_bit_info(0);

155         if ((one->set & two->possible) != one->set)
156                 return alloc_bit_info(0, -1ULL);
157         if ((two->set & one->possible) != two->set)
158                 return alloc_bit_info(0, -1ULL);

160         ret->set = one->set | two->set;
161         ret->possible = one->possible & two->possible;

163         return ret;
164 }

166 static struct bit_info *binfo_AND(struct bit_info *left, struct bit_info *right)
167 {
168         unsigned long long set = 0;
169         unsigned long long possible = -1ULL;

171         if (!left && !right) {
172                 /* nothing */
173         } else if (!left) {
174                 possible = right->possible;
175         } else if (!right) {
176                 possible = left->possible;
177         } else {
178                 set = left->set & right->set;
179                 possible = left->possible & right->possible;
180         }

182         return alloc_bit_info(set, possible);
183 }

185 static struct bit_info *binfo_OR(struct bit_info *left, struct bit_info *right)
186 {
187         unsigned long long set = 0;
188         unsigned long long possible = -1ULL;

190         if (!left && !right) {
191                 /* nothing */
192         } else if (!left) {
193                 set = right->set;
```



```
194         } else if (!right) {
195                 set = left->set;
196         } else {
197                 set = left->set | right->set;
198                 possible = left->possible | right->possible;
199         }

201         return alloc_bit_info(set, possible);
202 }

204 struct bit_info *get_bit_info(struct expression *expr)
205 {
206         struct range_list *rl;
207         struct smatch_state *bstate;
208         struct bit_info tmp;
209         struct bit_info *extra_info;
210         struct bit_info *bit_info;
211         sval_t known;

213         expr = strip_parens(expr);

215         if (get_implied_value(expr, &known))
216                 return alloc_bit_info(known.value, known.value);

218         if (expr->type == EXPR_BINOP) {
219                 if (expr->op == '&')
220                         return binfo_AND(get_bit_info(expr->left),
221                                          get_bit_info(expr->right));
222                 if (expr->op == '|')
223                         return binfo_OR(get_bit_info(expr->left),
224                                         get_bit_info(expr->right));
225         }

227         if (get_implied_rl(expr, &rl))
228                 extra_info = rl_to_binfo(rl);
229         else {
230                 struct symbol *type;

232                 tmp = unknown_bit_info;
233                 extra_info = &tmp;

235                 type = get_type(expr);
236                 if (!type)
237                         type = &ullong_ctype;
238                 if (type_bits(type) == 64)
239                         extra_info->possible = -1ULL;
240                 else
241                         extra_info->possible = (1ULL << type_bits(type)) - 1;
242         }

244         bstate = get_state_expr(my_id, expr);
245         if (bstate)
246                 bit_info = bstate->data;
247         else
248                 bit_info = (struct bit_info *)&unknown_bit_info;

250         return combine_bit_info(extra_info, bit_info);
251 }

253 static int is_single_bit(sval_t sval)
254 {
255         int i;
256         int count = 0;

258         for (i = 0; i < 64; i++) {
259                 if (sval.uvalue & 1ULL << i &&
```

```
260                     count++)
261                         return 0;
262         }
263         if (count == 1)
264                 return 1;
265         return 0;
266 }
267
268 static void match_compare(struct expression *expr)
269 {
270         sval_t val;
271
272         if (expr->type != EXPR_COMPARE)
273                 return;
274         if (expr->op != SPECIAL_EQUAL &&
275             expr->op != SPECIAL_NOTEQUAL)
276                 return;
277
278         if (!get_implied_value(expr->right, &val))
279                 return;
280
281         set_true_false_states_expr(my_id, expr->left,
282                         (expr->op == SPECIAL_EQUAL) ? alloc_bstate(val.uvalue, v
283                         (expr->op == SPECIAL_EQUAL) ? NULL : alloc_bstate(val.uv
284 }
285
286 static bool is_loop_iterator(struct expression *expr)
287 {
288         struct statement *pre_stmt, *loop_stmt;
289
290         pre_stmt = expr_get_parent_stmt(expr);
291         if (!pre_stmt || pre_stmt->type != STMT_EXPRESSION)
292                 return false;
293
294         loop_stmt = stmt_get_parent_stmt(pre_stmt);
295         if (!loop_stmt || loop_stmt->type != STMT_ITERATOR)
296                 return false;
297         if (loop_stmt->iterator_pre_statement != pre_stmt)
298                 return false;
299
300         return true;
301 }
302
303 static void match_assign(struct expression *expr)
304 {
305         struct bit_info *binfo;
306
307         if (expr->op != '=')
308                 return;
309         if (__in_fake_assign)
310                 return;
311         if (is_loop_iterator(expr))
312                 return;
313
314         binfo = get_bit_info(expr->right);
315         if (!binfo)
316                 return;
317         if (is_unknown_binfo(get_type(expr->left), binfo))
318                 return;
319         set_state_expr(my_id, expr->left, alloc_bstate(binfo->set, binfo->possib
320 }
321
322 static void match_condition(struct expression *expr)
323 {
324         struct bit_info *orig;
325         struct bit_info true_info;
```



```
326         struct bit_info false_info;
327         sval_t right;
328
329         if (expr->type != EXPR_BINOP ||
330             expr->op != '&')
331                 return;
332
333         if (!get_value(expr->right, &right))
334                 return;
335
336         orig = get_bit_info(expr->left);
337         true_info = *orig;
338         false_info = *orig;
339
340         if (right.uvalue == 0 || is_single_bit(right))
341                 true_info.set &= right.uvalue;
342
343         true_info.possible &= right.uvalue;
344         false_info.possible &= ~right.uvalue;
345
346         set_true_false_states_expr(my_id, expr->left,
347                                    alloc_bstate(true_info.set, true_info.possibl
348                                    alloc_bstate(false_info.set, false_info.possi
349 }
350
351 static void match_call_info(struct expression *expr)
352 {
353         struct bit_info *binfo, *rl_binfo;
354         struct expression *arg;
355         struct range_list *rl;
356         char buf[64];
357         int i;
358
359         i = -1;
360         FOR_EACH_PTR(expr->args, arg) {
361                 i++;
362                 binfo = get_bit_info(arg);
363                 if (!binfo)
364                         continue;
365                 if (is_unknown_binfo(get_type(arg), binfo))
366                         continue;
367                 if (get_implied_rl(arg, &rl)) {
368                         rl_binfo = rl_to_binfo(rl);
369                         if (binfo_equiv(rl_binfo, binfo))
370                                 continue;
371                 }
372                 // If is just non-negative continue
373                 // If ->set == ->possible continue
374                 snprintf(buf, sizeof(buf), "0x%llx,0x%llx", binfo->set, binfo->p
375                 sql_insert_caller_info(expr, BIT_INFO, i, "$", buf);
376         } END_FOR_EACH_PTR(arg);
377 }
378
379 static void struct_member_callback(struct expression *call, int param, char *pri
380 {
381         struct bit_info *binfo = sm->state->data;
382         struct smatch_state *estate;
383         struct bit_info *implied_binfo;
384         char buf[64];
385
386         if (!binfo)
387                 return;
388
389         /* This means it can only be one value, so it's handled by smatch_extra.
390         if (binfo->set == binfo->possible)
391                 return;
```

```
393         estate = get_state(SMATCH_EXTRA, sm->name, sm->sym);
394         if (is_unknown_binfo(estate_type(estate), binfo))
395                 return;

397         if (estate_rl(estate)) {
398                 sval_t sval;

400                 if (estate_get_single_value(estate, &sval))
401                         return;

403                 implied_binfo = rl_to_binfo(estate_rl(estate));
404                 if (binfo_equiv(implied_binfo, binfo))
405                         return;
406         }

408         snprintf(buf, sizeof(buf), "0x%llx,0x%llx", binfo->set, binfo->possible)
409         sql_insert_caller_info(call, BIT_INFO, param, printed_name, buf);
410 }

412 static void set_param_bits(const char *name, struct symbol *sym, char *key, char
413 {
414         char fullname[256];
415         unsigned long long set, possible;

417         if (strcmp(key, "*$") == 0)
418                 snprintf(fullname, sizeof(fullname), "*%s", name);
419         else if (strncmp(key, "$", 1) == 0)
420                 snprintf(fullname, 256, "%s%s", name, key + 1);
421         else
422                 return;

424         set = strtoull(value, &value, 16);
425         if (*value != ',')
426                 return;
427         value++;
428         possible = strtoull(value, &value, 16);

430         set_state(my_id, fullname, sym, alloc_bstate(set, possible));
431 }

433 void register_bits(int id)
434 {
435         my_id = id;

437         set_dynamic_states(my_id);

439         add_unmatched_state_hook(my_id, &unmatched_state);
440         add_merge_hook(my_id, &merge_bstates);

442         add_hook(&match_condition, CONDITION_HOOK);
443         add_hook(&match_compare, CONDITION_HOOK);
444         add_hook(&match_assign, ASSIGNMENT_HOOK);
445         add_modification_hook(my_id, &match_modify);

447         add_hook(&match_call_info, FUNCTION_CALL_HOOK);
448         add_member_info_callback(my_id, struct_member_callback);
449         select_caller_info_hook(set_param_bits, BIT_INFO);
450 }
```

```
**********************************************************
   20837 Mon Aug  5 08:38:20 2019
new/usr/src/tools/smatch/src/smatch_buf_comparison.c
11506 smatch resync
**********************************************************
```

```
   1 /*
   2  * Copyright (C) 2012 Oracle.
   3  *
   4  * This program is free software; you can redistribute it and/or
   5  * modify it under the terms of the GNU General Public License
   6  * as published by the Free Software Foundation; either version 2
   7  * of the License, or (at your option) any later version.
   8  *
   9  * This program is distributed in the hope that it will be useful,
  10  * but WITHOUT ANY WARRANTY; without even the implied warranty of
  11  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  12  * GNU General Public License for more details.
  13  *
  14  * You should have received a copy of the GNU General Public License
  15  * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
  16  */

  18 /*
  19  * The point here is to store that a buffer has x bytes even if we don't know
  20  * the value of x.
  21  *
  22  */

  24 #include "smatch.h"
  25 #include "smatch_extra.h"
  26 #include "smatch_slist.h"

  28 static int size_id;
  29 static int link_id;

  31 /*
  32  * There is a bunch of code which does this:
  32  * We need this for code which does:
  33  *
  34  *     if (size)
  35  *         foo = malloc(size);
  36  *
  37  * So if "size" is non-zero then the size of "foo" is size.  But really it's
  38  * also true if size is zero.  It's just better to assume to not trample over
  39  * the data that we have by merging &undefined states.
  37  * We want to record that the size of "foo" is "size" even after the merge.
  40  *
  41  */
  42 static struct smatch_state *unmatched_state(struct sm_state *sm)
  43 {
  42         struct expression *size_expr;
  43         sval_t sval;

  45         if (!sm->state->data)
  46                 return &undefined;
  47         size_expr = sm->state->data;
  48         if (!get_implied_value(size_expr, &sval) || sval.value != 0)
  49                 return &undefined;
  44         return sm->state;
  45 }
_____unchanged_portion_omitted_

  79 static const char *limit_map[] = {
  80         "byte_count",
  81         "elem_count",
  82         "elem_last",
```



```
  83         "used_count",
  84         "used_last",
  85 };

  87 int state_to_limit(struct smatch_state *state)
  85 static struct smatch_state *alloc_expr_state(struct expression *expr)
  88 {
  89         int i;

  91         if (!state || !state->data)
  92                 return -1;

  94         for (i = 0; i < ARRAY_SIZE(limit_map); i++) {
  95                 if (strncmp(state->name, limit_map[i], strlen(limit_map[i])) ==
  96                         return i + BYTE_COUNT;
  97         }

  99         return -1;
 100 }

 102 const char *limit_type_str(unsigned int limit_type)
 103 {
 104         if (limit_type - BYTE_COUNT >= ARRAY_SIZE(limit_map)) {
 105                 sm_msg("internal: wrong size type %u", limit_type);
 106                 return "unknown";
 107         }

 109         return limit_map[limit_type - BYTE_COUNT];
 110 }

 112 static struct smatch_state *alloc_compare_size(int limit_type, struct expression
 113 {
 114         struct smatch_state *state;
 115         char *name;
 116         char buf[256];

 118         state = __alloc_smatch_state(0);
 119         expr = strip_expr(expr);
 120         name = expr_to_str(expr);
 121         snprintf(buf, sizeof(buf), "%s %s", limit_type_str(limit_type), name);
 122         state->name = alloc_sname(buf);
  93         state->name = alloc_sname(name);
 123         free_string(name);
 124         state->data = expr;
 125         return state;
 126 }
_____unchanged_portion_omitted_

 143 static void db_save_type_links(struct expression *array, int type_limit, struct
 114 static void db_save_type_links(struct expression *array, struct expression *size
 144 {
 145         const char *array_name;

 147         array_name = get_data_info_name(array);
 148         if (!array_name)
 149                 array_name = "";
 150         sql_insert_data_info(size, type_limit, array_name);
 121         sql_insert_data_info(size, ARRAY_LEN, array_name);
 151 }

 153 static void match_alloc_helper(struct expression *pointer, struct expression *si
 154 {
 155         struct expression *tmp;
 156         struct sm_state *sm;
 157         int limit_type = ELEM_COUNT;
 158         sval_t sval;
```

```
159         int cnt = 0;

161         pointer = strip_expr(pointer);
162         size = strip_expr(size);
163         if (!size || !pointer)
164                 return;

166         while ((tmp = get_assigned_expr(size))) {
167                 size = strip_expr(tmp);
168                 if (cnt++ > 5)
169                         break;
170         }

172         if (size->type == EXPR_BINOP && size->op == '*') {
173                 struct expression *mult_left, *mult_right;

175                 mult_left = strip_expr(size->left);
176                 mult_right = strip_expr(size->right);

178                 if (get_implied_value(mult_left, &sval) &&
179                     sval.value == bytes_per_element(pointer))
180                         size = mult_right;
181                 else if (get_implied_value(mult_right, &sval) &&
182                     sval.value == bytes_per_element(pointer))
183                         size = mult_left;
184                 else
185                         return;
186         }

188         /* Only save links to variables, not fixed sizes */
189         if (get_value(size, &sval))
190                 return;

192         if (size->type == EXPR_BINOP && size->op == '+' &&
193             get_value(size->right, &sval) && sval.value == 1) {
194                 size = size->left;
195                 limit_type = ELEM_LAST;
196         }

198         db_save_type_links(pointer, limit_type, size);
199         sm = set_state_expr(size_id, pointer, alloc_compare_size(limit_type, siz
162         db_save_type_links(pointer, size);
163         sm = set_state_expr(size_id, pointer, alloc_expr_state(size));
200         if (!sm)
201                 return;
202         set_state_expr(link_id, size, alloc_state_expr(pointer));
166         set_state_expr(link_id, size, alloc_expr_state(pointer));
203 }
_____unchanged_portion_omitted_

216 static void match_calloc(const char *fn, struct expression *expr, void *_start_a
217 {
218         int start_arg = PTR_INT(_start_arg);
219         struct expression *pointer, *call, *arg;
220         struct sm_state *tmp;
221         int limit_type = ELEM_COUNT;
222         sval_t sval;

224         pointer = strip_expr(expr->left);
225         call = strip_expr(expr->right);
226         arg = get_argument_from_call_expr(call->args, start_arg);
227         if (get_implied_value(arg, &sval) &&
228             sval.value == bytes_per_element(pointer))
229                 arg = get_argument_from_call_expr(call->args, start_arg + 1);

231         if (arg->type == EXPR_BINOP && arg->op == '+' &&
```



```
232             get_value(arg->right, &sval) && sval.value == 1) {
233                 arg = arg->left;
234                 limit_type = ELEM_LAST;
235         }

237         db_save_type_links(pointer, limit_type, arg);
238         tmp = set_state_expr(size_id, pointer, alloc_compare_size(limit_type, ar
194         db_save_type_links(pointer, arg);
195         tmp = set_state_expr(size_id, pointer, alloc_expr_state(arg));
239         if (!tmp)
240                 return;
241         set_state_expr(link_id, arg, alloc_state_expr(pointer));
198         set_state_expr(link_id, arg, alloc_expr_state(pointer));
242 }

244 struct expression *get_size_variable(struct expression *buf, int *limit_type)
201 struct expression *get_size_variable(struct expression *buf)
245 {
246         struct smatch_state *state;

248         state = get_state_expr(size_id, buf);
249         if (!state)
250                 return NULL;
251         *limit_type = state_to_limit(state);
206         if (state)
252                 return state->data;
208         return NULL;
253 }
_____unchanged_portion_omitted_

265 static void array_check(struct expression *expr)
266 {
267         struct expression *array;
268         struct expression *size;
269         struct expression *offset;
270         char *array_str, *offset_str;
271         int limit_type;

273         expr = strip_expr(expr);
274         if (!is_array(expr))
275                 return;

277         array = get_array_base(expr);
278         size = get_size_variable(array, &limit_type);
233         size = get_size_variable(array);
279         if (!size)
280                 return;
281         if (limit_type != ELEM_COUNT)
282                 return;
283         offset = get_array_offset(expr);
284         if (!possible_comparison(size, SPECIAL_EQUAL, offset))
285                 return;

287         array_str = expr_to_str(array);
288         offset_str = expr_to_str(offset);
289         sm_warning("potentially one past the end of array '%s[%s]'", array_str,
290         free_string(array_str);
291         free_string(offset_str);
292 }
_____unchanged_portion_omitted_

367 int buf_comparison_index_ok(struct expression *expr)
320 static int known_access_ok_comparison(struct expression *expr)
368 {
369         struct expression *array;
370         struct expression *size;
```

```
371         struct expression *offset;
372         int limit_type;
373         int comparison;

375         array = get_array_base(expr);
376         size = get_size_variable(array, &limit_type);
328         size = get_size_variable(array);
377         if (!size)
378                 return 0;
379         offset = get_array_offset(expr);
380         comparison = get_comparison(offset, size);
381         if (!comparison)
382                 return 0;

384         if ((limit_type == ELEM_COUNT || limit_type == ELEM_LAST) &&
385             (comparison == '<' || comparison == SPECIAL_UNSIGNED_LT))
332         comparison = get_comparison(size, offset);
333         if (comparison == '>' || comparison == SPECIAL_UNSIGNED_GT)
386                 return 1;
387         if (limit_type == ELEM_LAST &&
388             (comparison == SPECIAL_LTE ||
389              comparison == SPECIAL_UNSIGNED_LTE ||
390              comparison == SPECIAL_EQUAL))
391                 return 1;

393         return 0;
394 }
```

*unchanged_portion_omitted_*

```
416 static void array_check_data_info(struct expression *expr)
417 {
418         struct expression *array;
419         struct expression *offset;
420         struct state_list *slist;
421         struct sm_state *sm;
422         struct compare_data *comp;
423         char *offset_name;
424         const char *equal_name = NULL;

426         expr = strip_expr(expr);
427         if (!is_array(expr))
428                 return;

430         if (known_access_ok_numbers(expr))
431                 return;
432         if (buf_comparison_index_ok(expr))
375         if (known_access_ok_comparison(expr))
433                 return;

435         array = get_array_base(expr);
436         offset = get_array_offset(expr);
437         offset_name = expr_to_var(offset);
438         if (!offset_name)
439                 return;
440         slist = get_all_possible_equal_comparisons(offset);
441         if (!slist)
442                 goto free;

444         FOR_EACH_PTR(slist, sm) {
445                 comp = sm->state->data;
446                 if (strcmp(comp->left_var, offset_name) == 0) {
447                         if (db_var_is_array_limit(array, comp->right_var, comp->
448                                 equal_name = comp->right_var;
449                                 break;
450                         }
451                 } else if (strcmp(comp->right_var, offset_name) == 0) {
```



```
452                         if (db_var_is_array_limit(array, comp->left_var, comp->l
453                                 equal_name = comp->left_var;
454                                 break;
455                         }
456                 }
457         } END_FOR_EACH_PTR(sm);

459         if (equal_name) {
460                 char *array_name = expr_to_str(array);

462                 sm_warning("potential off by one '%s[]' limit '%s'", array_name,
463                 free_string(array_name);
464         }

466 free:
467         free_slist(&slist);
468         free_string(offset_name);
469 }
```

*unchanged_portion_omitted_*

```
476 static int is_sizeof(struct expression *expr)
419 static char *buf_size_param_comparison(struct expression *array, struct expressi
477 {
478         const char *name;

480         if (expr->type == EXPR_SIZEOF)
481                 return 1;
482         name = pos_ident(expr->pos);
483         if (name && strcmp(name, "sizeof") == 0)
484                 return 1;
485         return 0;
486 }

488 static int match_size_binop(struct expression *size, struct expression *expr, in
489 {
490         int orig_type = *limit_type;
491         struct expression *left;
492         sval_t sval;

494         left = expr->left;
495         if (!expr_equiv(size, left))
496                 return 0;

498         if (expr->op == '-' &&
499             get_value(expr->right, &sval) &&
500             sval.value == 1 &&
501             orig_type == ELEM_COUNT) {
502                 *limit_type = ELEM_LAST;
503                 return 1;
504         }

506         if (expr->op == '+' &&
507             get_value(expr->right, &sval) &&
508             sval.value == 1 &&
509             orig_type == ELEM_LAST) {
510                 *limit_type = ELEM_COUNT;
511                 return 1;
512         }

514         if (expr->op == '*' &&
515             is_sizeof(expr->right) &&
516             orig_type == ELEM_COUNT) {
517                 *limit_type = BYTE_COUNT;
518                 return 1;
519         }
```

```
521         if (expr->op == '/' &&
522             is_sizeof(expr->right) &&
523             orig_type == BYTE_COUNT) {
524                 *limit_type = ELEM_COUNT;
525                 return 1;
526         }

528         return 0;
529 }

531 static char *buf_size_param_comparison(struct expression *array, struct expressi
532 {
533         struct expression *tmp, *arg;
421         struct expression *arg;
534         struct expression *size;
535         static char buf[32];
536         int i;

538         size = get_size_variable(array, limit_type);
426         size = get_size_variable(array);
539         if (!size)
540                 return NULL;

542         if (*limit_type == USED_LAST)
543                 *limit_type = ELEM_LAST;
544         if (*limit_type == USED_COUNT)
545                 *limit_type = ELEM_COUNT;

547         i = -1;
548         FOR_EACH_PTR(args, tmp) {
431         FOR_EACH_PTR(args, arg) {
549                 i++;
550                 arg = tmp;
551                 if (arg == array)
552                         continue;
553                 if (expr_equiv(arg, size) ||
554                     (arg->type == EXPR_BINOP &&
555                      match_size_binop(size, arg, limit_type))) {
435                 if (!expr_equiv(arg, size))
436                         continue;
556                         snprintf(buf, sizeof(buf), "==$%d", i);
557                         return buf;
558                 }
559         } END_FOR_EACH_PTR(tmp);
439         } END_FOR_EACH_PTR(arg);

561         return NULL;
562 }

564 static void match_call(struct expression *call)
565 {
566         struct expression *arg;
567         char *compare;
568         int param;
569         char buf[5];
570         int limit_type;

572         param = -1;
573         FOR_EACH_PTR(call->args, arg) {
574                 param++;
575                 if (!is_pointer(arg))
576                         continue;
577                 compare = buf_size_param_comparison(arg, call->args, &limit_type
455                 compare = buf_size_param_comparison(arg, call->args);
578                 if (!compare)
579                         continue;
```



```
580                 snprintf(buf, sizeof(buf), "%d", limit_type);
581                 sql_insert_caller_info(call, limit_type, param, compare, buf);
458                 sql_insert_caller_info(call, ARRAY_LEN, param, "$", compare);
582         } END_FOR_EACH_PTR(arg);
583 }
______unchanged_portion_omitted_

613 static void set_param_compare(const char *array_name, struct symbol *array_sym,
614 {
615         struct expression *array_expr;
616         struct expression *size_expr;
617         struct symbol *size_sym;
618         char *size_name;
619         long param;
620         struct sm_state *tmp;
621         int limit_type;

623         if (strncmp(key, "==$", 3) != 0)
499         if (strncmp(value, "==$", 3) != 0)
624                 return;
625         param = strtol(key + 3, NULL, 10);
501         param = strtol(value + 3, NULL, 10);
626         if (!get_param(param, &size_name, &size_sym))
627                 return;
628         array_expr = symbol_expression(array_sym);
629         size_expr = symbol_expression(size_sym);
630         limit_type = strtol(value, NULL, 10);

632         tmp = set_state_expr(size_id, array_expr, alloc_compare_size(limit_type,
507         tmp = set_state_expr(size_id, array_expr, alloc_expr_state(size_expr));
633         if (!tmp)
634                 return;
635         set_state_expr(link_id, size_expr, alloc_state_expr(array_expr));
510         set_state_expr(link_id, size_expr, alloc_expr_state(array_expr));
636 }

638 static void set_implied(struct expression *call, struct expression *array_expr,
513 static void set_arraysize_arg(const char *array_name, struct symbol *array_sym,
639 {
515         struct expression *array_expr;
640         struct expression *size_expr;
641         struct symbol *size_sym;
642         char *size_name;
643         long param;
644         struct sm_state *tmp;
645         int limit_type;

647         if (strncmp(key, "==$", 3) != 0)
648                 return;
649         param = strtol(key + 3, NULL, 10);
522         param = strtol(key, NULL, 10);
650         if (!get_param(param, &size_name, &size_sym))
651                 return;
525         array_expr = symbol_expression(array_sym);
652         size_expr = symbol_expression(size_sym);

654         limit_type = strtol(value, NULL, 10);
655         tmp = set_state_expr(size_id, array_expr, alloc_compare_size(limit_type,
528         tmp = set_state_expr(size_id, array_expr, alloc_expr_state(size_expr));
656         if (!tmp)
657                 return;
658         set_state_expr(link_id, size_expr, alloc_state_expr(array_expr));
531         set_state_expr(link_id, size_expr, alloc_expr_state(array_expr));
659 }
______unchanged_portion_omitted_
```

```
690 static void set_used(struct expression *expr)
691 {
692         struct expression *parent;
693         struct expression *array;
694         struct expression *offset;
695         struct sm_state *tmp;
696         int limit_type;

698         if (expr->op != SPECIAL_INCREMENT)
699                 return;

701         limit_type = USED_LAST;
702         if (expr->type == EXPR_POSTOP)
703                 limit_type = USED_COUNT;

705         parent = expr_get_parent_expr(expr);
706         if (!parent || parent->type != EXPR_BINOP)
707                 return;
708         parent = expr_get_parent_expr(parent);
709         if (!parent || !is_array(parent))
710                 return;

712         array = get_array_base(parent);
713         offset = get_array_offset(parent);
714         if (offset != expr)
715                 return;

717         tmp = set_state_expr(size_id, array, alloc_compare_size(limit_type, offs
718         if (!tmp)
719                 return;
720         set_state_expr(link_id, offset->unop, alloc_state_expr(array));
721 }

723 static int match_assign_array(struct expression *expr)
724 {
725         // FIXME: implement
726         return 0;
727 }

729 static int match_assign_size(struct expression *expr)
730 {
731         struct expression *right, *size, *array;
732         struct smatch_state *state;
733         struct sm_state *tmp;
734         int limit_type;

736         right = expr->right;
737         size = right;
738         if (size->type == EXPR_BINOP)
739                 size = size->left;

741         array = get_array_variable(size);
742         if (!array)
743                 return 0;
744         state = get_state_expr(size_id, array);
745         if (!state || !state->data)
746                 return 0;

748         limit_type = state_to_limit(state);
749         if (limit_type < 0)
750                 return 0;

752         if (right->type == EXPR_BINOP && !match_size_binop(size, right, &limit_t
753                 return 0;

755         tmp = set_state_expr(size_id, array, alloc_compare_size(limit_type, expr
```



```
756         if (!tmp)
757                 return 0;
758         set_state_expr(link_id, expr->left, alloc_state_expr(array));
759         return 1;
760 }

762 static void match_assign(struct expression *expr)
763 {
764         if (expr->op != '=')
765                 return;

767         if (match_assign_array(expr))
768                 return;
769         match_assign_size(expr);
770 }

772 static void match_copy(const char *fn, struct expression *expr, void *unused)
773 {
774         struct expression *src, *size;
775         int src_param, size_param;

777         src = get_argument_from_call_expr(expr->args, 1);
778         size = get_argument_from_call_expr(expr->args, 2);
779         src = strip_expr(src);
780         size = strip_expr(size);
781         if (!src || !size)
782                 return;
783         if (src->type != EXPR_SYMBOL || size->type != EXPR_SYMBOL)
784                 return;

786         src_param = get_param_num_from_sym(src->symbol);
787         size_param = get_param_num_from_sym(size->symbol);
788         if (src_param < 0 || size_param < 0)
789                 return;

791         sql_insert_cache(call_implies, "'%s', '%s', 0, %d, %d, %d, '==$%d', '%d'
792                          get_base_file(), get_function(), fn_static(),
793                          BYTE_COUNT, src_param, size_param, BYTE_COUNT);
794 }

796 void register_buf_comparison(int id)
797 {
798         int i;

800         size_id = id;

802         set_dynamic_states(size_id);

804         add_unmatched_state_hook(size_id, &unmatched_state);

806         add_allocation_function("malloc", &match_alloc, 0);
807         add_allocation_function("memdup", &match_alloc, 1);
808         add_allocation_function("realloc", &match_alloc, 1);
809         if (option_project == PROJ_KERNEL) {
810                 add_allocation_function("kmalloc", &match_alloc, 0);
811                 add_allocation_function("kzalloc", &match_alloc, 0);
812                 add_allocation_function("vmalloc", &match_alloc, 0);
813                 add_allocation_function("__vmalloc", &match_alloc, 0);
814                 add_allocation_function("sock_kmalloc", &match_alloc, 1);
815                 add_allocation_function("kmemdup", &match_alloc, 1);
816                 add_allocation_function("kmemdup_user", &match_alloc, 1);
817                 add_allocation_function("dma_alloc_attrs", &match_alloc, 1);
818                 add_allocation_function("pci_alloc_consistent", &match_alloc, 1)
819                 add_allocation_function("pci_alloc_coherent", &match_alloc, 1);
820                 add_allocation_function("devm_kmalloc", &match_alloc, 1);
821                 add_allocation_function("devm_kzalloc", &match_alloc, 1);
```

```
822                 add_allocation_function("kcalloc", &match_calloc, 0);
823                 add_allocation_function("devm_kcalloc", &match_calloc, 1);
824                 add_allocation_function("kmalloc_array", &match_calloc, 0);
825                 add_allocation_function("krealloc", &match_alloc, 1);

827                 add_function_hook("copy_from_user", &match_copy, NULL);
828                 add_function_hook("__copy_from_user", &match_copy, NULL);
829         }

831         add_hook(&array_check, OP_HOOK);
832         add_hook(&array_check_data_info, OP_HOOK);
833         add_hook(&set_used, OP_HOOK);

835         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);
836         add_hook(&munge_start_states, AFTER_DEF_HOOK);

838         add_hook(&match_assign, ASSIGNMENT_HOOK);

840         for (i = BYTE_COUNT; i <= USED_COUNT; i++) {
841                 select_call_implies_hook(i, &set_implied);
842                 select_caller_info_hook(set_param_compare, i);
843                 select_return_implies_hook(i, &set_implied);
844         }
845 }

847 void register_buf_comparison_links(int id)
848 {
849         link_id = id;
850         set_dynamic_states(link_id);
851         add_merge_hook(link_id, &merge_links);
852         add_modification_hook(link_id, &match_link_modify);
853 }
```

*unchanged_portion_omitted_*

```
**********************************************************
   21796 Mon Aug  5 08:38:20 2019
new/usr/src/tools/smatch/src/smatch_buf_size.c
11506 smatch resync
**********************************************************
```

_____unchanged_portion_omitted_

```
 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;
 333         if (expr->type != EXPR_PREOP || expr->op != '&')
 334                 return 0;
 335         type = get_type(expr);
 336         if (!type)
 337                 return 0;

 339         if (type->type == SYM_PTR)
 340                 type = get_base_type(type);

 342         ret = type_bytes(type);
 343         if (ret == 1)
 344                 return 0;  /* ignore char pointers */

 346         return ret;
 347 }
```

_____unchanged_portion_omitted_

```
 471 struct range_list *get_array_size_bytes_rl(struct expression *expr)
 472 {
 473         struct range_list *ret = NULL;
 474         int size;

 476         expr = remove_addr_fluff(expr);
 477         if (!expr)
 478                 return NULL;

 480         /* "BAR" */
 481         if (expr->type == EXPR_STRING)
 482                 return alloc_int_rl(expr->string->length);

 484         if (expr->type == EXPR_BINOP && expr->op == '+') {
 485                 sval_t offset;
 486                 struct symbol *type;
 487                 int bytes;

 489                 if (!get_implied_value(expr->right, &offset))
 490                         return NULL;
 491                 type = get_type(expr->left);
 492                 if (!type)
 493                         return NULL;
 494                 if (type->type != SYM_ARRAY && type->type != SYM_PTR)
 495                         return NULL;
 496                 type = get_real_base_type(type);
 497                 bytes = type_bytes(type);
 498                 if (bytes == 0)
 499                         return NULL;
 500                 offset.value *= bytes;
 501                 size = get_array_size_bytes(expr->left);
 502                 if (size <= 0)
 503                         return NULL;
 504                 return alloc_int_rl(size - offset.value);
 505         }
```



```
 507         size = get_stored_size_end_struct_bytes(expr);
 508         if (size)
 509                 return alloc_int_rl(size);

 511         /* buf[4] */
 512         size = get_real_array_size(expr);
 513         if (size)
 514                 return alloc_int_rl(elements_to_bytes(expr, size));

 516         /* buf = malloc(1024); */
 517         ret = get_stored_size_bytes(expr);
 518         if (ret)
 519                 return ret;

 519         size = get_stored_size_end_struct_bytes(expr);
 520         if (size)
 521                 return alloc_int_rl(size);

 521         /* char *foo = "BAR" */
 522         size = get_size_from_initializer(expr);
 523         if (size)
 524                 return alloc_int_rl(elements_to_bytes(expr, size));

 526         size = get_bytes_from_address(expr);
 527         if (size)
 528                 return alloc_int_rl(size);

 530         ret = size_from_db(expr);
 531         if (ret)
 532                 return ret;

 534         return NULL;
 535 }
```

_____unchanged_portion_omitted_

```
 709 static void match_calloc(const char *fn, struct expression *expr, void *unused)
 710 {
 711         struct expression *right;
 712         struct expression *size, *nr, *mult;
 713         struct range_list *rl;
 714         struct expression *arg;
 715         sval_t elements;
 716         sval_t size;

 715         right = strip_expr(expr->right);
 716         nr = get_argument_from_call_expr(right->args, 0);
 717         size = get_argument_from_call_expr(right->args, 1);
 718         mult = binop_expression(nr, '*', size);
 719         if (get_implied_rl(mult, &rl))
 720                 store_alloc(expr->left, rl);
 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))
 721         else
 722                 store_alloc(expr->left, size_to_rl(-1));
 723 }
```

_____unchanged_portion_omitted_

```
 868 void register_buf_size(int id)
 869 {
 870         my_size_id = id;
```

```
872         set_dynamic_states(my_size_id);

874         add_unmatched_state_hook(my_size_id, &unmatched_size_state);

876         select_caller_info_hook(set_param_buf_size, BUF_SIZE);
877         select_return_states_hook(BUF_SIZE, &db_returns_buf_size);
878         add_split_return_callback(print_returned_allocations);

880         allocation_funcs = create_function_hashtable(100);
881         add_allocation_function("malloc", &match_alloc, 0);
882         add_allocation_function("calloc", &match_calloc, 0);
883         add_allocation_function("memdup", &match_alloc, 1);
884         add_allocation_function("realloc", &match_alloc, 1);
885         if (option_project == PROJ_KERNEL) {
886                 add_allocation_function("kmalloc", &match_alloc, 0);
887                 add_allocation_function("kmalloc_node", &match_alloc, 0);
888                 add_allocation_function("kzalloc", &match_alloc, 0);
889                 add_allocation_function("kzalloc_node", &match_alloc, 0);
890                 add_allocation_function("vmalloc", &match_alloc, 0);
891                 add_allocation_function("__vmalloc", &match_alloc, 0);
892                 add_allocation_function("kcalloc", &match_calloc, 0);
893                 add_allocation_function("kmalloc_array", &match_calloc, 0);
894                 add_allocation_function("drm_malloc_ab", &match_calloc, 0);
895                 add_allocation_function("drm_calloc_large", &match_calloc, 0);
896                 add_allocation_function("sock_kmalloc", &match_alloc, 1);
897                 add_allocation_function("kmemdup", &match_alloc, 1);
898                 add_allocation_function("kmemdup_user", &match_alloc, 1);
899                 add_allocation_function("dma_alloc_attrs", &match_alloc, 1);
900                 add_allocation_function("pci_alloc_consistent", &match_alloc, 1)
901                 add_allocation_function("pci_alloc_coherent", &match_alloc, 1);
902                 add_allocation_function("devm_kmalloc", &match_alloc, 1);
903                 add_allocation_function("devm_kzalloc", &match_alloc, 1);
904                 add_allocation_function("krealloc", &match_alloc, 1);
905                 add_allocation_function("__alloc_bootmem", &match_alloc, 0);
906                 add_allocation_function("alloc_bootmem", &match_alloc, 0);
907                 add_allocation_function("kmap", &match_page, 0);
908                 add_allocation_function("get_zeroed_page", &match_page, 0);
909                 add_allocation_function("alloc_page", &match_page, 0);
910                 add_allocation_function("page_address", &match_page, 0);
911                 add_allocation_function("lowmem_page_address", &match_page, 0);
912                 add_allocation_function("alloc_pages", &match_alloc_pages, 1);
913                 add_allocation_function("alloc_pages_current", &match_alloc_page
914                 add_allocation_function("__get_free_pages", &match_alloc_pages,
915         }

917         add_allocation_function("strndup", match_strndup, 0);
918         if (option_project == PROJ_KERNEL)
919                 add_allocation_function("kstrndup", match_strndup, 0);

921         add_modification_hook(my_size_id, &set_size_undefined);

923         add_merge_hook(my_size_id, &merge_size_func);

925         if (option_info)
926                 add_hook(record_global_size, BASE_HOOK);
927 }
```

*_____unchanged_portion_omitted_*

```
**********************************************************
    8275 Mon Aug  5 08:38:21 2019
new/usr/src/tools/smatch/src/smatch_capped.c
11506 smatch resync
**********************************************************
______unchanged_portion_omitted_

 41 static struct smatch_state *unmatched_state(struct sm_state *sm)
 42 {
 43         struct smatch_state *state;

 45         state = __get_state(SMATCH_EXTRA, sm->name, sm->sym);
 45         state = get_state(SMATCH_EXTRA, sm->name, sm->sym);
 46         if (state && !estate_is_whole(state))
 47                 return &capped;
 48         return &uncapped;
 49 }
______unchanged_portion_omitted_

 69 int is_capped(struct expression *expr)
 70 {
 71         struct symbol *type;
 72         sval_t dummy;

 74         expr = strip_expr(expr);
 75         while (expr && expr->type == EXPR_POSTOP) {
 76                 expr = strip_expr(expr->unop);
 77         }
 78         if (!expr)
 79                 return 0;

 81         type = get_type(expr);
 82         if (is_ptr_type(type))
 83                 return 0;
 84         if (type == &bool_ctype)
 85                 return 0;
 86         if (type_bits(type) >= 0 && type_bits(type) <= 2)
 87                 return 0;

 89         if (get_hard_max(expr, &dummy))
 90                 return 1;

 92         if (is_capped_macro(expr))
 93                 return 1;

 95         if (expr->type == EXPR_BINOP) {
 96                 struct range_list *left_rl, *right_rl;

 98                 if (expr->op == '&')
 99                         return 1;
100                 if (expr->op == SPECIAL_RIGHTSHIFT)
101                         return 1;
102                 if (expr->op == '%' && is_capped(expr->right))
103                         return 1;
 93                 if (expr->op == '%')
 94                         return is_capped(expr->right);
104                 if (!is_capped(expr->left))
105                         return 0;
106                 if (expr->op == '/')
107                         return 1;
108                 if (!is_capped(expr->right))
109                         return 0;
110                 if (expr->op == '*') {
111                         get_absolute_rl(expr->left, &left_rl);
112                         get_absolute_rl(expr->right, &right_rl);
113                         if (sval_is_negative(rl_min(left_rl)) ||
```



```
114                             sval_is_negative(rl_min(right_rl)))
115                                 return 0;
116                 }
117                 return 1;
118         }
119         if (get_state_expr(my_id, expr) == &capped)
120                 return 1;
121         return 0;
122 }
______unchanged_portion_omitted_

141 static void match_condition(struct expression *expr)
142 {
143         struct expression *left, *right;
144         struct smatch_state *left_true = NULL;
145         struct smatch_state *left_false = NULL;
146         struct smatch_state *right_true = NULL;
147         struct smatch_state *right_false = NULL;
148         sval_t sval;


151         if (expr->type != EXPR_COMPARE)
152                 return;

154         left = strip_expr(expr->left);
155         right = strip_expr(expr->right);

157         while (left->type == EXPR_ASSIGNMENT)
158                 left = strip_expr(left->left);

160         /* If we're dealing with known expressions, that's for smatch_extra.c */
161         if (get_implied_value(left, &sval) ||
162             get_implied_value(right, &sval))
163                 return;

165         switch (expr->op) {
166         case '<':
167         case SPECIAL_LTE:
168         case SPECIAL_UNSIGNED_LT:
169         case SPECIAL_UNSIGNED_LTE:
170                 left_true = &capped;
171                 right_false = &capped;
172                 break;
173         case '>':
174         case SPECIAL_GTE:
175         case SPECIAL_UNSIGNED_GT:
176         case SPECIAL_UNSIGNED_GTE:
177                 left_false = &capped;
178                 right_true = &capped;
179                 break;
180         case SPECIAL_EQUAL:
181                 left_true = &capped;
182                 right_true = &capped;
183                 break;
184         case SPECIAL_NOTEQUAL:
185                 left_false = &capped;
186                 right_false = &capped;
187                 break;

189         default:
190                 return;
191         }

193         set_true_false_states_expr(my_id, left, left_true, left_false);
194         set_true_false_states_expr(my_id, right, right_true, right_false);
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);
195 }

197 static void match_assign(struct expression *expr)
198 {
199         struct symbol *type;

201         type = get_type(expr);
202         if (is_ptr_type(type))
203                 return;
204         if (type == &bool_ctype)
205                 return;
206         if (type_bits(type) >= 0 && type_bits(type) <= 2)
207                 return;

209         if (is_capped(expr->right)) {
210                 set_state_expr(my_id, expr->left, &capped);
211         } else {
212                 if (get_state_expr(my_id, expr->left))
213                         set_state_expr(my_id, expr->left, &uncapped);
214         }
215 }
```

*unchanged_portion_omitted_*

```
234 static void struct_member_callback(struct expression *call, int param, char *pri
235 {
236         struct smatch_state *estate;
237         sval_t sval;

239         if (sm->state != &capped)
240                 return;
241         estate = __get_state(SMATCH_EXTRA, sm->name, sm->sym);
209         estate = get_state(SMATCH_EXTRA, sm->name, sm->sym);
242         if (estate_get_single_value(estate, &sval))
243                 return;
244         sql_insert_caller_info(call, CAPPED_DATA, param, printed_name, "1");
245 }

247 static void print_return_implies_capped(int return_id, char *return_ranges, stru
248 {
249         struct smatch_state *orig, *estate;
250         struct sm_state *sm;
251         struct symbol *ret_sym;
252         const char *param_name;
253         char *return_str;
254         int param;
255         sval_t sval;
256         bool return_found = false;

258         expr = strip_expr(expr);
259         return_str = expr_to_str(expr);
260         ret_sym = expr_to_sym(expr);

262         FOR_EACH_MY_SM(my_id, __get_cur_stree(), sm) {
263                 if (sm->state != &capped)
264                         continue;

266                 param = get_param_num_from_sym(sm->sym);
267                 if (param < 0)
268                         continue;

270                 estate = __get_state(SMATCH_EXTRA, sm->name, sm->sym);
238                 estate = get_state(SMATCH_EXTRA, sm->name, sm->sym);
271                 if (estate_get_single_value(estate, &sval))
272                         continue;
```



```
274                 orig = get_state_stree(get_start_states(), my_id, sm->name, sm->
275                 if (orig == &capped && !param_was_set_var_sym(sm->name, sm->sym)
243                 if (orig == &capped)
276                         continue;

278                 param_name = get_param_name(sm);
279                 if (!param_name)
280                         continue;

282                 sql_insert_return_states(return_id, return_ranges, CAPPED_DATA,
283                                          param, param_name, "1");
284         } END_FOR_EACH_SM(sm);

286         FOR_EACH_MY_SM(my_id, __get_cur_stree(), sm) {
287                 if (!ret_sym)
288                         break;
289                 if (sm->state != &capped)
290                         continue;
291                 if (ret_sym != sm->sym)
292                         continue;

294                 estate = __get_state(SMATCH_EXTRA, sm->name, sm->sym);
295                 if (estate_get_single_value(estate, &sval))
296                         continue;

298                 param_name = state_name_to_param_name(sm->name, return_str);
299                 if (!param_name)
300                         continue;
301                 if (strcmp(param_name, "$") == 0)
302                         return_found = true;
303                 sql_insert_return_states(return_id, return_ranges, CAPPED_DATA,
304                                          -1, param_name, "1");
305         } END_FOR_EACH_SM(sm);

307         if (return_found)
308                 goto free_string;

310         if (option_project == PROJ_KERNEL && get_function() &&
311             strstr(get_function(), "nla_get_"))
312                 sql_insert_return_states(return_id, return_ranges, CAPPED_DATA,
313                                          -1, "$", "1");

315 free_string:
316         free_string(return_str);
317 }
```

*unchanged_portion_omitted_*

```
**********************************************************
   62964 Mon Aug  5 08:38:21 2019
new/usr/src/tools/smatch/src/smatch_comparison.c
11506 smatch resync
**********************************************************
_____unchanged_portion_omitted_
```

```
644 static void match_inc(struct sm_state *sm, bool preserve)
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                         if (preserve)
677                                 break;
678                         new = alloc_compare_state(
679                                         data->left, data->left_var, data->left_v
680                                         flip ? '<' : '>',
681                                         data->right, data->right_var, data->righ
682                         set_state(compare_id, tmp, NULL, new);
683                         break;
684                 case '<':
685                 case SPECIAL_UNSIGNED_LT:
686                         new = alloc_compare_state(
687                                         data->left, data->left_var, data->left_v
688                                         flip ? SPECIAL_GTE : SPECIAL_LTE,
689                                         data->right, data->right_var, data->righ
690                         set_state(compare_id, tmp, NULL, new);
691                         break;
692                 default:
693                         set_state(compare_id, tmp, NULL, &undefined);
694                 }
695         } END_FOR_EACH_PTR(tmp);
696 }

698 static void match_dec(struct sm_state *sm, bool preserve)
696 static void match_dec(struct sm_state *sm)
699 {
700         struct string_list *links;
```



```
701         struct smatch_state *state;
702         char *tmp;

704         links = sm->state->data;

706         FOR_EACH_PTR(links, tmp) {
707                 state = get_state(compare_id, tmp, NULL);

709                 switch (state_to_comparison(state)) {
710                 case SPECIAL_EQUAL:
711                 case SPECIAL_LTE:
712                 case SPECIAL_UNSIGNED_LTE:
713                 case '<':
714                 case SPECIAL_UNSIGNED_LT: {
715                         struct compare_data *data = state->data;
716                         struct smatch_state *new;

718                         if (preserve)
719                                 break;

721                         new = alloc_compare_state(
722                                         data->left, data->left_var, data->left_v
723                                         '<',
724                                         data->right, data->right_var, data->righ
725                         set_state(compare_id, tmp, NULL, new);
726                         break;
727                         }
728                 default:
729                         set_state(compare_id, tmp, NULL, &undefined);
730                 }
731         } END_FOR_EACH_PTR(tmp);
732 }

734 static void reset_sm(struct sm_state *sm)
735 {
736         struct string_list *links;
737         char *tmp;

739         links = sm->state->data;

741         FOR_EACH_PTR(links, tmp) {
742                 set_state(compare_id, tmp, NULL, &undefined);
743         } END_FOR_EACH_PTR(tmp);
744         set_state(link_id, sm->name, sm->sym, &undefined);
745 }

747 static bool match_add_sub_assign(struct sm_state *sm, struct expression *expr)
748 {
749         struct range_list *rl;
750         sval_t zero = { .type = &int_ctype };

752         if (!expr || expr->type != EXPR_ASSIGNMENT)
753                 return false;
754         if (expr->op != SPECIAL_ADD_ASSIGN && expr->op != SPECIAL_SUB_ASSIGN)
755                 return false;

757         get_absolute_rl(expr->right, &rl);
758         if (sval_is_negative(rl_min(rl))) {
759                 reset_sm(sm);
760                 return false;
761         }

763         if (expr->op == SPECIAL_ADD_ASSIGN)
764                 match_inc(sm, rl_has_sval(rl, zero));
765         else
766                 match_dec(sm, rl_has_sval(rl, zero));
```

```
 767         return true;
 768 }

 770 static void match_inc_dec(struct sm_state *sm, struct expression *mod_expr)
 771 {
 772         /*
 773          * if (foo > bar) then ++foo is also > bar.
 774          */
 775         if (!mod_expr)
 776                 return;
 777         if (match_add_sub_assign(sm, mod_expr))
 778                 return;
 779         if (mod_expr->type != EXPR_PREOP && mod_expr->type != EXPR_POSTOP)
 780                 return;

 782         if (mod_expr->op == SPECIAL_INCREMENT)
 783                 match_inc(sm, false);
 740                 match_inc(sm);
 784         else if (mod_expr->op == SPECIAL_DECREMENT)
 785                 match_dec(sm, false);
 742                 match_dec(sm);
 786 }
_____unchanged_portion_omitted_

 795 static void match_modify(struct sm_state *sm, struct expression *mod_expr)
 796 {
 754         struct string_list *links;
 755         char *tmp;

 797         if (mod_expr && is_self_assign(mod_expr))
 798                 return;

 800         /* handled by match_inc_dec() */
 801         if (mod_expr &&
 802             ((mod_expr->type == EXPR_PREOP || mod_expr->type == EXPR_POSTOP) &&
 803              (mod_expr->op == SPECIAL_INCREMENT || mod_expr->op == SPECIAL_DECRE
 804                 return;
 805         if (mod_expr && mod_expr->type == EXPR_ASSIGNMENT &&
 806             (mod_expr->op == SPECIAL_ADD_ASSIGN || mod_expr->op == SPECIAL_SUB_A
 807                 return;

 809         reset_sm(sm);
 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);
 810 }
_____unchanged_portion_omitted_

1645 static int get_comparison_helper(struct expression *a, struct expression *b, boo
1607 int get_comparison(struct expression *a, struct expression *b)
1646 {
1647         char *one = NULL;
1648         char *two = NULL;
1649         int ret = 0;

1651         if (!a || !b)
1652                 return 0;

1654         a = strip_parens(a);
1655         b = strip_parens(b);

1657         move_plus_to_minus(&a, &b);
```



```
1659         one = chunk_to_var(a);
1660         if (!one)
1661                 goto free;
1662         two = chunk_to_var(b);
1663         if (!two)
1664                 goto free;

1666         ret = get_comparison_strings(one, two);
1667         if (ret)
1668                 goto free;

1670         if (is_plus_one(a) || is_minus_one(a)) {
1671                 free_string(one);
1672                 one = chunk_to_var(a->left);
1673                 ret = get_comparison_strings(one, two);
1674         } else if (is_plus_one(b) || is_minus_one(b)) {
1675                 free_string(two);
1676                 two = chunk_to_var(b->left);
1677                 ret = get_comparison_strings(one, two);
1678         }

1680         if (!ret)
1681                 goto free;

1683         if ((is_plus_one(a) || is_minus_one(b)) && ret == '<')
1684                 ret = SPECIAL_LTE;
1685         else if ((is_minus_one(a) || is_plus_one(b)) && ret == '>')
1686                 ret = SPECIAL_GTE;
1687         else
1688                 ret = 0;

1690 free:
1691         free_string(one);
1692         free_string(two);

1694         if (!ret && use_extra)
1656         if (!ret)
1695                 return comparison_from_extra(a, b);
1696         return ret;
1697 }

1699 int get_comparison(struct expression *a, struct expression *b)
1700 {
1701         return get_comparison_helper(a, b, true);
1702 }

1704 int get_comparison_no_extra(struct expression *a, struct expression *b)
1705 {
1706         return get_comparison_helper(a, b, false);
1707 }

1709 int possible_comparison(struct expression *a, int comparison, struct expression
1710 {
1711         char *one = NULL;
1712         char *two = NULL;
1713         int ret = 0;
1714         char buf[256];
1715         struct sm_state *sm;
1716         int saved;

1718         one = chunk_to_var(a);
1719         if (!one)
1720                 goto free;
1721         two = chunk_to_var(b);
1722         if (!two)
1723                 goto free;
```

```
1726         if (strcmp(one, two) == 0 && comparison == SPECIAL_EQUAL) {
1727                 ret = 1;
1728                 goto free;
1729         }

1731         if (strcmp(one, two) > 0) {
1732                 char *tmp = one;

1734                 one = two;
1735                 two = tmp;
1736                 comparison = flip_comparison(comparison);
1737         }

1739         snprintf(buf, sizeof(buf), "%s vs %s", one, two);
1740         sm = get_sm_state(compare_id, buf, NULL);
1741         if (!sm)
1742                 goto free;

1744         FOR_EACH_PTR(sm->possible, sm) {
1745                 if (!sm->state->data)
1746                         continue;
1747                 saved = ((struct compare_data *)sm->state->data)->comparison;
1748                 if (saved == comparison)
1749                         ret = 1;
1750                 if (comparison == SPECIAL_EQUAL &&
1751                     (saved == SPECIAL_LTE ||
1752                      saved == SPECIAL_GTE ||
1753                      saved == SPECIAL_UNSIGNED_LTE ||
1754                      saved == SPECIAL_UNSIGNED_GTE))
1755                         ret = 1;
1756                 if (ret == 1)
1757                         goto free;
1758         } END_FOR_EACH_PTR(sm);

1760         return ret;
1761 free:
1762         free_string(one);
1763         free_string(two);
1764         return ret;
1765 }
```

_____***unchanged_portion_omitted_***

```
2370 static int split_op_param_key(char *value, int *op, int *param, char **key)
2371 {
2372         static char buf[256];
2373         char *p;

2375         if (!parse_comparison(&value, op))
2376                 return 0;

2378         snprintf(buf, sizeof(buf), "%s", value);
2330         snprintf(buf, sizeof(buf), value);

2380         p = buf;
2381         if (*p++ != '$')
2382                 return 0;

2384         *param = atoi(p);
2385         if (*param < 0 || *param > 99)
2386                 return 0;
2387         p++;
2388         if (*param > 9)
2389                 p++;
2390         p--;
```



```
2391         *p = '$';
2392         *key = p;

2394         return 1;
2395 }
```

_____***unchanged_portion_omitted_***

```
2543 void register_comparison(int id)
2544 {
2545         compare_id = id;
2546         set_dynamic_states(compare_id);
2547         add_hook(&save_start_states, AFTER_DEF_HOOK);
2548         add_unmatched_state_hook(compare_id, unmatched_comparison);
2549         add_pre_merge_hook(compare_id, &pre_merge_hook);
2550         add_merge_hook(compare_id, &merge_compare_states);
2551         add_hook(&free_data, AFTER_FUNC_HOOK);
2552         add_hook(&match_call_info, FUNCTION_CALL_HOOK);
2553         add_split_return_callback(&print_return_comparison);

2555         select_return_states_hook(PARAM_COMPARE, &db_return_comparison);
2556         add_hook(&match_preop, OP_HOOK);
2557 }
```

_____***unchanged_portion_omitted_***

```
2564 void register_comparison_links(int id)
2565 {
2566         link_id = id;
2567         db_ignore_states(link_id);
2568         set_dynamic_states(link_id);
2569         add_merge_hook(link_id, &merge_links);
2570         add_modification_hook(link_id, &match_modify);
2571         add_modification_hook_late(link_id, match_inc_dec);

2573         add_member_info_callback(link_id, struct_member_callback);
2574 }
```

_____***unchanged_portion_omitted_***

```
2582 void register_comparison_inc_dec_links(int id)
2583 {
2584         inc_dec_link_id = id;
2585         set_dynamic_states(inc_dec_link_id);
2586         set_up_link_functions(inc_dec_id, inc_dec_link_id);
2587 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   19182 Mon Aug  5 08:38:21 2019
new/usr/src/tools/smatch/src/smatch_conditions.c
11506 smatch resync
**********************************************************
```

_____unchanged_portion_omitted_

```
 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                 expr_set_parent_expr(expr->left, expr);
 424                 expr_set_parent_expr(expr->right, expr);
 425                 handle_logical(expr);
 426                 return;
 427         }

 429         switch (expr->type) {
 430         case EXPR_LOGICAL:
 431                 expr_set_parent_expr(expr->left, expr);
 432                 expr_set_parent_expr(expr->right, expr);
 433                 __pass_to_client(expr, LOGIC_HOOK);
 434                 handle_logical(expr);
 435                 return;
 436         case EXPR_COMPARE:
 437                 expr_set_parent_expr(expr->left, expr);
 438                 expr_set_parent_expr(expr->right, expr);
 439                 hackup_unsigned_compares(expr);
 440                 if (handle_zero_comparisons(expr))
 441                         return;
 442                 break;
 443         case EXPR_CALL:
 444                 if (ignore_builtin_expect(expr))
 445                         return;
 446                 break;
 447         case EXPR_PREOP:
 448                 expr_set_parent_expr(expr->unop, expr);
 449                 if (handle_preop(expr))
 450                         return;
 451                 break;
 452         case EXPR_CONDITIONAL:
 453         case EXPR_SELECT:
```



```
 454                 expr_set_parent_expr(expr->conditional, expr);
 455                 expr_set_parent_expr(expr->cond_true, expr);
 456                 expr_set_parent_expr(expr->cond_false, expr);
 457                 handle_select(expr);
 458                 return;
 459         case EXPR_COMMA:
 460                 expr_set_parent_expr(expr->left, expr);
 461                 expr_set_parent_expr(expr->right, expr);
 462                 handle_comma(expr);
 463                 return;
 464         }

 466         /* fixme: this should be in smatch_flow.c
 467            but because of the funny stuff we do with conditions
 468            it's awkward to put it there.  We would need to
 469            call CONDITION_HOOK in smatch_flow as well.
 470         */
 471         push_expression(&big_expression_stack, expr);
 472         push_expression(&big_condition_stack, expr);

 474         if (expr->type == EXPR_COMPARE) {
 475                 if (expr->left->type != EXPR_POSTOP)
 476                         __split_expr(expr->left);
 477                 if (expr->right->type != EXPR_POSTOP)
 478                         __split_expr(expr->right);
 479         } else if (expr->type != EXPR_POSTOP) {
 480                 __split_expr(expr);
 481         }
 482         do_condition(expr);
 483         if (expr->type == EXPR_COMPARE) {
 484                 if (expr->left->type == EXPR_POSTOP)
 485                         __split_expr(expr->left);
 486                 if (expr->right->type == EXPR_POSTOP)
 487                         __split_expr(expr->right);
 488         } else if (expr->type == EXPR_POSTOP) {
 489                 __split_expr(expr);
 490         }
 491         __push_fake_cur_stree();
 492         __process_post_op_stack();
 493         __fold_in_set_states();
 494         pop_expression(&big_condition_stack);
 495         pop_expression(&big_expression_stack);
 496 }
```

_____unchanged_portion_omitted_

```
**********************************************************
   12703 Mon Aug  5 08:38:22 2019
new/usr/src/tools/smatch/src/smatch_constraints.c
11506 smatch resync
**********************************************************
```

```
______unchanged_portion_omitted_

 195 char *get_constraint_str(struct expression *expr)
 196 {
 197         char *name;

 199         expr = strip_expr(expr);
 200         if (!expr)
 201                 return NULL;
 202         if (expr->type == EXPR_CALL)
 203                 return get_func_constraint(expr);
 204         if (expr->type == EXPR_BINOP)
 205                 return expr_to_str(expr);
 206         name = get_toplevel_name(expr);
 207         if (name)
 208                 return name;
 209         return get_member_name(expr);
 210 }
______unchanged_portion_omitted_

 342 struct string_list *saved_constraints;
 343 static void save_new_constraint(const char *con)
 344 {
 345         if (!insert_string(&saved_constraints, con))
 344         if (list_has_string(saved_constraints, con))
 346                 return;
 346         insert_string(&saved_constraints, con);
 347         sql_save_constraint(con);
 348 }

 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;

 359         /* known values are handled in smatch extra */
 360         if (get_value(left, &sval) || get_value(right, &sval))
 361                 return;

 363         if (local_debug)
 364                 sm_msg("COMPARE: %s %s %s", expr_to_str(left), show_special(op),

 363         constraint = get_constraint_str(right);
 364         if (!constraint)
 365                 return;
 369         if (local_debug)
 370                 sm_msg("EXPR: %s CONSTRAINT %s", expr_to_str(right), constraint)
 366         constraint_id = constraint_str_to_id(constraint);
 372         if (local_debug)
 373                 sm_msg("CONSTRAINT ID %d", constraint_id);
 367         if (constraint_id < 0)
 368                 save_new_constraint(constraint);
 369         free_string(constraint);
 370         if (constraint_id < 0)
 371                 return;

 373         constraints = get_constraints(left);
```



```
 374         constraints = clone_constraint_list(constraints);
 375         op = negate_gt(orig_op);
 376         add_constraint(&constraints, remove_unsigned_from_comparison(op), constr
 377         state = alloc_constraint_state(constraints);

 379         if (op == orig_op)
 386         if (op == orig_op) {
 387                 if (local_debug)
 388                         sm_msg("SETTING %s true %s", expr_to_str(left), state->n
 380                 set_true_false_states_expr(my_id, left, state, NULL);
 381         else
 390         } else {
 391                 if (local_debug)
 392                         sm_msg("SETTING %s false %s", expr_to_str(left), state->

 382                 set_true_false_states_expr(my_id, left, NULL, state);
 395         }
 383 }
______unchanged_portion_omitted_

 515 void register_constraints(int id)
 516 {
 517         my_id = id;

 519         set_dynamic_states(my_id);
 520         add_merge_hook(my_id, &merge_func);
 521         add_hook(&match_condition, CONDITION_HOOK);

 523         add_hook(&match_caller_info, FUNCTION_CALL_HOOK);
 524         add_member_info_callback(my_id, struct_member_callback);
 525         select_caller_info_hook(&set_param_constrained, CONSTRAINT);

 527         add_split_return_callback(print_return_implies_constrained);
 528         select_return_states_hook(CONSTRAINT, &db_returns_constrained);
 529 }
______unchanged_portion_omitted_
```

****
12381 Mon Aug 5 08:38:22 2019
new/usr/src/tools/smatch/src/smatch_constraints_required.c
11506 smatch resync
****
_____***unchanged_portion_omitted_***

```
 281 static void match_assign_has_buf_comparison(struct expression *expr)
 282 {
 283         struct expression *size;
 284         int limit_type;

 286         if (expr->op != '=')
 287                 return;
 288         if (expr->right->type == EXPR_CALL)
 289                 return;
 290         size = get_size_variable(expr->right, &limit_type);
 289         size = get_size_variable(expr->right);
 291         if (!size)
 292                 return;
 293         if (limit_type != ELEM_COUNT)
 294                 return;
 295         match_alloc_helper(expr->left, size, 1);
 296 }
```

_____***unchanged_portion_omitted_***

```
 457 void register_constraints_required(int id)
 458 {
 459         my_id = id;

 461         set_dynamic_states(my_id);
 462         add_hook(&match_assign_size, ASSIGNMENT_HOOK);
 463         add_hook(&match_assign_data, ASSIGNMENT_HOOK);
 464         add_hook(&match_assign_has_buf_comparison, ASSIGNMENT_HOOK);

 466         add_hook(&match_assign_ARRAY_SIZE, ASSIGNMENT_HOOK);
 467         add_hook(&match_assign_ARRAY_SIZE, GLOBAL_ASSIGNMENT_HOOK);
 468         add_hook(&match_assign_buf_comparison, ASSIGNMENT_HOOK);
 469         add_hook(&match_assign_constraint, ASSIGNMENT_HOOK);

 471         add_allocation_function("malloc", &match_alloc, 0);
 472         add_allocation_function("memdup", &match_alloc, 1);
 473         add_allocation_function("realloc", &match_alloc, 1);
 474         add_allocation_function("realloc", &match_calloc, 0);
 475         if (option_project == PROJ_KERNEL) {
 476                 add_allocation_function("kmalloc", &match_alloc, 0);
 477                 add_allocation_function("kzalloc", &match_alloc, 0);
 478                 add_allocation_function("vmalloc", &match_alloc, 0);
 479                 add_allocation_function("__vmalloc", &match_alloc, 0);
 480                 add_allocation_function("vzalloc", &match_alloc, 0);
 481                 add_allocation_function("sock_kmalloc", &match_alloc, 1);
 482                 add_allocation_function("kmemdup", &match_alloc, 1);
 483                 add_allocation_function("kmemdup_user", &match_alloc, 1);
 484                 add_allocation_function("dma_alloc_attrs", &match_alloc, 1);
 485                 add_allocation_function("pci_alloc_consistent", &match_alloc, 1)
 486                 add_allocation_function("pci_alloc_coherent", &match_alloc, 1);
 487                 add_allocation_function("devm_kmalloc", &match_alloc, 1);
 488                 add_allocation_function("devm_kzalloc", &match_alloc, 1);
 489                 add_allocation_function("kcalloc", &match_calloc, 0);
 490                 add_allocation_function("kmalloc_array", &match_calloc, 0);
 491                 add_allocation_function("devm_kcalloc", &match_calloc, 1);
 492                 add_allocation_function("krealloc", &match_alloc, 1);
 493         }
 494 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   15722 Mon Aug  5 08:38:22 2019
new/usr/src/tools/smatch/src/smatch_container_of.c
11506 smatch resync
**********************************************************
______unchanged_portion_omitted_

 115 static char *get_container_name_sm(struct sm_state *sm, int offset)
 115 static char *get_container_name(struct sm_state *sm, int offset)
 116 {
 117         static char buf[256];
 118         const char *name;

 120         name = get_param_name(sm);
 121         if (!name)
 122                 return NULL;

 124         if (name[0] == '$')
 125                 snprintf(buf, sizeof(buf), "$(-%d)%s", offset, name + 1);
 126         else if (name[0] == '*' || name[1] == '$')
 127                 snprintf(buf, sizeof(buf), "*$(-%d)%s", offset, name + 2);
 128         else
 129                 return NULL;

 131         return buf;
 132 }
______unchanged_portion_omitted_

 165 static void process_states(void)
 166 {
 167         struct sm_state *tmp;
 168         int arg, offset;
 169         const char *name;

 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_sm(tmp, offset);
 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);

 182         free_stree(&used_stree);
 183 }
______unchanged_portion_omitted_

 252 static int get_deref_count(struct expression *expr)
 252 static int get_shared_cnt(const char *one, const char *two)
 253 {
 254         int cnt = 0;
 254         int i;
 255         int on_end = false;

 256         while (expr && expr->type == EXPR_DEREF) {
 257                 expr = expr->deref;
 258                 if (expr->type == EXPR_PREOP && expr->op == '*')
 259                         expr = expr->unop;
 260                 cnt++;
 261                 if (cnt > 100)
 262                         return -1;
 257         i = 0;
 258         while (true) {
```



```
 259                 if (!one[i] || !two[i]) {
 260                         on_end = true;
 261                         break;
 263         }
 264         return cnt;
 265 }

 267 static struct expression *get_partial_deref(struct expression *expr, int cnt)
 268 {
 269         while (--cnt >= 0) {
 270                 if (!expr || expr->type != EXPR_DEREF)
 271                         return expr;
 272                 expr = expr->deref;
 273                 if (expr->type == EXPR_PREOP && expr->op == '*')
 274                         expr = expr->unop;
 263                 if (one[i] != two[i])
 264                         break;
 265                 i++;
 275         }
 276         return expr;
 277 }

 279 static int partial_deref_to_offset_str(struct expression *expr, int cnt, char op
 280 {
 281         int n, offset;

 283         if (cnt == 0)
 284                 return snprintf(buf, size, "%c0", op);

 286         n = 0;
 287         while (--cnt >= 0) {
 288                 offset = get_member_offset_from_deref(expr);
 289                 if (offset < 0)
 290                         return -1;
 291                 n += snprintf(buf + n, size - n, "%c%d", op, offset);
 292                 if (expr->type != EXPR_DEREF)
 293                         return -1;
 294                 expr = expr->deref;
 295                 if (expr->type == EXPR_PREOP && expr->op == '*')
 296                         expr = expr->unop;
 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--;
 297         }
 274         if (!on_end)
 275                 return 0;

 299         return n;
 277         return i + 1;
 300 }

 302 static char *get_shared_str(struct expression *container, struct expression *exp
 280 static int build_offset_str(struct expression *expr, const char *name,
 281                             int shared, char *buf, int size, int op)
 303 {
 304         struct expression *one, *two;
 305         int cont, exp, min, ret, n;
 306         static char buf[48];
 283         int chop = 0;
 284         int offset;
 285         int i;

 308         cont = get_deref_count(container);
```

```
309         exp = get_deref_count(expr);
310         if (cont < 0 || exp < 0)
311                 return NULL;

313         min = (cont < exp) ? cont : exp;
314         while (min >= 0) {
315                 one = get_partial_deref(container, cont - min);
316                 two = get_partial_deref(expr, exp - min);
317                 if (expr_equiv(one, two))
318                         goto found;
319                 min--;
287         i = shared;
288         while (name[i]) {
289                 if (name[i] == '.' || name[i] == '-')
290                         chop++;
291                 i++;
320         }

322         return NULL;
294         // FIXME:  Handle more chops
295         if (chop > 1)
296                 return 0;

324 found:
325         ret = partial_deref_to_offset_str(container, cont - min, '-', buf, sizeo
326         if (ret < 0)
327                 return NULL;
328         n = ret;
329         ret = partial_deref_to_offset_str(expr, exp - min, '+', buf + ret, sizeo
330         if (ret < 0)
331                 return NULL;
332         n += ret;
333         if (n >= sizeof(buf))
334                 return NULL;

336         return buf;
337 }

339 char *get_container_name(struct expression *container, struct expression *expr)
340 {
341         struct symbol *container_sym, *sym;
342         struct expression *tmp;
343         static char buf[64];
344         char *shared;
345         bool star;
346         int cnt;

348         container_sym = expr_to_sym(container);
349         sym = expr_to_sym(expr);
350         if (container_sym && container_sym == sym)
351                 goto found;

353         cnt = 0;
354         while ((tmp = get_assigned_expr(expr))) {
355                 expr = tmp;
356                 if (cnt++ > 3)
357                         break;
298         if (chop == 0) {
299                 offset = 0;
300         } else {
301                 offset = get_member_offset_from_deref(expr);
302                 if (offset < 0)
303                         return 0;
358         }

360         cnt = 0;
```



```
361         while ((tmp = get_assigned_expr(container))) {
362                 container = tmp;
363                 if (cnt++ > 3)
364                         break;
365         }

367 found:
368         expr = strip_expr(expr);
369         star = true;
370         if (expr->type == EXPR_PREOP && expr->op == '&') {
371                 expr = strip_expr(expr->unop);
372                 star = false;
373         }

375         container_sym = expr_to_sym(container);
376         if (!container_sym)
377                 return NULL;
378         sym = expr_to_sym(expr);
379         if (!sym || container_sym != sym)
380                 return NULL;

382         shared = get_shared_str(container, expr);
383         if (star)
384                 snprintf(buf, sizeof(buf), "*(%s)", shared);
385         else
386                 snprintf(buf, sizeof(buf), "%s", shared);

388         return buf;
306         snprintf(buf, size, "%c%d", (op == '+') ? '+' : '-', offset);
307         return 1;
389 }

391 static void match_call(struct expression *call)
392 {
393         struct expression *fn, *arg;
394         char *name;
395         int param;
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;

397         /*
398          * We're trying to link the function with the parameter.  There are a
399          * couple ways this can be passed:
400          * foo->func(foo, ...);
401          * foo->func(foo->x, ...);
402          * foo->bar.func(&foo->bar, ...);
403          * foo->bar->baz->func(foo, ...);
404          *
405          * So the method is basically to subtract the offsets until we get to
406          * the common bit, then add the member offsets to get the parameter.
407          *
408          * If we're taking an address then the offset math is not stared,
409          * otherwise it is.  Starred means dereferenced.
410          */
411         fn = strip_expr(call->fn);
335         fn_name = expr_to_var(fn);
336         if (!fn_name)
337                 return;

413         param = -1;
414         FOR_EACH_PTR(call->args, arg) {
415                 param++;
```

```
417                 name = get_container_name(fn, arg);
418                 if (!name)
419                         continue;
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                 }

421                 sql_insert_caller_info(call, CONTAINER, param, name, "$(-1)");
350                 arg_name = expr_to_var(arg);
351                 if (!arg_name)
352                         continue;
353                 shared = get_shared_cnt(fn_name, arg_name);
354                 if (!shared)
355                         goto free_arg_name;
356                 if (!build_offset_str(fn, fn_name, shared, minus_str, sizeof(min
357                         goto free_arg_name;
358                 if (!build_offset_str(arg, arg_name, shared, plus_str, sizeof(pl
359                         goto free_arg_name;
360                 if (star)
361                         snprintf(offset_str, sizeof(offset_str), "*(%s%s)", minu
362                 else
363                         snprintf(offset_str, sizeof(offset_str), "%s%s", minus_s
364                 sql_insert_caller_info(call, CONTAINER, param, offset_str, "$(-1
365 free_arg_name:
366                 free_string(arg_name);
422         } END_FOR_EACH_PTR(arg);

369         free_string(fn_name);
423 }

425 static void db_passed_container(const char *name, struct symbol *sym, char *key,
426 {
427         set_state(param_id, name, sym, alloc_state_str(key));
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));
428 }
```

_____***unchanged_portion_omitted_***


```
604 static void load_container_data(struct symbol *arg, const char *info)
577 static void handle_passed_container(struct symbol *sym)
605 {
606         mtag_t cur_tag, container_tag, arg_tag;
607         int container_offset, arg_offset;
608         char *p = (char *)info;
579         struct symbol *arg;
580         struct smatch_state *state;
609         struct sm_state *sm;
610         struct stree *stree;
611         bool star = 0;
583         mtag_t fn_tag, container_tag, arg_tag;
584         sval_t offset;
585         int container_offset, arg_offset;
586         int star;

613         if (p[0] == '*') {
614                 star = 1;
615                 p += 2;
616         }
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);

618         if (!get_toplevel_mtag(cur_func_sym, &cur_tag))
619                 return;

621         while (true) {
622                 container_offset = strtoul(p, &p, 0);
623                 if (local_debug)
624                         sm_msg("%s: cur_tag = %llu container_offset = %d",
625                                __func__, cur_tag, container_offset);
626                 if (!mtag_map_select_container(cur_tag, container_offset, &conta
595 found:
596         if (!estate_get_single_value(state, &offset))
627                         return;
628                 cur_tag = container_tag;
629                 if (local_debug)
630                         sm_msg("%s: container_tag = %llu p = '%s'",
631                                __func__, container_tag, p);
632                 if (!p)
633                         return;
634                 if (p[0] != '-')
635                         break;
636                 p++;
637         }
598         container_offset = -(offset.value & 0xffff);
599         arg_offset = (offset.value & 0xfff0000) >> 16;
600         star = !!(offset.value & (1ULL << 31));

639         if (p[0] != '+')
602         if (!get_toplevel_mtag(cur_func_sym, &fn_tag))
640                 return;

642         p++;
643         arg_offset = strtoul(p, &p, 0);
644         if (p && *p && *p != ')')
604         if (!mtag_map_select_container(fn_tag, container_offset, &container_tag)
645                 return;

647         if (!arg_offset || star) {
648                 arg_tag = container_tag;
649         } else {
```

```
650                 if (!mtag_map_select_tag(container_tag, -arg_offset, &arg_tag))
651                         return;
652         }

654         stree = load_tag_info_sym(arg_tag, arg, arg_offset, star);
655         FOR_EACH_SM(stree, sm) {
656                 set_state(sm->owner, sm->name, sm->sym, sm->state);
657         } END_FOR_EACH_SM(sm);
658         free_stree(&stree);
659 }

661 static void handle_passed_container(struct symbol *sym)
662 {
663         struct symbol *arg;
664         struct smatch_state *state;

666         FOR_EACH_PTR(cur_func_sym->ctype.base_type->arguments, arg) {
667                 state = get_state(param_id, arg->ident->name, arg);
668                 if (!state || state == &merged)
669                         continue;
670                 load_container_data(arg, state->name);
671         } END_FOR_EACH_PTR(arg);
672 }

674 void register_container_of(int id)
675 {
676         my_id = id;

678         add_hook(&match_function_def, FUNC_DEF_HOOK);

680         add_get_state_hook(&get_state_hook);

682         add_hook(&match_save_states, INLINE_FN_START);
683         add_hook(&match_restore_states, INLINE_FN_END);

685         select_return_implies_hook(CONTAINER, &set_param_used);
686         all_return_states_hook(&process_states);

688         add_split_return_callback(&print_returns_container_of);
689         select_return_states_hook(CONTAINER, &returns_container_of);

691         add_hook(&match_call, FUNCTION_CALL_HOOK);
692 }

640 static struct smatch_state *unmatched_state(struct sm_state *sm)
641 {
642         return alloc_estate_whole(estate_type(sm->state));
643 }

694 void register_container_of2(int id)
695 {
696         param_id = id;

698         set_dynamic_states(param_id);
699         select_caller_info_hook(db_passed_container, CONTAINER);
700         add_merge_hook(param_id, &merge_str_state);
701         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);
702 }
```

_____*unchanged_portion_omitted_*

****
456 Mon Aug 5 08:38:23 2019
new/usr/src/tools/smatch/src/smatch_data/db/apply_return_fixes.sh
11506 smatch resync
****

```
#!/bin/bash

if echo $1 | grep -q '^-p' ; then
    PROJ=$(echo $1 | cut -d = -f 2)
    shift
fi

bin_dir=$(dirname $0)
db_file=$1
if [ "$db_file" == "" ] ; then
    echo "usage: $0 -p=<project> <db_file>"
    exit
fi

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
```

****
1645 Mon Aug 5 08:38:23 2019
new/usr/src/tools/smatch/src/smatch_data/db/create_db.sh
11506 smatch resync
****

```
 1 #!/bin/bash

 3 if echo $1 | grep -q '^-p' ; then
 4     PROJ=$(echo $1 | cut -d = -f 2)
 5     shift
 6 fi

 8 info_file=$1

10 if [[ "$info_file" = "" ]] ; then
11     echo "Usage:  $0 -p=<project> <file with smatch messages>"
12     exit 1
13 fi

15 bin_dir=$(dirname $0)
16 db_file=smatch_db.sqlite.new

18 rm -f $db_file

20 for i in ${bin_dir}/*.schema ; do
21     cat $i | sqlite3 $db_file
22 done

24 ${bin_dir}/init_constraints.pl "$PROJ" $info_file $db_file
25 ${bin_dir}/init_constraints_required.pl "$PROJ" $info_file $db_file
26 ${bin_dir}/fill_db_sql.pl "$PROJ" $info_file $db_file
27 if [ -e ${info_file}.sql ] ; then
28     ${bin_dir}/fill_db_sql.pl "$PROJ" ${info_file}.sql $db_file
29 fi
30 ${bin_dir}/fill_db_caller_info.pl "$PROJ" $info_file $db_file
31 if [ -e ${info_file}.caller_info ] ; then
32     ${bin_dir}/fill_db_caller_info.pl "$PROJ" ${info_file}.caller_info $db_file
33 fi
34 ${bin_dir}/build_early_index.sh $db_file

36 ${bin_dir}/fill_db_type_value.pl "$PROJ" $info_file $db_file
37 ${bin_dir}/fill_db_type_size.pl "$PROJ" $info_file $db_file
38 ${bin_dir}/copy_required_constraints.pl "$PROJ" $info_file $db_file
39 ${bin_dir}/build_late_index.sh $db_file

41 ${bin_dir}/fixup_all.sh $db_file
42 if [ "$PROJ" != "" ] ; then
43     ${bin_dir}/fixup_${PROJ}.sh $db_file
44 fi

46 ${bin_dir}/remove_mixed_up_pointer_params.pl $db_file
47 ${bin_dir}/delete_too_common_fn_ptr.sh $db_file
48 ${bin_dir}/mark_function_ptrs_searchable.pl $db_file

50 # delete duplicate entrees and speed things up
51 echo "delete from function_ptr where rowid not in (select min(rowid) from functi

53 ${bin_dir}/apply_return_fixes.sh -p=${PROJ} $db_file
52 test -e  ${bin_dir}/${PROJ}.return_fixes && \
53 cat ${bin_dir}/${PROJ}.return_fixes | \
54 while read func old new ; do
55     echo "update return_states set return = '$new' where function = '$func' and
56 done

55 mv $db_file smatch_db.sqlite
```

****
331 Mon Aug 5 08:38:23 2019
new/usr/src/tools/smatch/src/smatch_data/db/delete_too_common_fn_ptr.sh
11506 smatch resync
****

```
#!/bin/bash

db_file=$1

IFS="|"
echo "select count(function), function from function_ptr group by function;" | \
    sqlite3 $db_file  | sort -n | tail -n 100 | \

while read cnt func ; do
    if [ $cnt -lt 200 ] ; then
        continue
    fi
    echo "delete from function_ptr where function = '$func';" | sqlite3 $db_file
done
```

```
**********************************************************
   13929 Mon Aug  5 08:38:23 2019
new/usr/src/tools/smatch/src/smatch_data/db/fixup_kernel.sh
11506 smatch resync
**********************************************************
```

```
#!/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

/* 8017 is USER_DATA and  9017 is USER_DATA_SET */
delete from caller_info where function = 'dev_hard_start_xmit' and type = 8017;
/* 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;
delete from caller_info where function = '(struct packet_type)->func' and type =
delete from caller_info where function = '(struct bio)->bi_end_io' and type = 80
delete from caller_info where caller = 'NF_HOOK_COND' and type = 8017;
delete from caller_info where caller = 'NF_HOOK' 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, 8017,
insert into caller_info values ('userspace', '', 'compat_sys_ioctl', 0, 0, 8017,
insert into caller_info values ('userspace', '', 'compat_sys_ioctl', 0, 0, 8017,
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, 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, '4096-ptr_max', 0,
insert into return_states values ('faked', 'vmalloc', 0, 1, '4096-ptr_max', 0, 1
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,
```

```
 89 /* store a bunch of capped functions */
 90 update return_states set return = '0-u32max[<=\$2]' where function = 'copy_to_us
 91 update return_states set return = '0-u32max[<=\$2]' where function = '_copy_to_u
 92 update return_states set return = '0-u32max[<=\$2]' where function = '__copy_to_
 93 update return_states set return = '0-u32max[<=\$2]' where function = 'copy_from_
 94 update return_states set return = '0-u32max[<=\$2]' where function = '_copy_from
 95 update return_states set return = '0-u32max[<=\$2]' where function = '__copy_fro

 97 update return_states set return = '0-8' where function = '__arch_hweight8';
 98 update return_states set return = '0-16' where function = '__arch_hweight16';
 99 update return_states set return = '0-32' where function = '__arch_hweight32';
100 update return_states set return = '0-64' where function = '__arch_hweight64';

102 /*
103  * Preserve the value across byte swapping.  By the time we use it for math it
104  * will be byte swapped back to CPU endian.
105  */
106 update return_states set return = '0-u64max[==\$0]' where function = '__fswab64'
107 update return_states set return = '0-u32max[==\$0]' where function = '__fswab32'
108 update return_states set return = '0-u16max[==\$0]' where function = '__fswab16'
109 update return_states set return = '0-u64max[==\$0]' where function = '__builtin_
110 update return_states set return = '0-u32max[==\$0]' where function = '__builtin_
111 update return_states set return = '0-u16max[==\$0]' where function = '__builtin_

113 delete from return_states where function = 'bitmap_allocate_region' and return =
114 /* Just delete a lot of returns that everyone ignores */
115 delete from return_states where file = 'drivers/pci/access.c' and (return >= 129

144 update return_states set return = '(-4095)-s32max[<=\$1]' where function = 'get_
145 update return_states set return = '(-4095)-s64max[<=\$1]' where function = 'get_

117 /* Smatch can't parse wait_for_completion() */
118 update return_states set return = '(-108),(-22),0' where function = '__spi_sync'

120 delete from caller_info where caller = '__kernel_write';

122 /* We sometimes use pre-allocated 4097 byte buffers for performance critical cod
123 update caller_info set value = 4096 where caller='kernfs_file_direct_read' and f
124 /* let's pretend firewire doesn't exist */
125 delete from caller_info where caller='init_fw_attribute_group' and function='(st
126 /* and let's fake the next dev_attr_show() call entirely */
127 delete from caller_info where caller='sysfs_kf_seq_show' and function='(struct s
128 insert into caller_info values ('fake', 'sysfs_kf_seq_show', '(struct sysfs_ops)
158 insert into caller_info values ('fake', 'sysfs_kf_seq_show', '(struct sysfs_ops)
129 insert into caller_info values ('fake', 'sysfs_kf_seq_show', '(struct sysfs_ops)
130 insert into caller_info values ('fake', 'sysfs_kf_seq_show', '(struct sysfs_ops)
160 insert into caller_info values ('fake', 'sysfs_kf_seq_show', '(struct sysfs_ops)
131 insert into caller_info values ('fake', 'sysfs_kf_seq_show', '(struct sysfs_ops)
132 /* config fs confuses smatch a little */
133 update caller_info set value = 4096 where caller='fill_read_buffer' and function

135 /* smatch sees the memset() but not the subsequent changes */
136 update return_states set value = "" where function = 'gfs2_ea_find' and return =

138 delete from type_value where type = '(struct fd)->file';
139 delete from type_value where type = '(struct fd)->flags';

141 /* This is sometimes an enum or a u64 */
142 delete from type_value where type = '(struct mc_cmd_header)->status';

144 /* this is handled in check_kernel.c */
145 delete from return_states where function = "__write_once_size";

147 update return_states set value = "s32min-s32max[\$1]" where function = 'atomic_s

149 /* handled in the check itself */
```



```
150 delete from return_states where function = 'atomic_inc_return' and (type = 8023
151 delete from return_states where function = 'atomic_add_return' and (type = 8023
152 delete from return_states where function = 'atomic_sub_return' and (type = 8023
153 delete from return_states where function = 'atomic_sub_and_test' and (type = 802
154 delete from return_states where function = 'atomic_dec_and_test' and (type = 802
155 delete from return_states where function = 'atomic_dec' and (type = 8023 or type
156 delete from return_states where function = 'atomic_inc' and (type = 8023 or type
157 delete from return_states where function = 'atomic_sub' and (type = 8023 or type
158 delete from return_states where function = 'refcount_add_not_zero' and (type = 8
159 delete from return_states where function = 'refcount_inc_not_zero' and (type = 8
160 delete from return_states where function = 'refcount_sub_and_test' and (type = 8

162 update return_states set return = '0-32,2147483648-2147483690' where function =
163 update return_states set value = '0-u64max' where function = '_parse_integer' an

165 /* delete some function pointers which are sometimes byte units */
166 delete from caller_info where function = '(struct i2c_algorithm)->master_xfer' a

168 /* this if from READ_ONCE().  We can't know anything about the data.  */
169 delete from type_info where key = '(union anonymous)->__val';

171 /* This is RIO_BAD_SIZE */
172 delete from return_states where file = 'drivers/rapidio/rio-access.c' and return

174 /* Smatch sucks at loops */
175 delete from return_states where function = 'ata_dev_next' and type = 103;

177 EOF

179 # fixme: this is totally broken
180 call_id=$(echo "select distinct call_id from caller_info where function = '__ker
181 for id in $call_id ; do
182     echo "insert into caller_info values ('fake', '', '__kernel_write', $id, 0,
206     echo "insert into caller_info values ('fake', '', '__kernel_write', $id, 0,
183 done

185 for i in $(echo "select distinct return from return_states where function = 'cle
186     echo "update return_states set return = \"$i[<=\$1]\" where return = \"$i\"
187 done

189 echo "select distinct file, function from function_ptr where ptr='(struct rtl_ha
190         | sqlite3 $db_file | sed -e 's/|/ /' | while read file function ; do

192     drv=$(echo $file | perl -ne 's/.*\/rtlwifi\/(.*?)\/sw.c/$1/; print')
193     if [ $drv = "" ] ; then
194         continue
195     fi

197     echo "update caller_info
198           set function = '$drv (struct rtl_hal_ops)->set_hw_reg'
199           where function = '(struct rtl_hal_ops)->set_hw_reg' and file like 'dri
200         | sqlite3 $db_file

202     echo "insert into function_ptr values ('$file', '$function', '$drv (struct r
203         | sqlite3 $db_file
204 done

207 for func in __kmalloc __kmalloc_track_caller ; do

209     cat << EOF | sqlite3 $db_file
210 delete from return_states where function = '$func';
211 insert into return_states values ('faked', '$func', 0, 1, '16', 0,    0,  -1, ''
212 insert into return_states values ('faked', '$func', 0, 1, '16', 0, 103,   0, '\$
213 insert into return_states values ('faked', '$func', 0, 2, '4096-ptr_max', 0,
214 insert into return_states values ('faked', '$func', 0, 2, '4096-ptr_max', 0, 103
```

```
215 insert into return_states values ('faked', '$func', 0, 2, '4096-ptr_max', 0, 103
216 insert into return_states values ('faked', '$func', 0, 3, '0', 0,     0,  -1, '',
217 insert into return_states values ('faked', '$func', 0, 3, '0', 0,    103,  0, '\
218 EOF
219 done
```

```
**********************************************************
     181 Mon Aug  5 08:38:24 2019
new/usr/src/tools/smatch/src/smatch_data/db/function_ptr.schema
11506 smatch resync
**********************************************************
```

```
1 CREATE TABLE function_ptr (
2         file varchar(128),
3         function varchar(64),
4         ptr varchar(256),
5         searchable integer,

7         CONSTRAINT function_ptr_constraint UNIQUE (file, function, ptr)
8 );
```

```
1 CREATE TABLE function_ptr (file varchar(128), function varchar(64), ptr varchar(
```

****
1721 Mon Aug 5 08:38:24 2019
new/usr/src/tools/smatch/src/smatch_data/db/init_constraints.pl
11506 smatch resync
****
_____***unchanged_portion_omitted_***

```
 43 sub load_manual_constraints($$)
 44 {
 45     my $full_path = shift;
 46     my $project = shift;
 47     my $dir = dirname($full_path);

 49     if ($project =~ /^$/) {
 50         return;
 51     }

 53     open(FILE, "$dir/$project.constraints");
 54     while (<FILE>) {
 55         s/\n//;
 56         $db->do("insert or ignore into constraints (str) values ('$_')");
 57     }
 58     close(FILE);

 60     open(FILE, "$dir/$project.constraints_required");
 61     while (<FILE>) {
 62         my $limit;
 63         my $dummy;

 65         ($dummy, $dummy, $limit) = split(/,/);
 66         $limit =~ s/^ +//;
 67         $limit =~ s/\n//;
 68         try {
 69             $db->do("insert or ignore into constraints (str) values ('$limit')")
 70         } catch {}
 71     }
 72     close(FILE);

 74     $db->commit();
 75 }
```

_____***unchanged_portion_omitted_***

****
1178 Mon Aug 5 08:38:24 2019
new/usr/src/tools/smatch/src/smatch_data/db/init_constraints_required.pl
11506 smatch resync
****
_____***unchanged_portion_omitted_***

```
 30 sub load_manual_constraints($$)
 31 {
 32     my $full_path = shift;
 33     my $project = shift;
 34     my $dir = dirname($full_path);
 35     my ($data, $op, $limit);

 37     if ($project =~ /^$/) {
 38         return;
 39     }

 41     open(FILE, "$dir/$project.constraints_required");
 42     while (<FILE>) {
 43         ($data, $op, $limit) = split(/,/);
 44         $op =~ s/ //g;
 45         $limit =~ s/^ +//;
 46         $limit =~ s/\n//;
 47         $db->do("insert into constraints_required values (?, ?, ?);", undef, $da
 48     }
 49     close(FILE);

 51     $db->commit();
 52 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    2704 Mon Aug  5 08:38:24 2019
new/usr/src/tools/smatch/src/smatch_data/db/kernel.return_fixes
11506 smatch resync
**********************************************************
   1 i2c_master_recv s32min-s32max 1-s32max[<=$2]
   2 i2c_master_recv s32min-0,2-s32max 1-s32max[<=$2]
   3 hid_hw_output_report s32min-s32max 1-s32max[<=$2]
   4 _regmap_read s32min-(-1),1-s32max (-4095)-(-1)
   5 regmap_bulk_read s32min-(-1),1-s32max (-4095)-(-1)
   6 scnprintf s32min-s32max 0-s32max[<$1]
   7 scnprintf s32min-(-2),0-2147483646[<$1] 0-s32max[<$1]
   8 scnprintf s32min-(-2),0-2147483646 0-s32max[<$1]
   9 scnprintf s32min-s32max[<=$1] 0-s32max[<$1]
  10 scnprintf 0-s32max 0-s32max[<$1]
  11 vscnprintf s32min-(-2),0-s32max[<$1] 0-s32max[<$1]
  12 down_interruptible s32min-s32max (-62),(-4)
  13 __sock_create s32min-(-1),1-s32max (-4095)-(-1)
  14 __sock_create s32min-(-90),(-88)-(-1),1-s32max (-4095)-(-90),(-88)-(-1)
  15 sock_create_kern s32min-(-1),1-s32max (-4095)-(-1)
  16 sock_create_kern s32min-(-90),(-88)-(-1),1-s32max (-4095)-(-90),(-88)-(-1)
  17 nilfs_cpfile_get_checkpoint_block s32min-(-18),(-16)-s32max (-4095)-(-18),(-16)-
  18 nilfs_cpfile_get_checkpoint_block s32min-(-18),(-16)-(-3),(-1),1-s32max (-4095)-
  19 nilfs_mdt_insert_new_block s32min-(-23),(-21)-(-1),1-s32max (-4095)-(-23),(-21)-
  20 simple_write_to_buffer s64min-s64max 0-s32max[<=$1]
  21 atomic_read s32min-s32max s32min-s32max[==$0->counter]
  22 notifier_to_errno (-2147483646)-(-1) (-4095)-(-1)
  23 mc_status_to_error s32min-s32max (-4095)-0
  22 dma_fence_wait_timeout s64min-s64max (-4095)-s64max
  23 dma_fence_wait_timeout s32min-s32max (-4095)-s32max
  24 fls s32min-s32max 0-32
  25 fls64 s64min-s64max 0-64
  26 __bitmap_weight s32min-s32max 0-s32max[<=$1]
  27 __bitmap_weight 0-s32max 0-s32max[<=$1]
  28 __ffs 0-u64max 0-63
  29 __ffs 0-u32max 0-31
  30 find_last_bit 0-u64max 0-u32max[<=$1]
  31 __spi_sync (-524),(-115),(-108),(-22) (-4095)-0
  32 tpm_tis_spi_read_bytes s32min-s32max (-4095)-0
  33 __irq_domain_activate_irq s32min-s32max (-4095)-0
  34 get_user_pages_fast s32min-s32max 1-s32max[<=$1]
  35 get_user_pages s32min-s32max (-4095)-s32max[<=$1]
  36 get_user_pages s64min-s64max (-4095)-s64max[<=$1]
  37 get_user_pages_remote 1-s64max 1-s64max[<=$3]
  38 get_user_pages_remote (-133),(-14),(-12),1-s64max (-133),(-14),(-12),1-s64max[<=
  33 get_user_pages_fast s32min-s32max 1-s32max[<$1]
  39 __nci_request s32min-s32max (-4095)-0
  40 wait_for_common s64min-s64max 0-s64max[<=$1]
  41 wait_for_common 64min-(-1),1-s64max 1-s64max[<=$1]
  42 dma_fence_wait_timeout s64min-(-1),1-s64max (-4095)-(-1),1-s32max[<=2]
  43 dma_fence_wait_timeout s64min-s64max (-4095)-s32max
  44 dma_fence_wait_timeout s32min-s32max (-4095)-s32max
  45 __fw_state_wait_common s32min-s32max (-4095)-(-1)
  46 __ilog2_u32 s32min-s32max 0-31
  47 __ilog2_u64 s32min-s32max 0-63
  48 driver_attach s32min-s32max (-4095)-0
  49 mbox_post_sync_cmd 255 0-255
  50 mmc_io_rw_extended s32min-(-1),1-s32max (-4095)-(-1)
  51 kernel_read s64min-s64max (-4095)-1000000000
  52 security_kernel_post_read_file s32min-(-1),1-s32max (-4095)-(-1)
  53 array_index_mask_nospec 0-u64max u64max
  54 array_index_mask_nospec 0-u32max u32max
  55 nla_len (-4)-65531[$0->nla_len\ -\ 4] 0-65531[$0->nla_len\ -\ 4]
```

```
**********************************************************
   21229 Mon Aug  5 08:38:25 2019
new/usr/src/tools/smatch/src/smatch_data/db/smdb.py
11506 smatch resync
**********************************************************
```

```
   1 #!/usr/bin/python

   3 # Copyright (C) 2013 Oracle.
   4 #
   5 # Licensed under the Open Software License version 1.1

   7 import sqlite3
   8 import sys
   9 import re

  11 try:
  12     con = sqlite3.connect('smatch_db.sqlite')
  13 except sqlite3.Error, e:
  14     print "Error %s:" % e.args[0]
  15     sys.exit(1)

  17 def usage():
  18     print "%s" %(sys.argv[0])
  19     print "<function> - how a function is called"
  20     print "info <type> - how a function is called, filtered by type"
  21     print "return_states <function> - what a function returns"
  22     print "call_tree <function> - show the call tree"
  23     print "where <struct_type> <member> - where a struct member is set"
  24     print "type_size <struct_type> <member> - how a struct member is allocated"
  25     print "data_info <struct_type> <member> - information about a given data typ
  26     print "function_ptr <function> - which function pointers point to this"
  27     print "trace_param <function> <param> - trace where a parameter came from"
  28     print "locals <file> - print the local values in a file."
  29     sys.exit(1)

  31 function_ptrs = []
  32 searched_ptrs = []
  33 def get_function_pointers_helper(func):
  34     cur = con.cursor()
  35     cur.execute("select distinct ptr from function_ptr where function = '%s';" %
  36     for row in cur:
  37         ptr = row[0]
  38         if ptr in function_ptrs:
  39             continue
  40         function_ptrs.append(ptr)
  41         if not ptr in searched_ptrs:
  42             searched_ptrs.append(ptr)
  43             get_function_pointers_helper(ptr)

  45 def get_function_pointers(func):
  46     global function_ptrs
  47     global searched_ptrs
  48     function_ptrs = [func]
  49     searched_ptrs = [func]
  50     get_function_pointers_helper(func)
  51     return function_ptrs

  53 db_types = {   0: "INTERNAL",
  54              101: "PARAM_CLEARED",
  55              103: "PARAM_LIMIT",
  56              104: "PARAM_FILTER",
  57             1001: "PARAM_VALUE",
  58             1002: "BUF_SIZE",
  58             1003: "USER_DATA",
  59             1004: "CAPPED_DATA",
  60             1005: "RETURN_VALUE",
```



```
  61             1006: "DEREFERENCE",
  62             1007: "RANGE_CAP",
  63             1008: "LOCK_HELD",
  64             1009: "LOCK_RELEASED",
  65             1010: "ABSOLUTE_LIMITS",
  66             1012: "PARAM_ADD",
  67             1013: "PARAM_FREED",
  68             1014: "DATA_SOURCE",
  69             1015: "FUZZY_MAX",
  70             1016: "STR_LEN",
  71             1017: "ARRAY_LEN",
  72             1018: "CAPABLE",
  73             1019: "NS_CAPABLE",
  74             1022: "TYPE_LINK",
  75             1023: "UNTRACKED_PARAM",
  76             1024: "CULL_PATH",
  77             1025: "PARAM_SET",
  78             1026: "PARAM_USED",
  79             1027: "BYTE_UNITS",
  80             1028: "COMPARE_LIMIT",
  81             1029: "PARAM_COMPARE",
  82             1030: "EXPECTS_TYPE",
  83             1031: "CONSTRAINT",
  84             1032: "PASSES_TYPE",
  85             1033: "CONSTRAINT_REQUIRED",
  86             1034: "BIT_INFO",
  87             1035: "NOSPEC",
  88             1036: "NOSPEC_WB",
  89             1037: "STMT_CNT",
  90             1038: "TERMINATED",
  91             1039: "SLEEP",
  92             1040: "NO_SLEEP_CNT",
  93             1041: "SMALLISH",
  94             1042: "FRESH_MTAG",

  96             8017: "USER_DATA",
  97             9017: "USER_DATA_SET",
  82             8017: "USER_DATA2",
  98             8018: "NO_OVERFLOW",
  99             8019: "NO_OVERFLOW_SIMPLE",
 100             8020: "LOCKED",
 101             8021: "UNLOCKED",
 102             8023: "ATOMIC_INC",
 103             8024: "ATOMIC_DEC",
 104 };

 106 def add_range(rl, min_val, max_val):
 107     check_next = 0
 108     done = 0
 109     ret = []
 110     idx = 0

 112     if len(rl) == 0:
 113         return [[min_val, max_val]]

 115     for idx in range(len(rl)):
 116         cur_min = rl[idx][0]
 117         cur_max = rl[idx][1]

 119         # we already merged the new range but we might need to change later
 120         # ranges if they over lap with more than one
 121         if check_next:
 122             # join with added range
 123             if max_val + 1 == cur_min:
 124                 ret[len(ret) - 1][1] = cur_max
 125                 done = 1
```

```
126                 break
127             # don't overlap
128             if max_val < cur_min:
129                 ret.append([cur_min, cur_max])
130                 done = 1
131                 break
132             # partially overlap
133             if max_val < cur_max:
134                 ret[len(ret) - 1][1] = cur_max
135                 done = 1
136                 break
137             # completely overlap
138             continue

140         # join 2 ranges into one
141         if max_val + 1 == cur_min:
142             ret.append([min_val, cur_max])
143             done = 1
144             break
145         # range is entirely below
146         if max_val < cur_min:
147             ret.append([min_val, max_val])
148             ret.append([cur_min, cur_max])
149             done = 1
150             break
151         # range is partially below
152         if min_val < cur_min:
153             if max_val <= cur_max:
154                 ret.append([min_val, cur_max])
155                 done = 1
156                 break
157             else:
158                 ret.append([min_val, max_val])
159                 check_next = 1
160                 continue
161         # range already included
162         if max_val <= cur_max:
163             ret.append([cur_min, cur_max])
164             done = 1
165             break;
166         # range partially above
167         if min_val <= cur_max:
168             ret.append([cur_min, max_val])
169             check_next = 1
170             continue
171         # join 2 ranges on the other side
172         if min_val - 1 == cur_max:
173             ret.append([cur_min, max_val])
174             check_next = 1
175             continue
176         # range is above
177         ret.append([cur_min, cur_max])

179     if idx + 1 < len(rl):          # we hit a break statement
180         ret = ret + rl[idx + 1:]
181     elif done:                     # we hit a break on the last iteration
182         pass
183     elif not check_next:           # it's past the end of the rl
184         ret.append([min_val, max_val])

186     return ret;

188 def rl_union(rl1, rl2):
189     ret = []
190     for r in rl1:
191         ret = add_range(ret, r[0], r[1])
```



```
192     for r in rl2:
193         ret = add_range(ret, r[0], r[1])

195     if (rl1 or rl2) and not ret:
196         print "bug: merging %s + %s gives empty" %(rl1, rl2)

198     return ret

200 def txt_to_val(txt):
201     if txt == "s64min":
202         return -(2**63)
203     elif txt == "s32min":
204         return -(2**31)
205     elif txt == "s16min":
206         return -(2**15)
207     elif txt == "s64max":
208         return 2**63 - 1
209     elif txt == "s32max":
210         return 2**31 - 1
211     elif txt == "s16max":
212         return 2**15 - 1
213     elif txt == "u64max":
214         return 2**64 - 1
215     elif txt == "ptr_max":
216         return 2**64 - 1
217     elif txt == "u32max":
218         return 2**32 - 1
219     elif txt == "u16max":
220         return 2**16 - 1
221     else:
222         try:
223             return int(txt)
224         except ValueError:
225             return 0

227 def val_to_txt(val):
228     if val == -(2**63):
229         return "s64min"
230     elif val == -(2**31):
231         return "s32min"
232     elif val == -(2**15):
233         return "s16min"
234     elif val == 2**63 - 1:
235         return "s64max"
236     elif val == 2**31 - 1:
237         return "s32max"
238     elif val == 2**15 - 1:
239         return "s16max"
240     elif val == 2**64 - 1:
241         return "u64max"
242     elif val == 2**32 - 1:
243         return "u32max"
244     elif val == 2**16 - 1:
245         return "u16max"
246     elif val < 0:
247         return "(%d)" %(val)
248     else:
249         return "%d" %(val)

251 def get_next_str(txt):
252     val = ""
253     parsed = 0

255     if txt[0] == '(':
256         parsed += 1
257         for char in txt[1:]:
```

```
258             if char == ')':
259                 break
260             parsed += 1
261         val = txt[1:parsed]
262         parsed += 1
263     elif txt[0] == 's' or txt[0] == 'u':
264         parsed += 6
265         val = txt[:parsed]
266     else:
267         if txt[0] == '-':
268             parsed += 1
269         for char in txt[parsed:]:
270             if char == '-':
271                 break
272             parsed += 1
273         val = txt[:parsed]
274     return [parsed, val]

276 def txt_to_rl(txt):
277     if len(txt) == 0:
278         return []

280     ret = []
281     pairs = txt.split(",")
282     for pair in pairs:
283         cnt, min_str = get_next_str(pair)
284         if cnt == len(pair):
285             max_str = min_str
286         else:
287             cnt, max_str = get_next_str(pair[cnt + 1:])
288         min_val = txt_to_val(min_str)
289         max_val = txt_to_val(max_str)
290         ret.append([min_val, max_val])

292 #    Hm...  Smatch won't call INT_MAX s32max if the variable is unsigned.
293 #    if txt != rl_to_txt(ret):
294 #        print "bug: converting: text = %s rl = %s internal = %s" %(txt, rl_to_t

296     return ret

298 def rl_to_txt(rl):
299     ret = ""
300     for idx in range(len(rl)):
301         cur_min = rl[idx][0]
302         cur_max = rl[idx][1]

304         if idx != 0:
305             ret += ","

307         if cur_min == cur_max:
308             ret += val_to_txt(cur_min)
309         else:
310             ret += val_to_txt(cur_min)
311             ret += "-"
312             ret += val_to_txt(cur_max)
313     return ret

315 def type_to_str(type_int):

317     t = int(type_int)
318     if db_types.has_key(t):
319         return db_types[t]
320     return type_int

322 def type_to_int(type_string):
323     for k in db_types.keys():
```



```
324         if db_types[k] == type_string:
325             return k
326     return -1

328 def display_caller_info(printed, cur, param_names):
329     for txt in cur:
330         if not printed:
331             print "file | caller | function | type | parameter | key | value |"
332         printed = 1

334         parameter = int(txt[6])
335         key = txt[7]
336         if len(param_names) and parameter in param_names:
337             key = key.replace("$", param_names[parameter])

339         print "%20s | %20s | %20s |" %(txt[0], txt[1], txt[2]),
340         print " %10s |" %(type_to_str(txt[5])),
341         print " %d | %s | %s" %(parameter, key, txt[8])
342     return printed

344 def get_caller_info(filename, ptrs, my_type):
345     cur = con.cursor()
346     param_names = get_param_names(filename, func)
347     printed = 0
348     type_filter = ""
349     if my_type != "":
350         type_filter = "and type = %d" %(type_to_int(my_type))
351     for ptr in ptrs:
352         cur.execute("select * from caller_info where function = '%s' %s;" %(ptr,
353         printed = display_caller_info(printed, cur, param_names)

355 def print_caller_info(filename, func, my_type = ""):
356     ptrs = get_function_pointers(func)
357     get_caller_info(filename, ptrs, my_type)

359 def merge_values(param_names, vals, cur):
360     for txt in cur:
361         parameter = int(txt[0])
362         name = txt[1]
363         rl = txt_to_rl(txt[2])
364         if parameter in param_names:
365             name = name.replace("$", param_names[parameter])

367         if not parameter in vals:
368             vals[parameter] = {}

370         # the first item on the list is the number of rows.  it's incremented
371         # every time we call merge_values().
372         if name in vals[parameter]:
373             vals[parameter][name] = [vals[parameter][name][0] + 1, rl_union(vals
374         else:
375             vals[parameter][name] = [1, rl]

377 def get_param_names(filename, func):
378     cur = con.cursor()
379     param_names = {}
380     cur.execute("select parameter, value from parameter_name where file = '%s' a
381     for txt in cur:
382         parameter = int(txt[0])
383         name = txt[1]
384         param_names[parameter] = name
385     if len(param_names):
386         return param_names

388     cur.execute("select parameter, value from parameter_name where function = '%
389     for txt in cur:
```

```
390         parameter = int(txt[0])
391         name = txt[1]
392         param_names[parameter] = name
393     return param_names

395 def get_caller_count(ptrs):
396     cur = con.cursor()
397     count = 0
398     for ptr in ptrs:
399         cur.execute("select count(distinct(call_id)) from caller_info where func
400         for txt in cur:
401             count += int(txt[0])
402     return count

404 def print_merged_caller_values(filename, func, ptrs, param_names, call_cnt):
405     cur = con.cursor()
406     vals = {}
407     for ptr in ptrs:
408         cur.execute("select parameter, key, value from caller_info where functio
409         merge_values(param_names, vals, cur);

411     for param in sorted(vals):
412         for name in sorted(vals[param]):
413             if vals[param][name][0] != call_cnt:
414                 continue
415             print "%d %s -> %s" %(param, name, rl_to_txt(vals[param][name][1]))


418 def print_unmerged_caller_values(filename, func, ptrs, param_names):
419     cur = con.cursor()
420     for ptr in ptrs:
421         prev = -1
422         cur.execute("select file, caller, call_id, parameter, key, value from ca
423         for filename, caller, call_id, parameter, name, value in cur:
424             if prev != int(call_id):
425                 prev = int(call_id)

427             parameter = int(parameter)
428             if parameter < len(param_names):
429                 name = name.replace("$", param_names[parameter])
430             else:
431                 name = name.replace("$", "$%d" %(parameter))

433             print "%s | %s | %s | %s" %(filename, caller, name, value)
434         print "========================="

436 def print_caller_values(filename, func, ptrs):
437     param_names = get_param_names(filename, func)
438     call_cnt = get_caller_count(ptrs)

440     print_merged_caller_values(filename, func, ptrs, param_names, call_cnt)
441     print "========================="
442     print_unmerged_caller_values(filename, func, ptrs, param_names)

444 def caller_info_values(filename, func):
445     ptrs = get_function_pointers(func)
446     print_caller_values(filename, func, ptrs)

448 def print_return_states(func):
449     cur = con.cursor()
450     cur.execute("select * from return_states where function = '%s';" %(func))
451     count = 0
452     for txt in cur:
453         printed = 1
454         if count == 0:
455             print "file | function | return_id | return_value | type | param | k
```



```
456         count += 1
457         print "%s | %s | %2s | %13s" %(txt[0], txt[1], txt[3], txt[4]),
458         print "| %13s |" %(type_to_str(txt[6])),
459         print " %2d | %20s | %20s |" %(txt[7], txt[8], txt[9])

461 def print_return_implies(func):
462     cur = con.cursor()
463     cur.execute("select * from return_implies where function = '%s';" %(func))
464     count = 0
465     for txt in cur:
466         if not count:
467             print "file | function | type | param | key | value |"
468         count += 1
469         print "%15s | %15s" %(txt[0], txt[1]),
470         print "| %15s" %(type_to_str(txt[4])),
471         print "| %3d | %s | %15s |" %(txt[5], txt[6], txt[7])

473 def print_type_size(struct_type, member):
474     cur = con.cursor()
475     cur.execute("select * from type_size where type like '(struct %s)->%s';" %(s
476     print "type | size"
477     for txt in cur:
478         print "%-15s | %s" %(txt[0], txt[1])

480     cur.execute("select * from function_type_size where type like '(struct %s)->
481     print "file | function | type | size"
482     for txt in cur:
483         print "%-15s | %-15s | %-15s | %s" %(txt[0], txt[1], txt[2], txt[3])

485 def print_data_info(struct_type, member):
486     cur = con.cursor()
487     cur.execute("select * from data_info where data like '(struct %s)->%s';" %(s
488     print "file | data | type | value"
489     for txt in cur:
490         print "%-15s | %-15s | %-15s | %s" %(txt[0], txt[1], type_to_str(txt[2])

492 def print_fn_ptrs(func):
493     ptrs = get_function_pointers(func)
494     if not ptrs:
495         return
496     print "%s = " %(func),
497     print(ptrs)

499 def print_functions(member):
500     cur = con.cursor()
501     cur.execute("select * from function_ptr where ptr like '%%->%s';" %(member))
502     print "File | Pointer | Function | Static"
503     for txt in cur:
504         print "%-15s | %-15s | %-15s | %s" %(txt[0], txt[2], txt[1], txt[3])

506 def get_callers(func):
507     ret = []
508     cur = con.cursor()
509     ptrs = get_function_pointers(func)
510     for ptr in ptrs:
511         cur.execute("select distinct caller from caller_info where function = '%
512         for row in cur:
513             ret.append(row[0])
514     return ret

516 printed_funcs = []
517 def call_tree_helper(func, indent = 0):
518     global printed_funcs
519     if func in printed_funcs:
520         return
521     print "%s%s()" %(" " * indent, func)
```

```
522     if func == "too common":
523         return
524     if indent > 6:
525         return
526     printed_funcs.append(func)
527     callers = get_callers(func)
528     if len(callers) >= 20:
529         print "Over 20 callers for %s()" %(func)
530         return
531     for caller in callers:
532         call_tree_helper(caller, indent + 2)

534 def print_call_tree(func):
535     global printed_funcs
536     printed_funcs = []
537     call_tree_helper(func)

539 def function_type_value(struct_type, member):
540     cur = con.cursor()
541     cur.execute("select * from function_type_value where type like '(struct %s)-
542     for txt in cur:
543         print "%-30s | %-30s | %s | %s" %(txt[0], txt[1], txt[2], txt[3])

545 def trace_callers(func, param):
546     sources = []
547     prev_type = 0

549     cur = con.cursor()
550     ptrs = get_function_pointers(func)
551     for ptr in ptrs:
552         cur.execute("select type, caller, value from caller_info where function
553         for row in cur:
554             data_type = int(row[0])
555             if data_type == 1014:
556                 sources.append((row[1], row[2]))
557             elif data_type == 1028:
558                 sources.append(("%", row[2])) # hack...
559             elif data_type == 0 and prev_type == 0:
560                 sources.append((row[1], ""))
561             prev_type = data_type
562     return sources

564 def trace_param_helper(func, param, indent = 0):
565     global printed_funcs
566     if func in printed_funcs:
567         return
568     print "%s%s(param %d)" %(" " * indent, func, param)
569     if func == "too common":
570         return
571     if indent > 20:
572         return
573     printed_funcs.append(func)
574     sources = trace_callers(func, param)
575     for path in sources:

577         if len(path[1]) and path[1][0] == 'p' and path[1][1] == ' ':
578             p = int(path[1][2:])
579             trace_param_helper(path[0], p, indent + 2)
580         elif len(path[0]) and path[0][0] == '%':
581             print "  %s%s" %(" " * indent, path[1])
582         else:
583             print "* %s%s %s" %(" " * (indent - 1), path[0], path[1])

585 def trace_param(func, param):
586     global printed_funcs
587     printed_funcs = []
```



```
588     print "tracing %s %d" %(func, param)
589     trace_param_helper(func, param)

591 def print_locals(filename):
592     cur = con.cursor()
593     cur.execute("select file,data,value from data_info where file = '%s' and typ
594     for txt in cur:
595         print "%s | %s | %s" %(txt[0], txt[1], txt[2])

597 def constraint(struct_type, member):
598     cur = con.cursor()
599     cur.execute("select * from constraints_required where data like '(struct %s)
600     for txt in cur:
601         print "%-30s | %-30s | %s | %s" %(txt[0], txt[1], txt[2], txt[3])

603 if len(sys.argv) < 2:
604     usage()

606 if len(sys.argv) == 2:
607     func = sys.argv[1]
608     print_caller_info("", func)
609 elif sys.argv[1] == "info":
610     my_type = ""
611     if len(sys.argv) == 4:
612         my_type = sys.argv[3]
613     func = sys.argv[2]
614     print_caller_info("", func, my_type)
615 elif sys.argv[1] == "call_info":
616     if len(sys.argv) != 4:
617         usage()
618     filename = sys.argv[2]
619     func = sys.argv[3]
620     caller_info_values(filename, func)
621     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")
622 elif sys.argv[1] == "function_ptr" or sys.argv[1] == "fn_ptr":
623     func = sys.argv[2]
624     print_fn_ptrs(func)
625 elif sys.argv[1] == "return_states":
626     func = sys.argv[2]
627     print_return_states(func)
628     print "================================================"
629     print_return_implies(func)
630 elif sys.argv[1] == "return_implies":
631     func = sys.argv[2]
632     print_return_implies(func)
633 elif sys.argv[1] == "type_size" or sys.argv[1] == "buf_size":
634     struct_type = sys.argv[2]
635     member = sys.argv[3]
636     print_type_size(struct_type, member)
637 elif sys.argv[1] == "data_info":
638     struct_type = sys.argv[2]
639     member = sys.argv[3]
640     print_data_info(struct_type, member)
641 elif sys.argv[1] == "call_tree":
642     func = sys.argv[2]
643     print_call_tree(func)
644 elif sys.argv[1] == "where":
645     if len(sys.argv) == 3:
646         struct_type = "%"
647         member = sys.argv[2]
```

```
    elif len(sys.argv) == 4:
        struct_type = sys.argv[2]
        member = sys.argv[3]
    function_type_value(struct_type, member)
elif sys.argv[1] == "local":
    filename = sys.argv[2]
    variable = ""
    if len(sys.argv) == 4:
        variable = sys.argv[3]
    local_values(filename, variable)
elif sys.argv[1] == "functions":
    member = sys.argv[2]
    print_functions(member)
elif sys.argv[1] == "trace_param":
    if len(sys.argv) != 4:
        usage()
    func = sys.argv[2]
    param = int(sys.argv[3])
    trace_param(func, param)
elif sys.argv[1] == "locals":
    if len(sys.argv) != 3:
        usage()
    filename = sys.argv[2]
    print_locals(filename);
elif sys.argv[1] == "constraint":
    if len(sys.argv) == 3:
        struct_type = "%"
        member = sys.argv[2]
    elif len(sys.argv) == 4:
        struct_type = sys.argv[2]
        member = sys.argv[3]
    constraint(struct_type, member)
elif sys.argv[1] == "test":
    filename = sys.argv[2]
    func = sys.argv[3]
    caller_info_values(filename, func)
else:
    usage()
```

```
**********************************************************
```
665 Mon Aug 5 08:38:25 2019
new/usr/src/tools/smatch/src/smatch_data/db/vim_smdb
11506 smatch resync
```
**********************************************************
```

```
  1 #!/bin/bash

  3 # Add these lines to your .vimrc file
  4 #
  5 # map <C-r> :! vim_smdb return_states <cword> <CR> :execute 'edit' system("cat ~
  6 # map <C-c> :! vim_smdb <cword> <CR> :execute 'edit' system("cat ~/.smdb_tmp/cur
  7 #
  8 # Now you can move your cursor over a function and hit CTRL-c to see how it's
  9 # called or CTRL-r to see what it returns.  Use the ":bd" command to get back to
 10 # your source.

 12 DIR="$HOME/.smdb_tmp"
 13 mkdir -p $DIR

 15 for i in $(seq 1 100) ; do
 16         if [ ! -e $DIR/$i ] ; then
 17                 break
 18         fi
 19 done

 21 if [ $i == 100 ] ; then
 22         i=1
 23 fi

 25 next=$(($i + 1))

 27 rm -f $DIR/$next
 28 rm -f $DIR/.${i}.swp
 28 rm $DIR/.${i}.swp
 29 smdb $* > $DIR/$i

 31 echo "$DIR/$i" > $DIR/cur
```

****
227 Mon Aug 5 08:38:25 2019
new/usr/src/tools/smatch/src/smatch_data/illumos_kernel.skipped_functions
11506 smatch resync
****

```
/* These are "too hairy" for smatch. */
ECDSA_VerifyDigest
dtrace_disx86
elf32exec
elfexec
iscsi_ioctl
lm_idle_chk
ld64_sym_validate
luaV_settable
nostore_generate_key_pair
sadb_common_add
segvn_fault_vnodepages
tcp_input_data
```

****
698 Mon Aug 5 08:38:26 2019
new/usr/src/tools/smatch/src/smatch_data/illumos_user.skipped_functions
11506 smatch resync
****

```
/*
 * 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/mdb */
iob_doprnt
/* cmd/pppd */
lcp_nakci
/* cmd/cmd-crypto */
execute_cmd

/* generated code */
ipf_yyparse
ipmon_yyparse
ipnat_yyparse
ippool_yyparse
ndr__ndr_hdr
yyerror
yylex
yylook
yyparse
yywinput
```

****
*** 46 Mon Aug 5 08:38:26 2019***
new/usr/src/tools/smatch/src/smatch_data/kernel.allocation_funcs_gfp.remove
11506 smatch resync
****

```
   1 acquire_group
   2 acquire_group 2
   3 acquire_group X
```

```
**********************************************************
     201 Mon Aug  5 08:38:26 2019
new/usr/src/tools/smatch/src/smatch_data/kernel.ignore_casted_params
11506 smatch resync
**********************************************************
```

```
set_bit
clear_bit
__clear_bit
__set_bit
test_and_set_bit
find_last_bit
change_bit
xfs_next_bit
find_next_bit
find_first_bit
__test_and_set_bit
sync_set_bit
bitmap_weight
bitmap_intersects
bitmap_empty
```

```
**********************************************************
    1255 Mon Aug  5 08:38:26 2019
new/usr/src/tools/smatch/src/smatch_data/kernel.ignore_side_effects
11506 smatch resync
**********************************************************
```

```
/*
 * 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
SK_REUSEPORT_LOAD_SKB_FIELD
SK_REUSEPORT_LOAD_SK_FIELD_SIZE_OFF
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_FIELD
SOCK_OPS_GET_OR_SET_FIELD
SOCK_OPS_GET_TCP32
unsafe_get_user
unsafe_put_user
VIA_OUT_RING_QW
WRITE64
Z
```

```
**********************************************************
    2207 Mon Aug  5 08:38:26 2019
new/usr/src/tools/smatch/src/smatch_data/kernel.ignore_uninitialized_param
11506 smatch resync
**********************************************************
```

```
  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
105 pci_user_read_config_byte 2
106 gen11_gu_misc_irq_ack 2
107 vsc73xx_read 4
108 smb_hc_read 2
109 smb_word_op 5
110 atl1c_read_phy_reg 2
111 adf7242_read_reg 2
```

```
**********************************************************
      35 Mon Aug  5 08:38:27 2019
new/usr/src/tools/smatch/src/smatch_data/kernel.ignored_warnings
11506 smatch resync
**********************************************************
   1 check_shift_to_zero overflows_type
```

****
210 Mon Aug 5 08:38:27 2019
new/usr/src/tools/smatch/src/smatch_data/kernel.no_inline_functions
11506 smatch resync
****

```
__fswab16
__fswab32
__fswab64
__builtin_bswap16
__builtin_bswap32
__builtin_bswap64
__arch_hweight8
__arch_hweight16
__arch_hweight32
__arch_hweight64
__write_once_size
atomic_set
atomic_read
notifier_to_errno
```

```
**********************************************************
      30 Mon Aug  5 08:38:27 2019
new/usr/src/tools/smatch/src/smatch_data/kernel.no_return_funcs.add
11506 smatch resync
**********************************************************
   1 YY_FATAL_ERROR
   2 malformed_line
```

```
**********************************************************
     250 Mon Aug  5 08:38:27 2019
new/usr/src/tools/smatch/src/smatch_data/kernel.silenced_functions
11506 smatch resync
**********************************************************
   1 /* Don't print anything from these functions */
   2 atomic_dec_and_test
   3 atomic_inc_and_test
   4 atomic64_dec_and_test
   5 atomic_sub_and_test
   6 test_and_clear_bit
   7 test_and_set_bit
   7 __copy_to_user_nocheck
   8 __copy_from_user_nocheck
   9 arch_static_branch
  10 __static_cpu_has
  11 __read_once_size
```

```
**********************************************************
    2765 Mon Aug  5 08:38:28 2019
new/usr/src/tools/smatch/src/smatch_data_source.c
11506 smatch resync
**********************************************************
```

```
 1 /*
 2  * Copyright (C) 2013 Oracle.
 3  *
 4  * This program is free software; you can redistribute it and/or
 5  * modify it under the terms of the GNU General Public License
 6  * as published by the Free Software Foundation; either version 2
 7  * of the License, or (at your option) any later version.
 8  *
 9  * This program is distributed in the hope that it will be useful,
10  * but WITHOUT ANY WARRANTY; without even the implied warranty of
11  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
12  * GNU General Public License for more details.
13  *
14  * You should have received a copy of the GNU General Public License
15  * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
16  */

18 #include "smatch.h"
19 #include "smatch_slist.h"
20 #include "smatch_extra.h"

22 static int my_id;

24 static char *get_source_parameter(struct expression *expr)
25 {
26         struct expression *tmp;
27         const char *param_name;
28         struct symbol *sym;
29         char *name;
30         int param;
31         char *ret = NULL;
32         char buf[32];
33         int cnt = 0;
34         bool modified = false;

36         tmp = expr;
37         while ((tmp = get_assigned_expr(tmp))) {
38                 expr = tmp;
39                 if (cnt++ > 3)
40                         break;
41         }

43         expr = strip_expr(expr);
44         if (expr->type != EXPR_SYMBOL)
45                 return NULL;

47         name = expr_to_var_sym(expr, &sym);
48         if (!name || !sym)
49                 goto free;
50         param = get_param_num_from_sym(sym);
51         if (param < 0)
52                 goto free;
53         param_name = get_param_name_var_sym(name, sym);
54         if (!param_name)
51         if (param_was_set(expr))
55                 goto free;
56         if (param_was_set_var_sym(name, sym))
57                 modified = true;

59         snprintf(buf, sizeof(buf), "$%d%s%s", param, param_name + 1,
60                  modified ? " [m]" : "");
```



```
54         snprintf(buf, sizeof(buf), "p %d", param);
61         ret = alloc_string(buf);

63 free:
64         free_string(name);
65         return ret;
66 }
```

_____*unchanged_portion_omitted_*

```
**********************************************************
   64159 Mon Aug  5 08:38:28 2019
new/usr/src/tools/smatch/src/smatch_db.c
11506 smatch resync
**********************************************************
_____unchanged_portion_omitted_
  69 ALLOCATOR(db_implies_callback, "return_implies callbacks");
  70 DECLARE_PTR_LIST(db_implies_cb_list, struct db_implies_callback);
  71 static struct db_implies_cb_list *return_implies_cb_list;
  72 static struct db_implies_cb_list *call_implies_cb_list;

  74 /* silently truncates if needed. */
  75 char *escape_newlines(const char *str)
  76 {
  77         char buf[1024] = "";
  78         bool found = false;
  79         int i, j;

  81         for (i = 0, j = 0; str[i] != '\0' && j != sizeof(buf); i++, j++) {
  82                 if (str[i] != '\r' && str[i] != '\n') {
  82                 if (str[i] != '\n') {
  83                         buf[j] = str[i];
  84                         continue;
  85                 }

  87                 found = true;
  88                 buf[j++] = '\\';
  89                 if (j == sizeof(buf))
  90                         break;
  91                 buf[j] = 'n';
  92         }

  94         if (!found)
  95                 return alloc_sname(str);

  97         if (j == sizeof(buf))
  98                 buf[j - 1] = '\0';
  99         return alloc_sname(buf);
 100 }
_____unchanged_portion_omitted_

 253 void sql_insert_function_ptr(const char *fn, const char *struct_name)
 254 {
 255         sql_insert_or_ignore(function_ptr, "'%s', '%s', '%s', 0",
 256                              get_base_file(), fn, struct_name);
 255         sql_insert(function_ptr, "'%s', '%s', '%s', 0", get_base_file(), fn,
 256                    struct_name);
 257 }
_____unchanged_portion_omitted_

 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');", esc
 334         sm_msg("SQL: insert or ignore into constraints (str) values('%s');", con
 335 }
_____unchanged_portion_omitted_

 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 }

 375 void sql_insert_mtag_map(mtag_t tag, int offset, mtag_t container)
```



```
 376 {
 377         sql_insert(mtag_map, "%lld, %d, %lld", tag, offset, container);
 378 }
_____unchanged_portion_omitted_

 748 int is_recursive_member(const char *name)
 749 {
 750         char buf[256];
 751         const char *p, *next;
 752         int size;

 754         p = strchr(name, '>');
 755         if (!p)
 756                 return 0;
 757         p++;
 758         while (true) {
 759                 next = strchr(p, '>');
 760                 if (!next)
 761                         return 0;
 762                 next++;

 764                 size = next - p;
 765                 if (size >= sizeof(buf))
 766                         return 0;
 767                 memcpy(buf, p, size);
 768                 buf[size] = '\0';
 769                 if (strstr(next, buf))
 770                         return 1;
 771                 p = next;
 772         }
 773 }

 775 static void print_struct_members(struct expression *call, struct expression *exp
 776         void (*callback)(struct expression *call, int param, char *printed_name,
 777 {
 778         struct sm_state *sm;
 779         const char *sm_name;
 780         char *name;
 781         struct symbol *sym;
 782         int len;
 783         char printed_name[256];
 784         int is_address = 0;
 785         bool add_star;
 786         struct symbol *type;

 788         expr = strip_expr(expr);
 789         if (!expr)
 790                 return;
 791         type = get_type(expr);
 792         if (type && type_bits(type) < type_bits(&ulong_ctype))
 793                 return;

 795         if (expr->type == EXPR_PREOP && expr->op == '&') {
 796                 expr = strip_expr(expr->unop);
 797                 is_address = 1;
 798         }

 772         type = get_type(expr);
 773         if (type && type_bits(type) < type_bits(&ulong_ctype))
 774                 return;

 800         name = expr_to_var_sym(expr, &sym);
 801         if (!name || !sym)
 802                 goto free;

 804         len = strlen(name);
```

```
 805         FOR_EACH_SM(stree, sm) {
 806                 if (sm->sym != sym)
 807                         continue;
 808                 sm_name = sm->name;
 809                 add_star = false;
 810                 if (sm_name[0] == '*') {
 811                         add_star = true;
 812                         sm_name++;
 813                 }
 814                 // FIXME: simplify?
 815                 if (!add_star && strcmp(name, sm_name) == 0) {
 784                 if (strcmp(name, sm->name) == 0) {
 816                         if (is_address)
 817                                 snprintf(printed_name, sizeof(printed_name), "*$
 818                         else /* these are already handled. fixme: handle them he
 819                                 continue;
 820                 } else if (add_star && strcmp(name, sm_name) == 0) {
 821                         snprintf(printed_name, sizeof(printed_name), "%s*$%s",
 822                                  is_address ? "*" : "", show_offset(offset));
 823                 } else if (strncmp(name, sm_name, len) == 0) {
 824                         if (sm_name[len] != '.' && sm_name[len] != '-')
 789                 } else if (sm->name[0] == '*' && strcmp(name, sm->name + 1) == 0
 790                         snprintf(printed_name, sizeof(printed_name), "*$%s", sho
 791                 } else if (strncmp(name, sm->name, len) == 0) {
 792                         if (isalnum(sm->name[len]))
 825                                 continue;
 826                         if (is_address)
 827                                 snprintf(printed_name, sizeof(printed_name),
 828                                         "%s$%s->%s", add_star ? "*" : "",
 829                                         show_offset(offset), sm_name + len + 1)
 795                                 snprintf(printed_name, sizeof(printed_name), "$%
 830                         else
 831                                 snprintf(printed_name, sizeof(printed_name),
 832                                         "%s$%s%s", add_star ? "*" : "",
 833                                         show_offset(offset), sm_name + len);
 797                                 snprintf(printed_name, sizeof(printed_name), "$%
 834                 } else {
 835                         continue;
 836                 }
 837                 if (is_recursive_member(printed_name))
 838                         continue;
 839                 callback(call, param, printed_name, sm);
 840         } END_FOR_EACH_SM(sm);
 841 free:
 842         free_string(name);
 843 }
______unchanged_portion_omitted_

1045 static char *get_next_ptr_name(void)
1046 {
1047         char *ptr;

1049         FOR_EACH_PTR(ptr_names, ptr) {
1050                 if (!insert_string(&ptr_names_done, ptr))
1012                 if (list_has_string(ptr_names_done, ptr))
1051                         continue;
1014                 insert_string(&ptr_names_done, ptr);
1052                 return ptr;
1053         } END_FOR_EACH_PTR(ptr);
1054         return NULL;
1055 }
______unchanged_portion_omitted_

1264 static char *get_return_compare_is_param(struct expression *expr)
1227 static void print_initializer_list(struct expression_list *expr_list,
1228                 struct symbol *struct_type)
```



```
1265 {
1266         char *var;
1267         char buf[256];
1268         int comparison;
1269         int param;
1230         struct expression *expr;
1231         struct symbol *base_type;
1232         char struct_name[256];

1271         param = get_param_num(expr);
1272         if (param < 0)
1273                 return NULL;

1275         var = expr_to_var(expr);
1276         if (!var)
1277                 return NULL;
1278         snprintf(buf, sizeof(buf), "%s orig", var);
1279         comparison = get_comparison_strings(var, buf);
1280         free_string(var);

1282         if (!comparison)
1283                 return NULL;

1285         snprintf(buf, sizeof(buf), "[%s$%d]", show_special(comparison), param);
1286         return alloc_sname(buf);
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);
1287 }

1289 static char *get_return_compare_str(struct expression *expr)
1255 static void global_variable(struct symbol *sym)
1290 {
1291         char *compare_str;
1257         struct symbol *struct_type;

1293         compare_str = get_return_compare_is_param(expr);
1294         if (compare_str)
1295                 return compare_str;

1297         compare_str = expr_lte_to_param(expr, -1);
1298         if (compare_str)
1299                 return compare_str;

1301         return expr_param_comparison(expr, -1);
1302 }

1304 static const char *get_return_ranges_str(struct expression *expr, struct range_l
1305 {
1306         struct range_list *rl;
```

```
1307          char *return_ranges;
1308          sval_t sval;
1309          char *compare_str;
1310          char *math_str;
1311          char buf[128];

1313          *rl_p = NULL;

1315          if (!expr)
1316                  return alloc_sname("");

1318          if (get_implied_value(expr, &sval)) {
1319                  sval = sval_cast(cur_func_return_type(), sval);
1320                  *rl_p = alloc_rl(sval, sval);
1321                  return sval_to_str_or_err_ptr(sval);
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;
1322          }

1324          compare_str = expr_equal_to_param(expr, -1);
1325          math_str = get_value_in_terms_of_parameter_math(expr);

1327          if (get_implied_rl(expr, &rl) && !is_whole_rl(rl)) {
1328                  rl = cast_rl(cur_func_return_type(), rl);
1329                  return_ranges = show_rl(rl);
1330          } else if (get_imaginary_absolute(expr, &rl)){
1331                  rl = cast_rl(cur_func_return_type(), rl);
1332                  return alloc_sname(show_rl(rl));
1333          } else {
1334                  get_absolute_rl(expr, &rl);
1335                  rl = cast_rl(cur_func_return_type(), rl);
1336                  return_ranges = show_rl(rl);
1337          }
1338          *rl_p = rl;

1340          if (compare_str) {
1341                  snprintf(buf, sizeof(buf), "%s%s", return_ranges, compare_str);
1342                  return alloc_sname(buf);
1343          }
1344          if (math_str) {
1345                  snprintf(buf, sizeof(buf), "%s[%s]", return_ranges, math_str);
1346                  return alloc_sname(buf);
1347          }
1348          compare_str = get_return_compare_str(expr);
1349          if (compare_str) {
1350                  snprintf(buf, sizeof(buf), "%s%s", return_ranges, compare_str);
1351                  return alloc_sname(buf);
1352          }

1354          return return_ranges;
1271          if (struct_type->type != SYM_STRUCT || !struct_type->ident)
1272                  return;
1273          print_initializer_list(sym->initializer->expr_list, struct_type);
1355  }
```

_____***unchanged_portion_omitted_***

```
1362  static void call_return_state_hooks_conditional(struct expression *expr)
```



```
1363  {
1364          struct returned_state_callback *cb;
1365          struct range_list *rl;
1366          const char *return_ranges;
1285          char *return_ranges;
1367          int final_pass_orig = final_pass;

1369          __push_fake_cur_stree();

1371          final_pass = 0;
1372          __split_whole_condition(expr->conditional);
1373          final_pass = final_pass_orig;

1375          return_ranges = get_return_ranges_str(expr->cond_true ?: expr->condition

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);
1377          set_state(RETURN_ID, "return_ranges", NULL, alloc_estate_rl(rl));

1379          return_id++;
1380          FOR_EACH_PTR(returned_state_callbacks, cb) {
1381                  cb->callback(return_id, (char *)return_ranges, expr->cond_true);
1303                  cb->callback(return_id, return_ranges, expr->cond_true);
1382          } END_FOR_EACH_PTR(cb);

1384          __push_true_states();
1385          __use_false_states();

1387          return_ranges = get_return_ranges_str(expr->cond_false, &rl);
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);
1388          set_state(RETURN_ID, "return_ranges", NULL, alloc_estate_rl(rl));

1390          return_id++;
1391          FOR_EACH_PTR(returned_state_callbacks, cb) {
1392                  cb->callback(return_id, (char *)return_ranges, expr->cond_false)
1318                  cb->callback(return_id, return_ranges, expr->cond_false);
1393          } END_FOR_EACH_PTR(cb);

1395          __merge_true_states();
1396          __free_fake_cur_stree();
1397  }
```

_____***unchanged_portion_omitted_***

```
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 }

1457 static int split_possible_helper(struct sm_state *sm, struct expression *expr)
1458 {
1459         struct returned_state_callback *cb;
1460         struct range_list *rl;
1461         char *return_ranges;
1462         struct sm_state *tmp;
1463         int ret = 0;
1464         int nr_possible, nr_states;
1465         char *compare_str;
1420         char *compare_str = NULL;
1466         char buf[128];
1467         struct state_list *already_handled = NULL;
1468         sval_t sval;

1470         if (!sm || !sm->merged)
1471                 return 0;

1473         if (too_many_possible(sm))
1474                 return 0;

1476         /* bail if it gets too complicated */
1477         nr_possible = 0;
1478         FOR_EACH_PTR(sm->possible, tmp) {
1479                 if (tmp->merged)
1480                         continue;
1481                 nr_possible++;
1482         } END_FOR_EACH_PTR(tmp);
1431         nr_possible = ptr_list_size((struct ptr_list *)sm->possible);
1483         nr_states = get_db_state_count();
1484         if (nr_states * nr_possible >= 2000)
1485                 return 0;

1487         FOR_EACH_PTR(sm->possible, tmp) {
1488                 if (tmp->merged)
1489                         continue;
1490                 if (ptr_in_list(tmp, already_handled))
1491                         continue;
1492                 add_ptr_list(&already_handled, tmp);

1494                 ret = 1;
1495                 __push_fake_cur_stree();

1497                 overwrite_states_using_pool(sm, tmp);

1499                 rl = cast_rl(cur_func_return_type(), estate_rl(tmp->state));
1500                 return_ranges = show_rl(rl);
1501                 set_state(RETURN_ID, "return_ranges", NULL, alloc_estate_rl(clon
1502                 if (!rl_to_sval(rl, &sval)) {
1503                         compare_str = get_return_compare_str(expr);
1504                         if (compare_str) {
1505                                 snprintf(buf, sizeof(buf), "%s%s", return_ranges
1506                                 return_ranges = alloc_sname(buf);
1507                         }
```



```
1508                 }

1510                 return_id++;
1511                 FOR_EACH_PTR(returned_state_callbacks, cb) {
1512                         cb->callback(return_id, return_ranges, expr);
1513                 } END_FOR_EACH_PTR(cb);

1515                 __free_fake_cur_stree();
1516         } END_FOR_EACH_PTR(tmp);

1518         free_slist(&already_handled);

1520         return ret;
1521 }
_____unchanged_portion_omitted_

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];

1490         *rl_p = NULL;

1492         if (!expr)
1493                 return alloc_sname("");

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         }

1501         compare_str = expr_equal_to_param(expr, -1);
1502         math_str = get_value_in_terms_of_parameter_math(expr);

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;

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         }

1530         return return_ranges;
```

```
1531 }

1534 static bool has_possible_negative(struct sm_state *sm)
1535 {
1536         struct sm_state *tmp;

1538         FOR_EACH_PTR(sm->possible, tmp) {
1539                 if (!estate_rl(tmp->state))
1540                         continue;
1541                 if (sval_is_negative(estate_min(tmp->state)) &&
1542                     sval_is_negative(estate_max(tmp->state)))
1543                         return true;
1544         } END_FOR_EACH_PTR(tmp);

1546         return false;
1547 }
_____unchanged_portion_omitted_

1564 static int split_positive_from_negative(struct expression *expr)
1565 {
1566         struct sm_state *sm;
1567         struct returned_state_callback *cb;
1568         struct range_list *rl;
1569         const char *return_ranges;
1570         struct range_list *ret_rl;
1571         int undo;
1572         bool has_zero;

1574         /* We're going to print the states 3 times */
1575         if (get_db_state_count() > 10000 / 3)
1576                 return 0;

1578         if (!get_implied_rl(expr, &rl) || !rl)
1579                 return 0;
1580         if (is_whole_rl(rl) || is_whole_rl_non_zero(rl))
1581                 return 0;
1582         /* Forget about INT_MAX and larger */
1583         if (rl_max(rl).value <= 0)
1584                 return 0;
1585         if (!sval_is_negative(rl_min(rl)))
1586                 return 0;

1588         sm = get_sm_state_expr(SMATCH_EXTRA, expr);
1589         if (!sm)
1590                 return 0;
1591         if (!has_possible_negative(sm))
1592                 return 0;
1593         has_zero = has_possible_zero_null(sm);

1595         if (!assume(compare_expression(expr, has_zero ? '>' : SPECIAL_GTE, zero_
1592         if (!assume(compare_expression(expr, '>', zero_expr())))
1596                 return 0;

1598         return_id++;
1599         return_ranges = get_return_ranges_str(expr, &ret_rl);
1600         set_state(RETURN_ID, "return_ranges", NULL, alloc_estate_rl(ret_rl));
1601         FOR_EACH_PTR(returned_state_callbacks, cb) {
1602                 cb->callback(return_id, (char *)return_ranges, expr);
1603         } END_FOR_EACH_PTR(cb);

1605         end_assume();

1607         if (has_zero) {
1604         if (rl_has_sval(rl, sval_type_val(rl_type(rl), 0))) {
1608                 undo = assume(compare_expression(expr, SPECIAL_EQUAL, zero_expr(
```



```
1610                 return_id++;
1611                 return_ranges = get_return_ranges_str(expr, &ret_rl);
1612                 set_state(RETURN_ID, "return_ranges", NULL, alloc_estate_rl(ret_
1613                 FOR_EACH_PTR(returned_state_callbacks, cb) {
1614                         cb->callback(return_id, (char *)return_ranges, expr);
1615                 } END_FOR_EACH_PTR(cb);

1617                 if (undo)
1618                         end_assume();
1619         }

1621         undo = assume(compare_expression(expr, '<', zero_expr()));

1623         return_id++;
1624         return_ranges = get_return_ranges_str(expr, &ret_rl);
1625         set_state(RETURN_ID, "return_ranges", NULL, alloc_estate_rl(ret_rl));
1626         FOR_EACH_PTR(returned_state_callbacks, cb) {
1627                 cb->callback(return_id, (char *)return_ranges, expr);
1628         } END_FOR_EACH_PTR(cb);

1630         if (undo)
1631                 end_assume();

1633         return 1;
1634 }

1636 static int call_return_state_hooks_split_null_non_null_zero(struct expression *e
1633 static int call_return_state_hooks_split_null_non_null(struct expression *expr)
1637 {
1638         struct returned_state_callback *cb;
1639         struct range_list *rl;
1640         struct range_list *nonnull_rl;
1641         sval_t null_sval;
1642         struct range_list *null_rl = NULL;
1643         char *return_ranges;
1644         struct sm_state *sm;
1645         struct smatch_state *state;
1646         int nr_states;
1647         int final_pass_orig = final_pass;

1649         if (!expr || expr_equal_to_param(expr, -1))
1650                 return 0;
1651         if (expr->type == EXPR_CALL)
1652                 return 0;
1650         if (!is_pointer(expr))
1651                 return 0;

1654         sm = get_sm_state_expr(SMATCH_EXTRA, expr);
1655         if (!sm)
1656                 return 0;
1657         if (ptr_list_size((struct ptr_list *)sm->possible) == 1)
1658                 return 0;
1659         state = sm->state;
1660         if (!estate_rl(state))
1661                 return 0;
1662         if (estate_min(state).value == 0 && estate_max(state).value == 0)
1663                 return 0;
1664         if (!has_possible_zero_null(sm))
1665                 return 0;

1667         nr_states = get_db_state_count();
1668         if (option_info && nr_states >= 1500)
1669                 return 0;

1671         rl = estate_rl(state);
```

```
1673         __push_fake_cur_stree();

1675         final_pass = 0;
1676         __split_whole_condition(expr);
1677         final_pass = final_pass_orig;

1679         nonnull_rl = rl_filter(rl, rl_zero());
1680         return_ranges = show_rl(nonnull_rl);
1681         set_state(RETURN_ID, "return_ranges", NULL, alloc_estate_rl(nonnull_rl))

1683         return_id++;
1684         FOR_EACH_PTR(returned_state_callbacks, cb) {
1685                 cb->callback(return_id, return_ranges, expr);
1686         } END_FOR_EACH_PTR(cb);

1688         __push_true_states();
1689         __use_false_states();

1691         return_ranges = alloc_sname("0");
1692         null_sval = sval_type_val(rl_type(rl), 0);
1693         add_range(&null_rl, null_sval, null_sval);
1694         set_state(RETURN_ID, "return_ranges", NULL, alloc_estate_rl(null_rl));
1695         return_id++;
1696         FOR_EACH_PTR(returned_state_callbacks, cb) {
1697                 cb->callback(return_id, return_ranges, expr);
1698         } END_FOR_EACH_PTR(cb);

1700         __merge_true_states();
1701         __free_fake_cur_stree();

1703         return 1;
1704 }
```

*_____unchanged_portion_omitted_*

```
1969 static void call_return_state_hooks(struct expression *expr)
1970 {
1971         struct returned_state_callback *cb;
1972         struct range_list *ret_rl;
1973         const char *return_ranges;
1974         int nr_states;
1975         sval_t sval;

1977         if (__path_is_null())
1978                 return;

1980         expr = strip_expr(expr);
1981         expr = strip_expr_statement(expr);

1983         if (is_impossible_path())
1984                 goto vanilla;

1986         if (expr && (expr->type == EXPR_COMPARE ||
1987                      !get_implied_value(expr, &sval)) &&
1988             (is_condition(expr) || is_boolean(expr))) {
1989                 call_return_state_hooks_compare(expr);
1990                 return;
1991         } else if (is_conditional(expr)) {
1992                 call_return_state_hooks_conditional(expr);
1993                 return;
1994         } else if (call_return_state_hooks_split_possible(expr)) {
1995                 return;
1996         } else if (split_positive_from_negative(expr)) {
1995         } else if (call_return_state_hooks_split_null_non_null(expr)) {
1997                 return;
1998         } else if (call_return_state_hooks_split_null_non_null_zero(expr)) {
1999                 return;
```



```
2000         } else if (call_return_state_hooks_split_success_fail(expr)) {
2001                 return;
2002         } else if (splitable_function_call(expr)) {
2003                 return;
2001         } else if (split_positive_from_negative(expr)) {
2002                 return;
2004         } else if (split_by_bool_param(expr)) {
2005         } else if (split_by_null_nonnull_param(expr)) {
2006                 return;
2007         }

2009 vanilla:
2010         return_ranges = get_return_ranges_str(expr, &ret_rl);
2011         set_state(RETURN_ID, "return_ranges", NULL, alloc_estate_rl(ret_rl));

2013         return_id++;
2014         nr_states = get_db_state_count();
2015         if (nr_states >= 10000) {
2016                 match_return_info(return_id, (char *)return_ranges, expr);
2017                 mark_all_params_untracked(return_id, (char *)return_ranges, expr
2018                 return;
2019         }
2020         FOR_EACH_PTR(returned_state_callbacks, cb) {
2021                 cb->callback(return_id, (char *)return_ranges, expr);
2022         } END_FOR_EACH_PTR(cb);
2023 }
```

*_____unchanged_portion_omitted_*

```
2199 static int save_cache_data(void *_table, int argc, char **argv, char **azColName
2200 {
2201         static char buf[4096];
2202         char tmp[256];
2203         char *p = buf;
2204         char *table = _table;
2205         int i;


2208         p += snprintf(p, 4096 - (p - buf), "insert or ignore into %s values (",
2209         for (i = 0; i < argc; i++) {
2210                 if (i)
2211                         p += snprintf(p, 4096 - (p - buf), ", ");
2212                 sqlite3_snprintf(sizeof(tmp), tmp, "%q", escape_newlines(argv[i]
2211                 sqlite3_snprintf(sizeof(tmp), tmp, "%q", argv[i]);
2213                 p += snprintf(p, 4096 - (p - buf), "'%s'", tmp);

2215         }
2216         p += snprintf(p, 4096 - (p - buf), ");");
2217         if (p - buf > 4096)
2218                 return 0;

2220         sm_msg("SQL: %s", buf);
2221         return 0;
2222 }
```

*_____unchanged_portion_omitted_*

```
2362 void register_definition_db_callbacks(int id)
2363 {
2364         add_hook(&match_call_info, FUNCTION_CALL_HOOK);
2364         add_hook(&global_variable, BASE_HOOK);
2365         add_hook(&global_variable, DECLARATION_HOOK);
2365         add_split_return_callback(match_return_info);
2366         add_split_return_callback(print_returned_struct_members);
2367         add_hook(&call_return_state_hooks, RETURN_HOOK);
2368         add_hook(&match_end_func_info, END_FUNC_HOOK);
2369         add_hook(&match_after_func, AFTER_FUNC_HOOK);
```

```
2371         add_hook(&match_data_from_db, FUNC_DEF_HOOK);
2372         add_hook(&match_call_implies, FUNC_DEF_HOOK);
2373         add_hook(&match_return_implies, CALL_HOOK_AFTER_INLINE);

2375         register_common_funcs();
2376         register_return_replacements();

2378         add_hook(&dump_cache, END_FILE_HOOK);
2379 }
______unchanged_portion_omitted_

2386 char *return_state_to_var_sym(struct expression *expr, int param, const char *ke
2387 {
2388         struct expression *arg;
2389         char *name = NULL;
2390         char member_name[256];

2392         *sym = NULL;

2394         if (param == -1) {
2395                 const char *star = "";

2397                 if (expr->type != EXPR_ASSIGNMENT)
2398                         return NULL;
2399                 if (get_type(expr->left) == &int_ctype && strcmp(key, "$") != 0)
2400                         return NULL;
2401                 name = expr_to_var_sym(expr->left, sym);
2402                 if (!name)
2403                         return NULL;
2404                 if (key[0] == '*') {
2405                         star = "*";
2406                         key++;
2407                 }
2408                 if (strncmp(key, "$", 1) != 0)
2409                         return name;
2410                 snprintf(member_name, sizeof(member_name), "%s%s%s", star, name,
2411                 free_string(name);
2412                 return alloc_string(member_name);
2413         }

2415         while (expr->type == EXPR_ASSIGNMENT)
2416                 expr = strip_expr(expr->right);
2417         if (expr->type != EXPR_CALL)
2418                 return NULL;

2420         arg = get_argument_from_call_expr(expr->args, param);
2421         if (!arg)
2422                 return NULL;

2424         return get_variable_from_key(arg, key, sym);
2425 }

2427 char *get_variable_from_key(struct expression *arg, const char *key, struct symb
2428 {
2429         char buf[256];
2430         char *tmp;
2431         bool add_star = false;

2433         if (!arg)
2434                 return NULL;

2436         arg = strip_expr(arg);

2438         if (strcmp(key, "$") == 0)
2439                 return expr_to_var_sym(arg, sym);
```



```
2441         if (strcmp(key, "*$") == 0) {
2442                 if (arg->type == EXPR_PREOP && arg->op == '&') {
2443                         arg = strip_expr(arg->unop);
2444                         return expr_to_var_sym(arg, sym);
2445                 } else {
2446                         tmp = expr_to_var_sym(arg, sym);
2447                         if (!tmp)
2448                                 return NULL;
2449                         snprintf(buf, sizeof(buf), "*%s", tmp);
2450                         free_string(tmp);
2451                         return alloc_string(buf);
2452                 }
2453         }

2455         if (key[0] == '*') {
2456                 add_star = true;
2457                 key++;
2458         }

2460         if (arg->type == EXPR_PREOP && arg->op == '&') {
2461                 arg = strip_expr(arg->unop);
2462                 tmp = expr_to_var_sym(arg, sym);
2463                 if (!tmp)
2464                         return NULL;
2465                 snprintf(buf, sizeof(buf), "%s%s.%s",
2466                          add_star ? "*" : "", tmp, key + 3);
2458                 snprintf(buf, sizeof(buf), "%s.%s", tmp, key + 3);
2467                 return alloc_string(buf);
2468         }

2470         tmp = expr_to_var_sym(arg, sym);
2471         if (!tmp)
2472                 return NULL;
2473         snprintf(buf, sizeof(buf), "%s%s%s", add_star ? "*" : "", tmp, key + 1);
2465         snprintf(buf, sizeof(buf), "%s%s", tmp, key + 1);
2474         free_string(tmp);
2475         return alloc_string(buf);
2476 }
______unchanged_portion_omitted_

2487 const char *state_name_to_param_name(const char *state_name, const char *param_n
2488 {
2489         int name_len;
2490         static char buf[256];
2491         bool add_star = false;

2493         name_len = strlen(param_name);

2495         if (state_name[0] == '*') {
2496                 add_star = true;
2497                 state_name++;
2498         }

2500         if (strcmp(state_name, param_name) == 0) {
2501                 snprintf(buf, sizeof(buf), "%s$", add_star ? "*" : "");
2502                 return buf;
2503         }

2505         if (state_name[name_len] == '-' && /* check for '-' from "->" */
2487                 return "$";
2488         } else if (state_name[name_len] == '-' && /* check for '-' from "->" */
2506            strncmp(state_name, param_name, name_len) == 0) {
2507                 snprintf(buf, sizeof(buf), "%s$%s",
2508                          add_star ? "*" : "", state_name + name_len);
2490                 snprintf(buf, sizeof(buf), "$%s", state_name + name_len);
2509                 return buf;
```

```
2492         } else if (state_name[0] == '*' && strcmp(state_name + 1, param_name) ==
2493                 return "*$";
2510         }
2511         return NULL;
2512 }
```

*unchanged_portion_omitted_*

```
**********************************************************
    6755 Mon Aug  5 08:38:29 2019
new/usr/src/tools/smatch/src/smatch_equiv.c
11506 smatch resync
**********************************************************
```

```
______unchanged_portion_omitted_

 105 static void debug_addition(struct related_list *rlist, const char *name)
 106 {
 107         struct relation *tmp;

 109         if (!option_debug_related)
 110                 return;

 112         sm_prefix();
 113         sm_printf("(");
 114         FOR_EACH_PTR(rlist, tmp) {
 115                 sm_printf("%s ", tmp->name);
 116         } END_FOR_EACH_PTR(tmp);
 117         sm_printf(") <-- %s\n", name);
 118 }

 105 static void add_related(struct related_list **rlist, const char *name, struct sy
 106 {
 107         struct relation *rel;
 108         struct relation *new;
 109         struct relation tmp = {
 110                 .name = (char *)name,
 111                 .sym = sym
 112         };

 129         debug_addition(*rlist, name);

 114         FOR_EACH_PTR(*rlist, rel) {
 115                 if (cmp_relation(rel, &tmp) < 0)
 116                         continue;
 117                 if (cmp_relation(rel, &tmp) == 0)
 118                         return;
 119                 new = alloc_relation(name, sym);
 120                 INSERT_CURRENT(new, rel);
 121                 return;
 122         } END_FOR_EACH_PTR(rel);
 123         new = alloc_relation(name, sym);
 124         add_ptr_list(rlist, new);
 125 }
______unchanged_portion_omitted_

 191 /*
 192  * set_equiv() is only used for assignments where we set one variable
 193  * equal to the other.  a = b;.  It's not used for if conditions where
 194  * a == b.
 195  */
 196 void set_equiv(struct expression *left, struct expression *right)
 197 {
 198         struct sm_state *right_sm, *left_sm, *other_sm;
 215         struct sm_state *right_sm, *left_sm;
 199         struct relation *rel;
 200         char *left_name;
 201         struct symbol *left_sym;
 202         struct related_list *rlist;
 203         char *other_name;
 204         struct symbol *other_sym;

 206         left_name = expr_to_var_sym(left, &left_sym);
 207         if (!left_name || !left_sym)
 208                 goto free;
```



```
 210         other_name = get_other_name_sym(left_name, left_sym, &other_sym);

 212         right_sm = get_sm_state_expr(SMATCH_EXTRA, right);
 213         if (!right_sm) {
 214                 struct range_list *rl;

 216                 if (!get_implied_rl(right, &rl))
 217                         rl = alloc_whole_rl(get_type(right));
 218                 right_sm = set_state_expr(SMATCH_EXTRA, right, alloc_estate_rl(r
 219         }
 220         if (!right_sm)
 227                 right_sm = set_state_expr(SMATCH_EXTRA, right, alloc_estate_whol
 228         if (!right_sm)
 221                 goto free;

 223         /* This block is because we want to preserve the implications. */
 224         left_sm = clone_sm(right_sm);
 225         left_sm->name = alloc_string(left_name);
 226         left_sm->sym = left_sym;
 227         left_sm->state = clone_estate_cast(get_type(left), right_sm->state);
 228         /* FIXME: The expression we're passing is wrong */
 229         set_extra_mod_helper(left_name, left_sym, left, left_sm->state);
 230         __set_sm(left_sm);

 232         if (other_name && other_sym) {
 233                 other_sm = clone_sm(right_sm);
 234                 other_sm->name = alloc_string(other_name);
 235                 other_sm->sym = other_sym;
 236                 other_sm->state = clone_estate_cast(get_type(left), left_sm->sta
 237                 set_extra_mod_helper(other_name, other_sym, NULL, other_sm->stat
 238                 __set_sm(other_sm);
 239         }

 241         rlist = clone_related_list(estate_related(right_sm->state));
 242         add_related(&rlist, right_sm->name, right_sm->sym);
 243         add_related(&rlist, left_name, left_sym);
 244         if (other_name && other_sym)
 245                 add_related(&rlist, other_name, other_sym);

 247         FOR_EACH_PTR(rlist, rel) {
 248                 struct sm_state *old_sm, *new_sm;

 250                 old_sm = get_sm_state(SMATCH_EXTRA, rel->name, rel->sym);
 251                 if (!old_sm)  /* shouldn't happen */
 252                         continue;
 253                 new_sm = clone_sm(old_sm);
 254                 new_sm->state = clone_estate(old_sm->state);
 255                 get_dinfo(new_sm->state)->related = rlist;
 256                 __set_sm(new_sm);
 257         } END_FOR_EACH_PTR(rel);
 258 free:
 259         free_string(left_name);
 260 }
______unchanged_portion_omitted_
```

```
**********************************************************
    9497 Mon Aug  5 08:38:29 2019
new/usr/src/tools/smatch/src/smatch_estate.c
11506 smatch resync
**********************************************************
```

```
/*
 * 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_rl(s2) || estate_has_hard_max(s
            (estate_has_hard_max(s2) && (!estate_rl(s1) || estate_has_hard_max(s
        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_

        if (estate_capped(s1) && estate_capped(s2))
                estate_set_capped(tmp);

        return tmp;
}
```

*unchanged_portion_omitted_*

```
bool estate_capped(struct smatch_state *state)
```



```
{
        if (!state)
                return false;
        /* impossible states are capped */
        if (!estate_rl(state))
                return true;
        return get_dinfo(state)->capped;
}

void estate_set_capped(struct smatch_state *state)
{
        get_dinfo(state)->capped = true;
}

sval_t estate_min(struct smatch_state *state)
{
        return rl_min(estate_rl(state));
}
```

*unchanged_portion_omitted_*

```
int estates_equiv(struct smatch_state *one, struct smatch_state *two)
{
        if (!one || !two)
                return 0;
        if (one == two)
                return 1;
        if (!rlists_equiv(estate_related(one), estate_related(two)))
                return 0;
        if (estate_capped(one) != estate_capped(two))
                return 0;
        if (strcmp(one->name, two->name) == 0)
                return 1;
        return 0;
}
```

*unchanged_portion_omitted_*

```
struct smatch_state *clone_partial_estate(struct smatch_state *state, struct ran
{
        struct smatch_state *ret;

        if (!state)
                return NULL;

        rl = cast_rl(estate_type(state), rl);

        ret = alloc_estate_rl(rl);
        set_related(ret, clone_related_list(estate_related(state)));
        if (estate_has_hard_max(state))
                estate_set_hard_max(ret);
        if (estate_has_fuzzy_max(state))
                estate_set_fuzzy_max(ret, estate_get_fuzzy_max(state));

        return ret;
}

struct smatch_state *alloc_estate_empty(void)
{
        struct smatch_state *state;
        struct data_info *dinfo;

        dinfo = alloc_dinfo();
        state = __alloc_smatch_state(0);
        state->data = dinfo;
        state->name = "";
        return state;
}
```

*unchanged_portion_omitted_*

```
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;
373
374         if (!orig)
375                 orig = alloc_estate_whole(filter_min.type);
376
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 }
385
386 struct smatch_state *estate_filter_sval(struct smatch_state *orig, sval_t sval)
387 {
388         return estate_filter_range(orig, sval, sval);
389 }
390
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;
398
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 }
```

_____unchanged_portion_omitted_

****
5221 Mon Aug 5 08:38:30 2019
new/usr/src/tools/smatch/src/smatch_expressions.c
11506 smatch resync
****
_____unchanged_portion_omitted_

```
 165 struct expression *gen_expression_from_key(struct expression *arg, const char *k
 166 {
 167         struct expression *ret;
 168         struct token *token, *prev, *end;
 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 || token->special == '.'))
 193                 prev = token;
 192                token->special == SPECIAL_DEREFERENCE) {
 194                 token = token->next;
 195                 if (token_type(token) != TOKEN_IDENT)
 196                         return NULL;
 197                 ret = deref_expression(ret);
 198                 ret = member_expression(ret,
 199                                         (prev->special == SPECIAL_DEREFERENCE) ?
 200                                         token->ident);
 197                 ret = member_expression(ret, '*', token->ident);
 201                 token = token->next;
 202         }

 204         if (token_type(token) != TOKEN_STREAMEND)
 205                 return NULL;

 207         return ret;
 208 }
```

_____unchanged_portion_omitted_

```
**********************************************************
   72935 Mon Aug  5 08:38:30 2019
new/usr/src/tools/smatch/src/smatch_extra.c
11506 smatch resync
**********************************************************
```

```
 1 /*
 2  * Copyright (C) 2008 Dan Carpenter.
 3  *
 4  * This program is free software; you can redistribute it and/or
 5  * modify it under the terms of the GNU General Public License
 6  * as published by the Free Software Foundation; either version 2
 7  * of the License, or (at your option) any later version.
 8  *
 9  * This program is distributed in the hope that it will be useful,
10  * but WITHOUT ANY WARRANTY; without even the implied warranty of
11  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
12  * GNU General Public License for more details.
13  *
14  * You should have received a copy of the GNU General Public License
15  * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
16  */

18 /*
19  * smatch_extra.c is supposed to track the value of every variable.
20  *
21  */

23 #define _GNU_SOURCE
24 #include <string.h>

26 #include <stdlib.h>
27 #include <errno.h>
28 #ifndef __USE_ISOC99
29 #define __USE_ISOC99
30 #endif
31 #include <limits.h>
32 #include "parse.h"
33 #include "smatch.h"
34 #include "smatch_slist.h"
35 #include "smatch_extra.h"

37 static int my_id;
38 static int link_id;
39 extern int check_assigned_expr_id;

41 static void match_link_modify(struct sm_state *sm, struct expression *mod_expr);

43 struct string_list *__ignored_macros = NULL;
44 int in_warn_on_macro(void)
43 static int in_warn_on_macro(void)
45 {
46         struct statement *stmt;
47         char *tmp;
48         char *macro;

50         stmt = get_current_statement();
51         if (!stmt)
52                 return 0;
53         macro = get_macro_name(stmt->pos);
54         if (!macro)
55                 return 0;

57         FOR_EACH_PTR(__ignored_macros, tmp) {
58                 if (!strcmp(tmp, macro))
59                         return 1;
60         } END_FOR_EACH_PTR(tmp);
```



```
61         return 0;
62 }
______unchanged_portion_omitted_

138 char *get_other_name_sym_from_chunk(const char *name, const char *chunk, int len
137 char *get_other_name_sym(const char *name, struct symbol *sym, struct symbol **n
139 {
140         struct expression *assigned;
141         char *orig_name = NULL;
142         char buf[256];
143         char *ret;
142         char *ret = NULL;
143         int skip;

145         assigned = get_assigned_expr_name_sym(chunk, sym);
145         *new_sym = NULL;

147         if (!sym || !sym->ident)
148                 return NULL;

150         ret = get_pointed_at(name, sym, new_sym);
151         if (ret)
152                 return ret;

154         skip = strlen(sym->ident->name);
155         if (name[skip] != '-' || name[skip + 1] != '>')
156                 return NULL;
157         skip += 2;

159         assigned = get_assigned_expr_name_sym(sym->ident->name, sym);
146         if (!assigned)
147                 return NULL;
148         if (assigned->type == EXPR_CALL)
149                 return map_call_to_other_name_sym(name, sym, new_sym);
150         if (assigned->type == EXPR_PREOP && assigned->op == '&') {
164         if (assigned->type == EXPR_PREOP || assigned->op == '&') {

152                 orig_name = expr_to_var_sym(assigned, new_sym);
153                 if (!orig_name || !*new_sym)
154                         goto free;

156                 snprintf(buf, sizeof(buf), "%s.%s", orig_name + 1, name + len);
170                 snprintf(buf, sizeof(buf), "%s.%s", orig_name + 1, name + skip);
157                 ret = alloc_string(buf);
158                 free_string(orig_name);
159                 return ret;
160         }

176         if (assigned->type != EXPR_DEREF)
177                 goto free;

162         orig_name = expr_to_var_sym(assigned, new_sym);
163         if (!orig_name || !*new_sym)
164                 goto free;

166         snprintf(buf, sizeof(buf), "%s->%s", orig_name, name + len);
183         snprintf(buf, sizeof(buf), "%s->%s", orig_name, name + skip);
167         ret = alloc_string(buf);
168         free_string(orig_name);
169         return ret;

170 free:
171         free_string(orig_name);
172         return NULL;
173 }
```

```
175 static char *get_long_name_sym(const char *name, struct symbol *sym, struct symb
176 {
177         struct expression *tmp;
178         struct sm_state *sm;
179         char buf[256];

181         /*
182          * Just prepend the name with a different name/sym and return that.
183          * For example, if we set "foo->bar = bar;" then we clamp "bar->baz",
184          * that also clamps "foo->bar->baz".
185          *
186          */

188         FOR_EACH_MY_SM(check_assigned_expr_id, __get_cur_stree(), sm) {
189                 tmp = sm->state->data;
190                 if (!tmp || tmp->type != EXPR_SYMBOL)
191                         continue;
192                 if (tmp->symbol == sym)
193                         goto found;
194         } END_FOR_EACH_SM(sm);

196         return NULL;

198 found:
199         snprintf(buf, sizeof(buf), "%s%s", sm->name, name + tmp->symbol->ident->
200         *new_sym = sm->sym;
201         return alloc_string(buf);
202 }

204 char *get_other_name_sym_helper(const char *name, struct symbol *sym, struct sym
205 {
206         char buf[256];
207         char *ret;
208         int len;

210         *new_sym = NULL;

212         if (!sym || !sym->ident)
213                 return NULL;

215         ret = get_pointed_at(name, sym, new_sym);
216         if (ret)
217                 return ret;

219         ret = map_long_to_short_name_sym(name, sym, new_sym, use_stack);
220         if (ret)
221                 return ret;

223         len = snprintf(buf, sizeof(buf), "%s", name);
224         if (len >= sizeof(buf) - 2)
225                 return NULL;

227         while (len >= 1) {
228                 if (buf[len] == '>' && buf[len - 1] == '-') {
229                         len--;
230                         buf[len] = '\0';
231                         ret = get_other_name_sym_from_chunk(name, buf, len + 2,
232                         if (ret)
233                                 return ret;
234                 }
235                 len--;
236         }

238         ret = get_long_name_sym(name, sym, new_sym);
239         if (ret)
240                 return ret;
```



```
242         return NULL;
243 }

245 char *get_other_name_sym(const char *name, struct symbol *sym, struct symbol **n
246 {
247         return get_other_name_sym_helper(name, sym, new_sym, true);
248 }

250 char *get_other_name_sym_nostack(const char *name, struct symbol *sym, struct sy
251 {
252         return get_other_name_sym_helper(name, sym, new_sym, false);
253 }

255 void set_extra_mod(const char *name, struct symbol *sym, struct expression *expr
256 {
257         char *new_name;
258         struct symbol *new_sym;

260         set_extra_mod_helper(name, sym, expr, state);
261         new_name = get_other_name_sym(name, sym, &new_sym);
262         if (new_name && new_sym)
263                 set_extra_mod_helper(new_name, new_sym, expr, state);
264         free_string(new_name);
265 }
_____unchanged_portion_omitted_

339 void set_extra_nomod(const char *name, struct symbol *sym, struct expression *ex
340 {
341         char *new_name;
342         struct symbol *new_sym;
343         struct relation *rel;
344         struct smatch_state *orig_state;

346         orig_state = get_state(SMATCH_EXTRA, name, sym);

348         /* don't save unknown states if leaving it blank is the same */
349         if (!orig_state && estate_is_unknown(state))
350                 return;

352         new_name = get_other_name_sym(name, sym, &new_sym);
353         if (new_name && new_sym)
354                 set_extra_nomod_helper(new_name, new_sym, expr, state);
355         free_string(new_name);

357         if (!estate_related(orig_state)) {
358                 set_extra_nomod_helper(name, sym, expr, state);
359                 return;
360         }

362         set_related(state, estate_related(orig_state));
363         FOR_EACH_PTR(estate_related(orig_state), rel) {
364                 struct smatch_state *estate;

304                 if (option_debug_related)
305                         sm_msg("%s updating related %s to %s", name, rel->name,
366                 estate = get_state(SMATCH_EXTRA, rel->name, rel->sym);
367                 if (!estate)
368                         continue;
369                 set_extra_nomod_helper(rel->name, rel->sym, expr, clone_estate_c
370         } END_FOR_EACH_PTR(rel);
371 }
_____unchanged_portion_omitted_

543 static struct sm_state *handle_canonical_while_count_down(struct statement *loop
544 {
```

```
545         struct expression *iter_var;
546         struct expression *condition, *unop;
547         struct symbol *type;
548         struct sm_state *sm;
549         struct smatch_state *estate;
550         int op;
551         sval_t start, right;

553         right.type = &int_ctype;
554         right.value = 0;

556         condition = strip_expr(loop->iterator_pre_condition);
557         if (!condition)
558                 return NULL;

560         if (!get_countdown_info(condition, &unop, &op, &right))
561                 return NULL;

563         iter_var = unop->unop;

565         sm = get_sm_state_expr(SMATCH_EXTRA, iter_var);
566         if (!sm)
567                 return NULL;
568         if (sval_cmp(estate_min(sm->state), right) < 0)
569                 return NULL;
570         start = estate_max(sm->state);

572         type = get_type(iter_var);
573         right = sval_cast(type, right);
574         start = sval_cast(type, start);

576         if  (sval_cmp(start, right) <= 0)
577                 return NULL;
578         if (!sval_is_max(start))
579                 start.value--;

581         if (op == SPECIAL_GTE)
582                 right.value--;

584         if (unop->type == EXPR_PREOP) {
585                 right.value++;
586                 estate = alloc_estate_range(right, start);
587                 if (estate_has_hard_max(sm->state))
588                         estate_set_hard_max(estate);
589                 estate_copy_fuzzy_max(estate, sm->state);
590                 set_extra_expr_mod(iter_var, estate);
591         }
592         if (unop->type == EXPR_POSTOP) {
593                 estate = alloc_estate_range(right, start);
594                 if (estate_has_hard_max(sm->state))
595                         estate_set_hard_max(estate);
596                 estate_copy_fuzzy_max(estate, sm->state);
597                 set_extra_expr_mod(iter_var, estate);
598         }
599         return get_sm_state_expr(SMATCH_EXTRA, iter_var);
600 }

602 static struct sm_state *handle_canonical_for_inc(struct expression *iter_expr,
603                                                  struct expression *condition)
604 {
605         struct expression *iter_var;
606         struct sm_state *sm;
607         struct smatch_state *estate;
608         sval_t start, end, max;
609         struct symbol *type;
```



```
611         iter_var = iter_expr->unop;
612         sm = get_sm_state_expr(SMATCH_EXTRA, iter_var);
613         if (!sm)
614                 return NULL;
615         if (!estate_get_single_value(sm->state, &start))
616                 return NULL;
617         if (!get_implied_value(condition->right, &end))
618                 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));

620         if (get_sm_state_expr(SMATCH_EXTRA, condition->left) != sm)
621                 return NULL;

623         switch (condition->op) {
624         case SPECIAL_UNSIGNED_LT:
625         case SPECIAL_NOTEQUAL:
626         case '<':
627                 if (!sval_is_min(end))
628                         end.value--;
629                 break;
630         case SPECIAL_UNSIGNED_LTE:
631         case SPECIAL_LTE:
632                 break;
633         default:
634                 return NULL;
635         }
636         if (sval_cmp(end, start) < 0)
637                 return NULL;
638         type = get_type(iter_var);
639         start = sval_cast(type, start);
640         end = sval_cast(type, end);
641         estate = alloc_estate_range(start, end);
642         if (get_hard_max(condition->right, &max)) {
643                 if (!get_macro_name(condition->pos))
644                         estate_set_hard_max(estate);
645                 if (condition->op == '<' ||
646                     condition->op == SPECIAL_UNSIGNED_LT ||
647                     condition->op == SPECIAL_NOTEQUAL)
648                         max.value--;
649                 max = sval_cast(type, max);
650                 estate_set_fuzzy_max(estate, max);
651         }
652         set_extra_expr_mod(iter_var, estate);
653         return get_sm_state_expr(SMATCH_EXTRA, iter_var);
654 }

656 static struct sm_state *handle_canonical_for_dec(struct expression *iter_expr,
657                                                  struct expression *condition)
658 {
659         struct expression *iter_var;
660         struct sm_state *sm;
661         struct smatch_state *estate;
662         sval_t start, end;

664         iter_var = iter_expr->unop;
665         sm = get_sm_state_expr(SMATCH_EXTRA, iter_var);
666         if (!sm)
667                 return NULL;
668         if (!estate_get_single_value(sm->state, &start))
669                 return NULL;
670         if (!get_implied_min(condition->right, &end))
671                 end = sval_type_min(get_type(iter_var));
672         end = sval_cast(estate_type(sm->state), end);
```

```
673         if (get_sm_state_expr(SMATCH_EXTRA, condition->left) != sm)
674                 return NULL;

676         switch (condition->op) {
677         case SPECIAL_NOTEQUAL:
678         case '>':
679                 if (!sval_is_max(end))
608                 if (!sval_is_min(end) && !sval_is_max(end))
680                         end.value++;
681                 break;
682         case SPECIAL_GTE:
683                 break;
684         default:
685                 return NULL;
686         }
687         if (sval_cmp(end, start) > 0)
688                 return NULL;
689         estate = alloc_estate_range(end, start);
690         estate_set_hard_max(estate);
691         estate_set_fuzzy_max(estate, estate_get_fuzzy_max(estate));
692         set_extra_expr_mod(iter_var, estate);
693         return get_sm_state_expr(SMATCH_EXTRA, iter_var);
694 }
______unchanged_portion_omitted_

764 void __extra_pre_loop_hook_after(struct sm_state *sm,
765                                  struct statement *iterator,
766                                  struct expression *condition)
767 {
768         struct expression *iter_expr;
769         sval_t limit;
770         struct smatch_state *state;

772         if (!iterator) {
773                 while_count_down_after(sm, condition);
774                 return;
775         }

777         iter_expr = iterator->expression;

779         if (condition->type != EXPR_COMPARE)
780                 return;
781         if (iter_expr->op == SPECIAL_INCREMENT) {
782                 limit = sval_binop(estate_max(sm->state), '+',
783                                    sval_type_val(estate_type(sm->state), 1));
784         } else {
785                 limit = sval_binop(estate_min(sm->state), '-',
786                                    sval_type_val(estate_type(sm->state), 1));
787         }
788         limit = sval_cast(estate_type(sm->state), limit);
789         if (!estate_has_hard_max(sm->state) && !__has_breaks()) {
790                 if (iter_expr->op == SPECIAL_INCREMENT)
791                         state = alloc_estate_range(estate_min(sm->state), limit)
792                 else
793                         state = alloc_estate_range(limit, estate_max(sm->state))
794         } else {
795                 state = alloc_estate_sval(limit);
796         }
797         if (!estate_has_hard_max(sm->state)) {
798                 estate_clear_hard_max(state);
799         }
800         if (estate_has_fuzzy_max(sm->state)) {
801                 sval_t hmax = estate_get_fuzzy_max(sm->state);
802                 sval_t max = estate_max(sm->state);

804                 if (sval_cmp(hmax, max) != 0)
```



```
805                         estate_clear_fuzzy_max(state);
806         } else if (!estate_has_fuzzy_max(sm->state)) {
807                 estate_clear_fuzzy_max(state);
808         }

810         set_extra_mod(sm->name, sm->sym, iter_expr, state);
811 }

813 static bool get_global_rl(const char *name, struct symbol *sym, struct range_lis
814 {
815         struct expression *expr;

817         if (!sym || !(sym->ctype.modifiers & MOD_TOPLEVEL) || !sym->ident)
818                 return false;
819         if (strcmp(sym->ident->name, name) != 0)
820                 return false;

822         expr = symbol_expression(sym);
823         return get_implied_rl(expr, rl);
824 }

826 static struct stree *unmatched_stree;
827 static struct smatch_state *unmatched_state(struct sm_state *sm)
828 {
829         struct smatch_state *state;
830         struct range_list *rl;

832         if (unmatched_stree) {
833                 state = get_state_stree(unmatched_stree, SMATCH_EXTRA, sm->name,
834                 if (state)
835                         return state;
836         }
837         if (parent_is_gone_var_sym(sm->name, sm->sym))
838                 return alloc_estate_empty();
839         if (get_global_rl(sm->name, sm->sym, &rl))
840                 return alloc_estate_rl(rl);
841         return alloc_estate_whole(estate_type(sm->state));
842 }
______unchanged_portion_omitted_

906 int values_fit_type(struct expression *left, struct expression *right)
818 static int values_fit_type(struct expression *left, struct expression *right)
907 {
908         struct range_list *rl;
909         struct symbol *type;

911         type = get_type(left);
912         if (!type)
913                 return 0;
914         get_absolute_rl(right, &rl);
915         if (type == rl_type(rl))
916                 return 1;
917         if (type_unsigned(type) && sval_is_negative(rl_min(rl)))
918                 return 0;
919         if (sval_cmp(sval_type_min(type), rl_min(rl)) > 0)
920                 return 0;
921         if (sval_cmp(sval_type_max(type), rl_max(rl)) < 0)
922                 return 0;
923         return 1;
924 }
______unchanged_portion_omitted_

973 static void match_vanilla_assign(struct expression *left, struct expression *rig
974 {
975         struct range_list *orig_rl = NULL;
976         struct range_list *rl = NULL;
```

```
 977         struct symbol *right_sym;
 978         struct symbol *left_type;
 979         struct symbol *right_type;
 980         char *right_name = NULL;
 981         struct symbol *sym;
 982         char *name;
 983         sval_t sval, max;
 984         struct smatch_state *state;
 985         int comparison;

 987         if (is_struct(left))
 988                 return;

 990         save_chunk_info(left, right);

 992         name = expr_to_var_sym(left, &sym);
 993         if (!name) {
 994                 if (chunk_has_array(left))
 995                         do_array_assign(left, '=', right);
 996                 return;
 997         }

 999         left_type = get_type(left);
1000         right_type = get_type(right);

1002         right_name = expr_to_var_sym(right, &right_sym);

1004         if (!__in_fake_assign &&
1005             !(right->type == EXPR_PREOP && right->op == '&') &&
1006             right_name && right_sym &&
1007             values_fit_type(left, strip_expr(right)) &&
1008             !has_symbol(right, sym)) {
1009                 set_equiv(left, right);
1010                 goto free;
1011         }

1013         if (is_pointer(right) && get_address_rl(right, &rl)) {
1014                 state = alloc_estate_rl(rl);
1015                 goto done;
1016         }

1018         if (get_implied_value(right, &sval)) {
1019                 state = alloc_estate_sval(sval_cast(left_type, sval));
1020                 goto done;
1021         }

1023         if (__in_fake_assign) {
1024                 struct smatch_state *right_state;
1025                 sval_t sval;

1027                 if (get_value(right, &sval)) {
1028                         sval = sval_cast(left_type, sval);
1029                         state = alloc_estate_sval(sval);
1030                         goto done;
1031                 }

1033                 right_state = get_state(SMATCH_EXTRA, right_name, right_sym);
1034                 if (right_state) {
1035                         /* simple assignment */
1036                         state = clone_estate(right_state);
1037                         goto done;
1038                 }

1040                 state = alloc_estate_rl(alloc_whole_rl(left_type));
1041                 goto done;
1042         }
```



```
1044         comparison = get_comparison_no_extra(left, right);
 954         comparison = get_comparison(left, right);
1045         if (comparison) {
1046                 comparison = flip_comparison(comparison);
1047                 get_implied_rl(left, &orig_rl);
1048         }

1050         if (get_implied_rl(right, &rl)) {
1051                 rl = cast_rl(left_type, rl);
1052                 if (orig_rl)
1053                         filter_by_comparison(&rl, comparison, orig_rl);
1054                 state = alloc_estate_rl(rl);
1055                 if (get_hard_max(right, &max)) {
1056                         estate_set_hard_max(state);
1057                         estate_set_fuzzy_max(state, max);
1058                 }
1059         } else {
1060                 rl = alloc_whole_rl(right_type);
1061                 rl = cast_rl(left_type, rl);
1062                 if (orig_rl)
1063                         filter_by_comparison(&rl, comparison, orig_rl);
1064                 state = alloc_estate_rl(rl);
1065         }

1067 done:
1068         set_extra_mod(name, sym, left, state);
1069 free:
1070         free_string(right_name);
1071 }
```

```
 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 }
```

```
1073 static void match_assign(struct expression *expr)
1074 {
1075         struct range_list *rl = NULL;
1076         struct expression *left;
1077         struct expression *right;
1078         struct expression *binop_expr;
1079         struct symbol *left_type;
```

```
1080         struct symbol *sym;
1081         char *name;
1020         sval_t left_min, left_max;
1021         sval_t right_min, right_max;
1022         sval_t res_min, res_max;

1083         left = strip_expr(expr->left);

1085         right = strip_parens(expr->right);
1086         if (right->type == EXPR_CALL && sym_name_is("__builtin_expect", right->f
1087                 right = get_argument_from_call_expr(right->args, 0);
1088         while (right->type == EXPR_ASSIGNMENT && right->op == '=')
1089                 right = strip_parens(right->left);

1091         if (expr->op == '=' && is_condition(expr->right))
1092                 return; /* handled in smatch_condition.c */
1093         if (expr->op == '=' && right->type == EXPR_CALL)
1094                 return; /* handled in smatch_function_hooks.c */
1095         if (expr->op == '=') {
1096                 match_vanilla_assign(left, right);
1097                 return;
1098         }

1100         name = expr_to_var_sym(left, &sym);
1101         if (!name)
1102                 return;

1104         left_type = get_type(left);

1047         res_min = sval_type_min(left_type);
1048         res_max = sval_type_max(left_type);

1106         switch (expr->op) {
1107         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;
1108         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;
1109         case SPECIAL_AND_ASSIGN:
1110         case SPECIAL_MOD_ASSIGN:
1111         case SPECIAL_SHL_ASSIGN:
1112         case SPECIAL_SHR_ASSIGN:
1113         case SPECIAL_OR_ASSIGN:
1114         case SPECIAL_XOR_ASSIGN:
1115         case SPECIAL_MUL_ASSIGN:
1116         case SPECIAL_DIV_ASSIGN:
1117                 binop_expr = binop_expression(expr->left,
1118                                               op_remove_assign(expr->op),
1119                                               expr->right);
1120                 get_absolute_rl(binop_expr, &rl);
1097                 if (get_absolute_rl(binop_expr, &rl)) {
1121                 rl = cast_rl(left_type, rl);
1122                 if (inside_loop()) {
1123                         if (expr->op == SPECIAL_ADD_ASSIGN)
1124                                 add_range(&rl, rl_max(rl), sval_type_max(rl_type

1126                         if (expr->op == SPECIAL_SUB_ASSIGN &&
1127                             !sval_is_negative(rl_min(rl))) {
1128                                 sval_t zero = { .type = rl_type(rl) };

1130                                 add_range(&rl, rl_min(rl), zero);
1131                         }
1132                 }
1133                 set_extra_mod(name, sym, left, alloc_estate_rl(rl));
1134                 goto free;
1135         }
1136         set_extra_mod(name, sym, left, alloc_estate_whole(left_type));
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));
1137 free:
1138         free_string(name);
1139 }
```

_____***unchanged_portion_omitted_***

```
1251 static void check_dereference(struct expression *expr)
1252 {
1253         struct smatch_state *state;

1255         if (__in_fake_assign)
1256                 return;
1257         if (outside_of_function())
1258                 return;
1259         state = get_extra_state(expr);
1260         if (state) {
1261                 struct range_list *rl;

1263                 rl = rl_intersection(estate_rl(state), valid_ptr_rl);
1264                 if (rl_equiv(rl, estate_rl(state)))
1265                         return;
1266                 set_extra_expr_nomod(expr, alloc_estate_rl(rl));
1267         } else {
1268                 struct range_list *rl;

1270                 if (get_mtag_rl(expr, &rl))
1271                         rl = rl_intersection(rl, valid_ptr_rl);
1272                 else
1273                         rl = clone_rl(valid_ptr_rl);

1275                 set_extra_expr_nomod(expr, alloc_estate_rl(rl));
1237                 set_extra_expr_nomod(expr, alloc_estate_range(valid_ptr_min_sval
```

```
1276         }
1277 }
```

*unchanged_portion_omitted_*

```
1338 static int handle_postop_inc(struct expression *left, int op, struct expression
1339 {
1340         struct statement *stmt;
1341         struct expression *cond;
1342         struct smatch_state *true_state, *false_state;
1343         struct symbol *type;
1344         sval_t start;
1345         sval_t limit;

1347         /*
1348          * If we're decrementing here then that's a canonical while count down
1349          * so it's handled already.  We're only handling loops like:
1350          * i = 0;
1351          * do { ... } while (i++ < 3);
1352          */

1354         if (left->type != EXPR_POSTOP || left->op != SPECIAL_INCREMENT)
1355                 return 0;

1357         stmt = __cur_stmt->parent;
1358         if (!stmt)
1359                 return 0;
1360         if (stmt->type == STMT_COMPOUND)
1361                 stmt = stmt->parent;
1362         if (!stmt || stmt->type != STMT_ITERATOR || !stmt->iterator_post_conditi
1363                 return 0;

1365         cond = strip_expr(stmt->iterator_post_condition);
1366         if (cond->type != EXPR_COMPARE || cond->op != op)
1367                 return 0;
1368         if (left != strip_expr(cond->left) || right != strip_expr(cond->right))
1369                 return 0;

1371         if (!get_implied_value(left->unop, &start))
1372                 return 0;
1373         if (!get_implied_value(right, &limit))
1374                 return 0;
1375         type = get_type(left->unop);
1376         limit = sval_cast(type, limit);

1377         if (sval_cmp(start, limit) > 0)
1378                 return 0;

1380         switch (op) {
1381         case '<':
1382         case SPECIAL_UNSIGNED_LT:
1383                 break;
1384         case SPECIAL_LTE:
1385         case SPECIAL_UNSIGNED_LTE:
1386                 limit = add_one(limit);
1387         default:
1388                 return 0;

1390         }

1392         true_state = alloc_estate_range(add_one(start), limit);
1393         false_state = alloc_estate_range(add_one(limit), add_one(limit));

1395         /* Currently we just discard the false state but when two passes is
1396          * implimented correctly then it will use it.
1397          */
```



```
1399         set_extra_expr_true_false(left->unop, true_state, false_state);

1401         return 1;
1402 }
```

*unchanged_portion_omitted_*

```
1414 static bool in_macro(struct expression *left, struct expression *right)
1415 {
1416         if (!left || !right)
1417                 return 0;
1418         if (left->pos.line != right->pos.line || left->pos.pos != right->pos.pos
1419                 return 0;
1420         if (get_macro_name(left->pos))
1421                 return 1;
1422         return 0;
1423 }

1425 static void handle_comparison(struct symbol *type, struct expression *left, int
1426 {
1427         struct range_list *left_orig;
1428         struct range_list *left_true;
1429         struct range_list *left_false;
1430         struct range_list *right_orig;
1431         struct range_list *right_true;
1432         struct range_list *right_false;
1433         struct smatch_state *left_true_state;
1434         struct smatch_state *left_false_state;
1435         struct smatch_state *right_true_state;
1436         struct smatch_state *right_false_state;
1437         sval_t dummy, hard_max;
1438         int left_postop = 0;
1439         int right_postop = 0;

1441         if (left->op == SPECIAL_INCREMENT || left->op == SPECIAL_DECREMENT) {
1442                 if (left->type == EXPR_POSTOP) {
1443                         left->smatch_flags |= Handled;
1444                         left_postop = left->op;
1445                         if (handle_postop_inc(left, op, right))
1446                                 return;
1447                 }
1448                 left = strip_parens(left->unop);
1449         }
1450         while (left->type == EXPR_ASSIGNMENT)
1451                 left = strip_parens(left->left);

1453         if (right->op == SPECIAL_INCREMENT || right->op == SPECIAL_DECREMENT) {
1454                 if (right->type == EXPR_POSTOP) {
1455                         right->smatch_flags |= Handled;
1456                         right_postop = right->op;
1457                 }
1458                 right = strip_parens(right->unop);
1459         }

1461         if (is_impossible_variable(left) || is_impossible_variable(right))
1462                 return;

1464         get_real_absolute_rl(left, &left_orig);
1465         left_orig = cast_rl(type, left_orig);

1467         get_real_absolute_rl(right, &right_orig);
1468         right_orig = cast_rl(type, right_orig);

1470         split_comparison_rl(left_orig, op, right_orig, &left_true, &left_false,

1472         left_true = rl_truncate_cast(get_type(strip_expr(left)), left_true);
1473         left_false = rl_truncate_cast(get_type(strip_expr(left)), left_false);
```

```
1474         right_true = rl_truncate_cast(get_type(strip_expr(right)), right_true);
1475         right_false = rl_truncate_cast(get_type(strip_expr(right)), right_false)

1477         if (!left_true || !left_false) {
1478                 struct range_list *tmp_true, *tmp_false;

1480                 split_comparison_rl(alloc_whole_rl(type), op, right_orig, &tmp_t
1481                 tmp_true = rl_truncate_cast(get_type(strip_expr(left)), tmp_true
1482                 tmp_false = rl_truncate_cast(get_type(strip_expr(left)), tmp_fal
1483                 if (tmp_true && tmp_false)
1484                         __save_imaginary_state(left, tmp_true, tmp_false);
1485         }

1487         if (!right_true || !right_false) {
1488                 struct range_list *tmp_true, *tmp_false;

1490                 split_comparison_rl(alloc_whole_rl(type), op, right_orig, NULL,
1491                 tmp_true = rl_truncate_cast(get_type(strip_expr(right)), tmp_tru
1492                 tmp_false = rl_truncate_cast(get_type(strip_expr(right)), tmp_fa
1493                 if (tmp_true && tmp_false)
1494                         __save_imaginary_state(right, tmp_true, tmp_false);
1495         }

1497         left_true_state = alloc_estate_rl(left_true);
1498         left_false_state = alloc_estate_rl(left_false);
1499         right_true_state = alloc_estate_rl(right_true);
1500         right_false_state = alloc_estate_rl(right_false);

1502         switch (op) {
1503         case '<':
1504         case SPECIAL_UNSIGNED_LT:
1505         case SPECIAL_UNSIGNED_LTE:
1506         case SPECIAL_LTE:
1507                 if (get_implied_value(right, &dummy) && !in_macro(left, right))
1456                 if (get_hard_max(right, &dummy))
1508                         estate_set_hard_max(left_true_state);
1509                 if (get_implied_value(left, &dummy) && !in_macro(left, right))
1458                 if (get_hard_max(left, &dummy))
1510                         estate_set_hard_max(right_false_state);
1511                 break;
1512         case '>':
1513         case SPECIAL_UNSIGNED_GT:
1514         case SPECIAL_UNSIGNED_GTE:
1515         case SPECIAL_GTE:
1516                 if (get_implied_value(left, &dummy) && !in_macro(left, right))
1465                 if (get_hard_max(left, &dummy))
1517                         estate_set_hard_max(right_true_state);
1518                 if (get_implied_value(right, &dummy) && !in_macro(left, right))
1467                 if (get_hard_max(right, &dummy))
1519                         estate_set_hard_max(left_false_state);
1520                 break;
1521         }

1523         switch (op) {
1524         case '<':
1525         case SPECIAL_UNSIGNED_LT:
1526         case SPECIAL_UNSIGNED_LTE:
1527         case SPECIAL_LTE:
1528                 if (get_hard_max(right, &hard_max)) {
1529                         if (op == '<' || op == SPECIAL_UNSIGNED_LT)
1530                                 hard_max.value--;
1531                         estate_set_fuzzy_max(left_true_state, hard_max);
1532                 }
1533                 if (get_implied_value(right, &hard_max)) {
1534                         if (op == SPECIAL_UNSIGNED_LTE ||
1535                             op == SPECIAL_LTE)
```



```
1536                                 hard_max.value++;
1537                         estate_set_fuzzy_max(left_false_state, hard_max);
1538                 }
1539                 if (get_hard_max(left, &hard_max)) {
1540                         if (op == SPECIAL_UNSIGNED_LTE ||
1541                             op == SPECIAL_LTE)
1542                                 hard_max.value--;
1543                         estate_set_fuzzy_max(right_false_state, hard_max);
1544                 }
1545                 if (get_implied_value(left, &hard_max)) {
1546                         if (op == '<' || op == SPECIAL_UNSIGNED_LT)
1547                                 hard_max.value++;
1548                         estate_set_fuzzy_max(right_true_state, hard_max);
1549                 }
1550                 break;
1551         case '>':
1552         case SPECIAL_UNSIGNED_GT:
1553         case SPECIAL_UNSIGNED_GTE:
1554         case SPECIAL_GTE:
1555                 if (get_hard_max(left, &hard_max)) {
1556                         if (op == '>' || op == SPECIAL_UNSIGNED_GT)
1557                                 hard_max.value--;
1558                         estate_set_fuzzy_max(right_true_state, hard_max);
1559                 }
1560                 if (get_implied_value(left, &hard_max)) {
1561                         if (op == SPECIAL_UNSIGNED_GTE ||
1562                             op == SPECIAL_GTE)
1563                                 hard_max.value++;
1564                         estate_set_fuzzy_max(right_false_state, hard_max);
1565                 }
1566                 if (get_hard_max(right, &hard_max)) {
1567                         if (op == SPECIAL_UNSIGNED_LTE ||
1568                             op == SPECIAL_LTE)
1569                                 hard_max.value--;
1570                         estate_set_fuzzy_max(left_false_state, hard_max);
1571                 }
1572                 if (get_implied_value(right, &hard_max)) {
1573                         if (op == '>' ||
1574                             op == SPECIAL_UNSIGNED_GT)
1575                                 hard_max.value++;
1576                         estate_set_fuzzy_max(left_true_state, hard_max);
1577                 }
1578                 break;
1579         case SPECIAL_EQUAL:
1580                 if (get_hard_max(left, &hard_max))
1581                         estate_set_fuzzy_max(right_true_state, hard_max);
1582                 if (get_hard_max(right, &hard_max))
1583                         estate_set_fuzzy_max(left_true_state, hard_max);
1584                 break;
1585         }

1587         if (get_hard_max(left, &hard_max)) {
1588                 estate_set_hard_max(left_true_state);
1589                 estate_set_hard_max(left_false_state);
1590         }
1591         if (get_hard_max(right, &hard_max)) {
1592                 estate_set_hard_max(right_true_state);
1593                 estate_set_hard_max(right_false_state);
1594         }

1596         if (left_postop == SPECIAL_INCREMENT) {
1597                 left_true_state = increment_state(left_true_state);
1598                 left_false_state = increment_state(left_false_state);
1599         }
1600         if (left_postop == SPECIAL_DECREMENT) {
1601                 left_true_state = decrement_state(left_true_state);
```

```
1602                    left_false_state = decrement_state(left_false_state);
1603            }
1604            if (right_postop == SPECIAL_INCREMENT) {
1605                    right_true_state = increment_state(right_true_state);
1606                    right_false_state = increment_state(right_false_state);
1607            }
1608            if (right_postop == SPECIAL_DECREMENT) {
1609                    right_true_state = decrement_state(right_true_state);
1610                    right_false_state = decrement_state(right_false_state);
1611            }

1613            if (estate_rl(left_true_state) && estates_equiv(left_true_state, left_fa
1614                    left_true_state = NULL;
1615                    left_false_state = NULL;
1616            }

1618            if (estate_rl(right_true_state) && estates_equiv(right_true_state, right
1619                    right_true_state = NULL;
1620                    right_false_state = NULL;
1621            }

1623            /* Don't introduce new states for known true/false conditions */
1624            if (rl_equiv(left_orig, estate_rl(left_true_state)))
1625                    left_true_state = NULL;
1626            if (rl_equiv(left_orig, estate_rl(left_false_state)))
1627                    left_false_state = NULL;
1628            if (rl_equiv(right_orig, estate_rl(right_true_state)))
1629                    right_true_state = NULL;
1630            if (rl_equiv(right_orig, estate_rl(right_false_state)))
1631                    right_false_state = NULL;

1633            set_extra_expr_true_false(left, left_true_state, left_false_state);
1634            set_extra_expr_true_false(right, right_true_state, right_false_state);
1635    }
```

*unchanged_portion_omitted_*

```
1652    static int flip_op(int op)
1653    {
1654            /* We only care about simple math */
1655            switch (op) {
1656            case '+':
1657                    return '-';
1658            case '-':
1659                    return '+';
1660            case '*':
1661                    return '/';
1662            }
1663            return 0;
1664    }

1666    static void move_known_to_rl(struct expression **expr_p, struct range_list **rl_
1667    {
1668            struct expression *expr = *expr_p;
1669            struct range_list *rl = *rl_p;
1670            sval_t sval;

1672            if (!is_simple_math(expr))
1673                    return;

1675            if (get_implied_value(expr->right, &sval)) {
1676                    *expr_p = expr->left;
1677                    *rl_p = rl_binop(rl, flip_op(expr->op), alloc_rl(sval, sval));
1678                    move_known_to_rl(expr_p, rl_p);
1679                    return;
1680            }
1681            if (expr->op == '-')
```



```
1682                    return;
1683            if (get_implied_value(expr->left, &sval)) {
1684                    *expr_p = expr->right;
1685                    *rl_p = rl_binop(rl, flip_op(expr->op), alloc_rl(sval, sval));
1686                    move_known_to_rl(expr_p, rl_p);
1687                    return;
1688            }
1689    }

1691    static void move_known_values(struct expression **left_p, struct expression **ri
1692    {
1693            struct expression *left = *left_p;
1694            struct expression *right = *right_p;
1695            sval_t sval, dummy;

1697            if (get_implied_value(left, &sval)) {
1698                    if (!is_simple_math(right))
1699                            return;
1700                    if (get_implied_value(right, &dummy))
1701                            return;
1702                    if (right->op == '*') {
1703                            sval_t divisor;

1705                            if (!get_value(right->right, &divisor))
1706                                    return;
1707                            if (divisor.value == 0)
1708                                    return;
1709                            *left_p = binop_expression(left, invert_op(right->op), r
1710                            *right_p = right->left;
1711                            return;
1712                    }
1713                    if (right->op == '+' && get_value(right->left, &sval)) {
1714                            *left_p = binop_expression(left, invert_op(right->op), r
1715                            *right_p = right->right;
1716                            return;
1717                    }
1718                    if (get_value(right->right, &sval)) {
1719                            *left_p = binop_expression(left, invert_op(right->op), r
1720                            *right_p = right->left;
1721                            return;
1722                    }
1723                    return;
1724            }
1725            if (get_implied_value(right, &sval)) {
1726                    if (!is_simple_math(left))
1727                            return;
1728                    if (get_implied_value(left, &dummy))
1729                            return;
1730                    if (left->op == '*') {
1731                            sval_t divisor;

1733                            if (!get_value(left->right, &divisor))
1734                                    return;
1735                            if (divisor.value == 0)
1736                                    return;
1737                            *right_p = binop_expression(right, invert_op(left->op),
1738                            *left_p = left->left;
1739                            return;
1740                    }
1741                    if (left->op == '+' && get_value(left->left, &sval)) {
1742                            *right_p = binop_expression(right, invert_op(left->op),
1743                            *left_p = left->right;
1744                            return;
1745                    }

1747                    if (get_value(left->right, &sval)) {
```

```
1748                        *right_p = binop_expression(right, invert_op(left->op),
1749                        *left_p = left->left;
1750                        return;
1751                }
1752                return;
1753        }
1754 }
```

*unchanged_portion_omitted_*

```
1795 static int match_func_comparison(struct expression *expr)
1796 {
1797         struct expression *left = strip_expr(expr->left);
1798         struct expression *right = strip_expr(expr->right);
1709         sval_t sval;

1800         if (left->type == EXPR_CALL || right->type == EXPR_CALL) {
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          */

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                 }
1801                 function_comparison(left, expr->op, right);
1802                 return 1;
1803         }

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         }

1805         return 0;
1806 }

1808 /* Handle conditions like "if (foo + bar < foo) {" */
1809 static int handle_integer_overflow_test(struct expression *expr)
1810 {
1811         struct expression *left, *right;
1812         struct symbol *type;
1813         sval_t left_min, right_min, min, max;

1815         if (expr->op != '<' && expr->op != SPECIAL_UNSIGNED_LT)
1816                 return 0;

1818         left = strip_parens(expr->left);
1819         right = strip_parens(expr->right);

1821         if (left->op != '+')
1822                 return 0;

1824         type = get_type(expr);
1825         if (!type)
1826                 return 0;
1827         if (type_positive_bits(type) == 32) {
1828                 max.type = &uint_ctype;
1829                 max.uvalue = (unsigned int)-1;
```



```
1830         } else if (type_positive_bits(type) == 64) {
1831                 max.type = &ulong_ctype;
1832                 max.value = (unsigned long long)-1;
1833         } else {
1834                 return 0;
1835         }

1837         if (!expr_equiv(left->left, right) && !expr_equiv(left->right, right))
1838                 return 0;

1840         get_absolute_min(left->left, &left_min);
1841         get_absolute_min(left->right, &right_min);
1842         min = sval_binop(left_min, '+', right_min);

1844         type = get_type(left);
1845         min = sval_cast(type, min);
1846         max = sval_cast(type, max);

1848         set_extra_chunk_true_false(left, NULL, alloc_estate_range(min, max));
1849         return 1;
1850 }
```

*unchanged_portion_omitted_*

```
1946 static bool handle_bit_test(struct expression *expr)
1947 {
1948         struct range_list *orig_rl, *rl;
1949         struct expression *shift, *mask, *var;
1950         struct bit_info *bit_info;
1951         sval_t sval;
1952         sval_t high = { .type = &int_ctype };
1953         sval_t low = { .type = &int_ctype };

1955         shift = strip_expr(expr->right);
1956         mask = strip_expr(expr->left);
1957         if (shift->type != EXPR_BINOP || shift->op != SPECIAL_LEFTSHIFT) {
1958                 shift = strip_expr(expr->left);
1959                 mask = strip_expr(expr->right);
1960                 if (shift->type != EXPR_BINOP || shift->op != SPECIAL_LEFTSHIFT)
1961                         return false;
1962         }
1963         if (!get_implied_value(shift->left, &sval) || sval.value != 1)
1964                 return false;
1965         var = strip_expr(shift->right);

1967         bit_info = get_bit_info(mask);
1968         if (!bit_info)
1969                 return false;
1970         if (!bit_info->possible)
1971                 return false;

1973         get_absolute_rl(var, &orig_rl);
1974         if (sval_is_negative(rl_min(orig_rl)) ||
1975             rl_max(orig_rl).uvalue > type_bits(get_type(shift->left)))
1976                 return false;

1978         low.value = ffsll(bit_info->possible);
1979         high.value = sm_fls64(bit_info->possible);
1980         rl = alloc_rl(low, high);
1981         rl = cast_rl(get_type(var), rl);
1982         rl = rl_intersection(orig_rl, rl);
1983         if (!rl)
1984                 return false;

1986         set_extra_expr_true_false(shift->right, alloc_estate_rl(rl), NULL);

1988         return true;
```

```
1989 }

1991 static void handle_AND_op(struct expression *var, sval_t known)
1992 {
1993         struct range_list *orig_rl;
1994         struct range_list *true_rl = NULL;
1995         struct range_list *false_rl = NULL;
1996         int bit;
1997         sval_t low_mask = known;
1998         sval_t high_mask;
1999         sval_t max;

2001         get_absolute_rl(var, &orig_rl);

2003         if (known.value > 0) {
2004                 bit = ffsll(known.value) - 1;
2005                 low_mask.uvalue = (1ULL << bit) - 1;
2006                 true_rl = remove_range(orig_rl, sval_type_val(known.type, 0), lo
2007         }
2008         high_mask = get_high_mask(known);
2009         if (high_mask.value) {
2010                 bit = ffsll(high_mask.value) - 1;
2011                 low_mask.uvalue = (1ULL << bit) - 1;

2013                 false_rl = orig_rl;
2014                 if (sval_is_negative(rl_min(orig_rl)))
2015                         false_rl = remove_range(false_rl, sval_type_min(known.ty
2016                 false_rl = remove_range(false_rl, low_mask, sval_type_max(known.
2017                 if (type_signed(high_mask.type) && type_unsigned(rl_type(false_r
2018                         false_rl = remove_range(false_rl,
2019                                                 sval_type_val(rl_type(false_rl),
2020                                         sval_type_val(rl_type(false_rl), -1));
2021                 }
2022         } else if (known.value == 1 &&
2023                    get_hard_max(var, &max) &&
2024                    sval_cmp(max, rl_max(orig_rl)) == 0 &&
2025                    max.value & 1) {
2026                 false_rl = remove_range(orig_rl, max, max);
2027         }
2028         set_extra_expr_true_false(var,
2029                                   true_rl ? alloc_estate_rl(true_rl) : NULL,
2030                                   false_rl ? alloc_estate_rl(false_rl) : NULL);
2031 }

2033 static void handle_AND_condition(struct expression *expr)
2034 {
2035         sval_t known;

2037         if (handle_bit_test(expr))
2038                 return;

2040         if (get_implied_value(expr->left, &known))
2041                 handle_AND_op(expr->right, known);
2042         else if (get_implied_value(expr->right, &known))
2043                 handle_AND_op(expr->left, known);
2044 }

2046 static void handle_MOD_condition(struct expression *expr)
2047 {
2048         struct range_list *orig_rl;
2049         struct range_list *true_rl;
2050         struct range_list *false_rl = NULL;
2051         sval_t right;
2052         sval_t zero = { 0, };
1931         sval_t zero;
```



```
2054         if (!get_implied_value(expr->right, &right) || right.value == 0)
2055                 return;
2056         get_absolute_rl(expr->left, &orig_rl);

2058         zero.value = 0;
2059         zero.type = rl_type(orig_rl);

2061         /* We're basically dorking around the min and max here */
2062         true_rl = remove_range(orig_rl, zero, zero);
2063         if (!sval_is_max(rl_max(true_rl)) &&
2064             !(rl_max(true_rl).value % right.value))
2065                 true_rl = remove_range(true_rl, rl_max(true_rl), rl_max(true_rl)

2067         if (rl_equiv(true_rl, orig_rl))
2068                 true_rl = NULL;

2070         if (sval_is_positive(rl_min(orig_rl)) &&
2071             (rl_max(orig_rl).value - rl_min(orig_rl).value) / right.value < 5) {
2072                 sval_t add;
2073                 int i;

2075                 add = rl_min(orig_rl);
2076                 add.value += right.value - (add.value % right.value);
2077                 add.value -= right.value;

2079                 for (i = 0; i < 5; i++) {
2080                         add.value += right.value;
2081                         if (add.value > rl_max(orig_rl).value)
2082                                 break;
2083                         add_range(&false_rl, add, add);
2084                 }
2085         } else {
2086                 if (rl_min(orig_rl).uvalue != 0 &&
2087                     rl_min(orig_rl).uvalue < right.uvalue) {
2088                         sval_t chop = right;
2089                         chop.value--;
2090                         false_rl = remove_range(orig_rl, zero, chop);
2091                 }

2093                 if (!sval_is_max(rl_max(orig_rl)) &&
2094                     (rl_max(orig_rl).value % right.value)) {
2095                         sval_t chop = rl_max(orig_rl);
2096                         chop.value -= chop.value % right.value;
2097                         chop.value++;
2098                         if (!false_rl)
2099                                 false_rl = clone_rl(orig_rl);
2100                         false_rl = remove_range(false_rl, chop, rl_max(orig_rl))
2101                 }
2102         }

2104         set_extra_expr_true_false(expr->left,
2105                                   true_rl ? alloc_estate_rl(true_rl) : NULL,
2106                                   false_rl ? alloc_estate_rl(false_rl) : NULL);
2107 }

2109 /* this is actually hooked from smatch_implied.c...  it's hacky, yes */
2110 void __extra_match_condition(struct expression *expr)
2111 {
1991         struct smatch_state *pre_state;
1992         struct smatch_state *true_state;
1993         struct smatch_state *false_state;
1994         struct range_list *pre_rl;

2112         expr = strip_expr(expr);
2113         switch (expr->type) {
2114         case EXPR_CALL:
```

```
2115                 function_comparison(expr, SPECIAL_NOTEQUAL, zero_expr());
2116                 return;
2117         case EXPR_PREOP:
2118         case EXPR_SYMBOL:
2119         case EXPR_DEREF:
2120                 handle_comparison(get_type(expr), expr, SPECIAL_NOTEQUAL, zero_e
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))
2121                 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         }
2122         case EXPR_COMPARE:
2123                 match_comparison(expr);
2124                 return;
2125         case EXPR_ASSIGNMENT:
2126                 __extra_match_condition(expr->left);
2127                 return;
2128         case EXPR_BINOP:
2129                 if (expr->op == '&')
2130                         handle_AND_condition(expr);
2131                 if (expr->op == '%')
2132                         handle_MOD_condition(expr);
2133                 return;
2134         }
2135 }
```

_____*unchanged_portion_omitted_*

```
2254 static int is_kzalloc_info(struct sm_state *sm)
2156 static int filter_unused_kzalloc_info(struct expression *call, int param, char *
2255 {
2256         sval_t sval;

2258         /*
2259          * kzalloc() information is treated as special because so there is just
2260          * a lot of stuff initialized to zero and it makes building the database
2261          * take hours and hours.
2262          *
2263          * In theory, we should just remove this line and not pass any unused
2264          * information, but I'm not sure enough that this code works so I want
2265          * to hold off on that for now.
2266          */
2267         if (!estate_get_single_value(sm->state, &sval))
2268                 return 0;
2269         if (sval.value != 0)
2270                 return 0;
2271         return 1;
2272 }

2274 static int is_really_long(struct sm_state *sm)
2275 {
```



```
2276         const char *p;
2277         int cnt = 0;

2279         p = sm->name;
2280         while ((p = strstr(p, "->"))) {
2281                 p += 2;
2282                 cnt++;
2283         }

2285         if (cnt < 3 ||
2286             strlen(sm->name) < 40)
2287                 return 0;
2288         return 1;
2289 }

2291 static int filter_unused_param_value_info(struct expression *call, int param, ch
2292 {
2293         int found = 0;

2295         /* for function pointers assume everything is used */
2296         if (call->fn->type != EXPR_SYMBOL)
2297                 return 0;

2299         /*
2300          * This is to handle __builtin_mul_overflow().  In an ideal world we
2301          * would only need this for invalid code.
2302          *
2303          */
2304         if (!call->fn->symbol)
2305                 return 0;

2307         if (!is_kzalloc_info(sm) && !is_really_long(sm))
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)
2308                 return 0;

2310         run_sql(&param_used_callback, &found,
2311                 "select * from return_implies where %s and type = %d and paramet
2312                 get_static_filter(call->fn->symbol), PARAM_USED, param, printed_
2313         if (found)
2314                 return 0;

2316         /* If the database is not built yet, then assume everything is used */
2317         run_sql(&param_used_callback, &found,
2318                 "select * from return_implies where %s and type = %d;",
2319                 get_static_filter(call->fn->symbol), PARAM_USED);
2320         if (!found)
2321                 return 0;

2323         return 1;
2324 }
```

_____*unchanged_portion_omitted_*

```
2371 static void struct_member_callback(struct expression *call, int param, char *pri
2372 {
2373         struct range_list *rl;
2374         sval_t dummy;
```

```
2376         if (estate_is_whole(sm->state))
2377                 return;
2378         if (filter_unused_param_value_info(call, param, printed_name, sm))
2252         if (filter_unused_kzalloc_info(call, param, printed_name, sm))
2379                 return;
2380         rl = estate_rl(sm->state);
2381         rl = intersect_with_real_abs_var_sym(sm->name, sm->sym, rl);
2382         sql_insert_caller_info(call, PARAM_VALUE, param, printed_name, show_rl(r
2383         if (!estate_get_single_value(sm->state, &dummy)) {
2384                 if (estate_has_hard_max(sm->state))
2385                         sql_insert_caller_info(call, HARD_MAX, param, printed_na
2386                                                sval_to_str(estate_max(sm->state)
2387                 if (estate_has_fuzzy_max(sm->state))
2388                         sql_insert_caller_info(call, FUZZY_MAX, param, printed_n
2389                                                sval_to_str(estate_get_fuzzy_max(
2390         }
2391 }

2393 static void returned_struct_members(int return_id, char *return_ranges, struct e
2394 {
2395         struct symbol *returned_sym;
2396         char *returned_name;
2397         struct sm_state *sm;
2266         const char *param_name;
2398         char *compare_str;
2399         char name_buf[256];
2400         char val_buf[256];
2401         int len;
2268         char buf[256];

2403         // FIXME handle *$

2405         if (!is_pointer(expr))
2270         returned_sym = expr_to_sym(expr);
2271         if (!returned_sym)
2406                 return;

2408         returned_name = expr_to_var_sym(expr, &returned_sym);
2409         if (!returned_name || !returned_sym)
2410                 goto free;
2411         len = strlen(returned_name);

2413         FOR_EACH_MY_SM(my_id, __get_cur_stree(), sm) {
2414                 if (!estate_rl(sm->state))
2415                         continue;
2416                 if (returned_sym != sm->sym)
2417                         continue;
2418                 if (strncmp(returned_name, sm->name, len) != 0)

2280                 param_name = get_param_name(sm);
2281                 if (!param_name)
2419                         continue;
2420                 if (sm->name[len] != '-')
2283                 if (strcmp(param_name, "$") == 0)
2421                         continue;

2423                 snprintf(name_buf, sizeof(name_buf), "$%s", sm->name + len);

2425                 compare_str = name_sym_to_param_comparison(sm->name, sm->sym);
2426                 if (!compare_str && estate_is_whole(sm->state))
2427                         continue;
2428                 snprintf(val_buf, sizeof(val_buf), "%s%s", sm->state->name, comp
2288                 snprintf(buf, sizeof(buf), "%s%s", sm->state->name, compare_str

2430                 sql_insert_return_states(return_id, return_ranges, PARAM_VALUE,
2431                                          -1, name_buf, val_buf);
```



```
2291                                                  -1, param_name, buf);
2432         } END_FOR_EACH_SM(sm);

2434 free:
2435         free_string(returned_name);
2436 }
______unchanged_portion_omitted_

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 }

2448 static int basically_the_same(struct range_list *orig, struct range_list *new)
2449 {
2450         if (rl_equiv(orig, new))
2451                 return 1;

2453         /*
2454          * The whole range is essentially the same as 0,4096-27777777777 so
2455          * don't overwrite the implications just to store that.
2456          *
2457          */
2458         if (rl_type(orig)->type == SYM_PTR &&
2459             is_whole_rl(orig) &&
2460             rl_min(new).value == 0 &&
2461             rl_max(new).value == valid_ptr_max)
2462                 return 1;
2463         return 0;
2464 }
______unchanged_portion_omitted_

2486 static void db_param_limit_filter(struct expression *expr, int param, char *key,
2487 {
2488         struct expression *arg;
2489         char *name;
2490         struct symbol *sym;
2491         struct var_sym_list *vsl = NULL;
2492         struct sm_state *sm;
2493         struct symbol *compare_type, *var_type;
2494         struct range_list *rl;
2495         struct range_list *limit;
2496         struct range_list *new;
2497         char *other_name;
2498         struct symbol *other_sym;
2366         char *tmp_name;
2367         struct symbol *tmp_sym;

2500         while (expr->type == EXPR_ASSIGNMENT)
2501                 expr = strip_expr(expr->right);
2502         if (expr->type != EXPR_CALL)
2503                 return;

2505         arg = get_argument_from_call_expr(expr->args, param);
2506         if (!arg)
2507                 return;

2509         if (strcmp(key, "$") == 0)
2510                 compare_type = get_arg_type(expr->fn, param);
```

```
2511          else
2512                  compare_type = get_member_type_from_key(arg, key);

2514          call_results_to_rl(expr, compare_type, value, &limit);
2515          if (strcmp(key, "$") == 0)
2516                  move_known_to_rl(&arg, &limit);
2517          name = get_chunk_from_key(arg, key, &sym, &vsl);
2518          if (!name)
2519                  return;
2520          if (op != PARAM_LIMIT && !sym)
2521                  goto free;

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);

2523          sm = get_sm_state(SMATCH_EXTRA, name, sym);
2524          if (sm)
2525                  rl = estate_rl(sm->state);
2526          else
2527                  rl = alloc_whole_rl(compare_type);

2529          if (op == PARAM_LIMIT && !rl_fits_in_type(rl, compare_type))
2530                  goto free;

2398          call_results_to_rl(expr, compare_type, value, &limit);
2532          new = rl_intersection(rl, limit);

2534          var_type = get_member_type_from_key(arg, key);
2535          new = cast_rl(var_type, new);

2537          /* We want to preserve the implications here */
2538          if (sm && basically_the_same(rl, new))
2405          if (sm && basically_the_same(estate_rl(sm->state), new))
2539                  goto free;
2540          other_name = get_other_name_sym(name, sym, &other_sym);
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          }

2542          if (op == PARAM_LIMIT)
2543                  set_extra_nomod_vsl(name, sym, vsl, NULL, alloc_estate_rl(new));
2544          else
2545                  set_extra_mod(name, sym, NULL, alloc_estate_rl(new));

2547          if (other_name && other_sym) {
2548                  if (op == PARAM_LIMIT)
2549                          set_extra_nomod_vsl(other_name, other_sym, vsl, NULL, al
2550                  else
2551                          set_extra_mod(other_name, other_sym, NULL, alloc_estate_
2552          }

2554          if (op == PARAM_LIMIT && arg->type == EXPR_BINOP)
2555                  db_param_limit_binops(arg, key, new);
2556  free:
2557          free_string(name);
2558  }
```

*unchanged_portion_omitted_*

```
2570  static void db_param_add_set(struct expression *expr, int param, char *key, char
2571  {
2572          struct expression *arg;
```



```
2573          char *name;
2574          char *other_name = NULL;
2575          struct symbol *sym, *other_sym;
2438          char *name, *tmp_name;
2439          struct symbol *sym, *tmp_sym;
2576          struct symbol *param_type, *arg_type;
2577          struct smatch_state *state;
2578          struct range_list *new = NULL;
2579          struct range_list *added = NULL;

2581          while (expr->type == EXPR_ASSIGNMENT)
2582                  expr = strip_expr(expr->right);
2583          if (expr->type != EXPR_CALL)
2584                  return;

2586          arg = get_argument_from_call_expr(expr->args, param);
2587          if (!arg)
2588                  return;

2590          arg_type = get_arg_type_from_key(expr->fn, param, arg, key);
2591          param_type = get_member_type_from_key(arg, key);
2592          name = get_variable_from_key(arg, key, &sym);
2593          if (!name || !sym)
2594                  goto free;

2596          state = get_state(SMATCH_EXTRA, name, sym);
2597          if (state)
2598                  new = estate_rl(state);

2600          call_results_to_rl(expr, arg_type, value, &added);
2601          added = cast_rl(param_type, added);
2602          if (op == PARAM_SET)
2603                  new = added;
2604          else
2605                  new = rl_union(new, added);

2607          other_name = get_other_name_sym_nostack(name, sym, &other_sym);
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          }
2608          set_extra_mod(name, sym, NULL, alloc_estate_rl(new));
2609          if (other_name && other_sym)
2610                  set_extra_mod(other_name, other_sym, NULL, alloc_estate_rl(new))
2611  free:
2612          free_string(other_name);
2613          free_string(name);
2614  }
```

*unchanged_portion_omitted_*

```
2630  static void match_lost_param(struct expression *call, int param)
2631  {
2632          struct expression *arg;

2634          if (is_const_param(call->fn, param))
2635                  return;

2637          arg = get_argument_from_call_expr(call->args, param);
2638          if (!arg)
2639                  return;

2641          arg = strip_expr(arg);
2642          if (arg->type == EXPR_PREOP && arg->op == '&')
2643                  set_extra_expr_mod(arg->unop, alloc_estate_whole(get_type(arg->u
```

```
2644         else
2645                 ; /* if pointer then set struct members, maybe?*/
2646 }

2648 static void db_param_value(struct expression *expr, int param, char *key, char *
2649 {
2650         struct expression *call;
2651         char *name;
2652         struct symbol *sym;
2653         struct symbol *type;
2654         struct range_list *rl = NULL;

2656         if (param != -1)
2657                 return;

2659         call = expr;
2660         while (call->type == EXPR_ASSIGNMENT)
2661                 call = strip_expr(call->right);
2662         if (call->type != EXPR_CALL)
2663                 return;

2665         type = get_member_type_from_key(expr->left, key);
2666         name = get_variable_from_key(expr->left, key, &sym);
2667         if (!name || !sym)
2668                 goto free;

2670         call_results_to_rl(call, type, value, &rl);

2672         set_extra_mod(name, sym, NULL, alloc_estate_rl(rl));
2673 free:
2674         free_string(name);
2675 }

2677 static void match_call_info(struct expression *expr)
2678 {
2679         struct smatch_state *state;
2680         struct range_list *rl = NULL;
2681         struct expression *arg;
2682         struct symbol *type;
2683         sval_t dummy;
2684         int i = 0;

2686         FOR_EACH_PTR(expr->args, arg) {
2687                 type = get_arg_type(expr->fn, i);

2689                 get_absolute_rl(arg, &rl);
2690                 rl = cast_rl(type, rl);

2692                 if (!is_whole_rl(rl)) {
2693                         rl = intersect_with_real_abs_expr(arg, rl);
2694                         sql_insert_caller_info(expr, PARAM_VALUE, i, "$", show_r
2695                 }
2696                 state = get_state_expr(SMATCH_EXTRA, arg);
2697                 if (!estate_get_single_value(state, &dummy) && estate_has_hard_m
2698                         sql_insert_caller_info(expr, HARD_MAX, i, "$",
2699                                                sval_to_str(estate_max(state)));
2700                 }
2701                 if (estate_has_fuzzy_max(state)) {
2702                         sql_insert_caller_info(expr, FUZZY_MAX, i, "$",
2703                                                sval_to_str(estate_get_fuzzy_max(
2704                 }
2705                 i++;
2706         } END_FOR_EACH_PTR(arg);
2707 }

2709 static void set_param_value(const char *name, struct symbol *sym, char *key, cha
```



```
2710 {
2711         struct expression *expr;
2712         struct range_list *rl = NULL;
2713         struct smatch_state *state;
2714         struct symbol *type;
2715         char fullname[256];
2716         char *key_orig = key;
2717         bool add_star = false;
2718         sval_t dummy;

2720         if (key[0] == '*') {
2721                 add_star = true;
2722                 key++;
2723         }
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;

2725         snprintf(fullname, 256, "%s%s%s", add_star ? "*" : "", name, key + 1);

2727         expr = symbol_expression(sym);
2728         type = get_member_type_from_key(expr, key_orig);
2567         type = get_member_type_from_key(symbol_expression(sym), key);
2729         str_to_rl(type, value, &rl);
2730         state = alloc_estate_rl(rl);
2731         if (estate_get_single_value(state, &dummy))
2732                 estate_set_hard_max(state);
2733         set_state(SMATCH_EXTRA, fullname, sym, state);
2734 }

2736 static void set_param_fuzzy_max(const char *name, struct symbol *sym, char *key,
2575 static void set_param_hard_max(const char *name, struct symbol *sym, char *key,
2737 {
2738         struct range_list *rl = NULL;
2739         struct smatch_state *state;
2740         struct symbol *type;
2741         char fullname[256];
2742         sval_t max;

2744         if (strcmp(key, "*$") == 0)
2745                 snprintf(fullname, sizeof(fullname), "*%s", name);
2746         else if (strncmp(key, "$", 1) == 0)
2747                 snprintf(fullname, 256, "%s%s", name, key + 1);
2748         else
2749                 return;

2751         state = get_state(SMATCH_EXTRA, fullname, sym);
2752         if (!state)
2753                 return;
2754         type = estate_type(state);
2593         type = get_member_type_from_key(symbol_expression(sym), key);
2755         str_to_rl(type, value, &rl);
2756         if (!rl_to_sval(rl, &max))
2757                 return;
2758         estate_set_fuzzy_max(state, max);
2759 }

2761 static void set_param_hard_max(const char *name, struct symbol *sym, char *key,
2762 {
2763         struct smatch_state *state;
2764         char fullname[256];

2766         if (strcmp(key, "*$") == 0)
```

```
2767                 snprintf(fullname, sizeof(fullname), "*%s", name);
2768         else if (strncmp(key, "$", 1) == 0)
2769                 snprintf(fullname, 256, "%s%s", name, key + 1);
2770         else
2771                 return;

2773         state = get_state(SMATCH_EXTRA, fullname, sym);
2774         if (!state)
2775                 return;
2776         estate_set_hard_max(state);
2777 }

2779 struct sm_state *get_extra_sm_state(struct expression *expr)
2780 {
2781         char *name;
2782         struct symbol *sym;
2783         struct sm_state *ret = NULL;

2785         name = expr_to_known_chunk_sym(expr, &sym);
2786         if (!name)
2787                 goto free;

2789         ret = get_sm_state(SMATCH_EXTRA, name, sym);
2790 free:
2791         free_string(name);
2792         return ret;
2793 }
______unchanged_portion_omitted_

2805 void register_smatch_extra(int id)
2806 {
2807         my_id = id;

2809         set_dynamic_states(my_id);
2810         add_merge_hook(my_id, &merge_estates);
2811         add_unmatched_state_hook(my_id, &unmatched_state);
2812         select_caller_info_hook(set_param_value, PARAM_VALUE);
2813         select_caller_info_hook(set_param_fuzzy_max, FUZZY_MAX);
2814         select_caller_info_hook(set_param_hard_max, HARD_MAX);
2633         select_caller_info_hook(set_param_hard_max, FUZZY_MAX);
2815         select_return_states_before(&db_limited_before);
2816         select_return_states_hook(PARAM_LIMIT, &db_param_limit);
2817         select_return_states_hook(PARAM_FILTER, &db_param_filter);
2818         select_return_states_hook(PARAM_ADD, &db_param_add);
2819         select_return_states_hook(PARAM_SET, &db_param_set);
2820         add_lost_param_hook(&match_lost_param);
2821         select_return_states_hook(PARAM_VALUE, &db_param_value);
2822         select_return_states_after(&db_limited_after);
2823 }
______unchanged_portion_omitted_

2845 void register_smatch_extra_links(int id)
2846 {
2847         link_id = id;
2848         set_dynamic_states(link_id);
2849 }
______unchanged_portion_omitted_
```

```
**********************************************************
   11815 Mon Aug  5 08:38:31 2019
new/usr/src/tools/smatch/src/smatch_extra.h
11506 smatch resync
**********************************************************
______unchanged_portion_omitted_

 26 DECLARE_PTR_LIST(related_list, struct relation);

 28 struct data_info {
 29         struct related_list *related;
 30         struct range_list *value_ranges;
 31         sval_t fuzzy_max;
 32         unsigned int hard_max:1;
 33         unsigned int capped:1;
 34 };
 35 DECLARE_ALLOCATOR(data_info);

 37 extern struct string_list *__ignored_macros;

 39 /* these are implemented in smatch_ranges.c */
 40 struct range_list *rl_zero(void);
 41 struct range_list *rl_one(void);
 42 char *show_rl(struct range_list *list);
 43 int str_to_comparison_arg(const char *c, struct expression *call, int *compariso
 44 void str_to_rl(struct symbol *type, char *value, struct range_list **rl);
 45 void call_results_to_rl(struct expression *call, struct symbol *type, const char
 44 void call_results_to_rl(struct expression *call, struct symbol *type, char *valu

 47 struct data_range *alloc_range(sval_t min, sval_t max);
 48 struct data_range *alloc_range_perm(sval_t min, sval_t max);

 50 int rl_fits_in_type(struct range_list *rl, struct symbol *type);

 52 struct range_list *alloc_rl(sval_t min, sval_t max);
 53 struct range_list *clone_rl(struct range_list *list);
 54 struct range_list *clone_rl_permanent(struct range_list *list);
 55 struct range_list *alloc_whole_rl(struct symbol *type);

 57 void add_range(struct range_list **list, sval_t min, sval_t max);
 58 struct range_list *remove_range(struct range_list *list, sval_t min, sval_t max)
 59 void tack_on(struct range_list **list, struct data_range *drange);

 61 int true_comparison_range(struct data_range *left, int comparison, struct data_r
 62 int true_comparison_range_LR(int comparison, struct data_range *var, struct data
 63 int false_comparison_range_LR(int comparison, struct data_range *var, struct dat

 65 int possibly_true(struct expression *left, int comparison, struct expression *ri
 66 int possibly_true_rl(struct range_list *left_ranges, int comparison, struct rang
 67 int possibly_true_rl_LR(int comparison, struct range_list *a, struct range_list

 69 int possibly_false(struct expression *left, int comparison, struct expression *r
 70 int possibly_false_rl(struct range_list *left_ranges, int comparison, struct ran
 71 int possibly_false_rl_LR(int comparison, struct range_list *a, struct range_list

 73 int rl_has_sval(struct range_list *rl, sval_t sval);
 74 int ranges_equiv(struct data_range *one, struct data_range *two);

 76 int rl_equiv(struct range_list *one, struct range_list *two);
 77 int is_whole_rl(struct range_list *rl);
 78 int is_unknown_ptr(struct range_list *rl);
 79 int is_whole_rl_non_zero(struct range_list *rl);
 80 int estate_is_unknown(struct smatch_state *state);

 82 sval_t rl_min(struct range_list *rl);
 83 sval_t rl_max(struct range_list *rl);
```



```
 84 int rl_to_sval(struct range_list *rl, sval_t *sval);
 85 struct symbol *rl_type(struct range_list *rl);

 83 struct range_list *rl_invert(struct range_list *orig);
 87 struct range_list *rl_filter(struct range_list *rl, struct range_list *filter);
 88 struct range_list *rl_intersection(struct range_list *one, struct range_list *tw
 89 struct range_list *rl_union(struct range_list *one, struct range_list *two);
 90 struct range_list *rl_binop(struct range_list *left, int op, struct range_list *

 92 void push_rl(struct range_list_stack **rl_stack, struct range_list *rl);
 93 struct range_list *pop_rl(struct range_list_stack **rl_stack);
 94 struct range_list *top_rl(struct range_list_stack *rl_stack);
 95 void filter_top_rl(struct range_list_stack **rl_stack, struct range_list *filter

 97 struct range_list *rl_truncate_cast(struct symbol *type, struct range_list *rl);
 98 struct range_list *cast_rl(struct symbol *type, struct range_list *rl);
 99 int get_implied_rl(struct expression *expr, struct range_list **rl);
100 int get_absolute_rl(struct expression *expr, struct range_list **rl);
101 int get_real_absolute_rl(struct expression *expr, struct range_list **rl);
102 struct range_list *var_to_absolute_rl(struct expression *expr);
103 int custom_get_absolute_rl(struct expression *expr,
104                            struct range_list *(*fn)(struct expression *expr),
105                            struct range_list **rl);
106 int get_implied_rl_var_sym(const char *var, struct symbol *sym, struct range_lis
107 void split_comparison_rl(struct range_list *left_orig, int op, struct range_list
108                 struct range_list **left_true_rl, struct range_list **left_false
109                 struct range_list **right_true_rl, struct range_list **right_fal

111 void free_data_info_allocs(void);
112 void free_all_rl(void);

114 /* smatch_estate.c */

116 struct smatch_state *alloc_estate_empty(void);
117 struct smatch_state *alloc_estate_sval(sval_t sval);
118 struct smatch_state *alloc_estate_range(sval_t min, sval_t max);
119 struct smatch_state *alloc_estate_rl(struct range_list *rl);
120 struct smatch_state *alloc_estate_whole(struct symbol *type);
121 struct smatch_state *clone_estate(struct smatch_state *state);
122 struct smatch_state *clone_estate_cast(struct symbol *type, struct smatch_state
123 struct smatch_state *clone_partial_estate(struct smatch_state *state, struct ran

125 struct smatch_state *merge_estates(struct smatch_state *s1, struct smatch_state

127 int estates_equiv(struct smatch_state *one, struct smatch_state *two);
128 int estate_is_whole(struct smatch_state *state);
129 int estate_is_empty(struct smatch_state *state);

131 struct range_list *estate_rl(struct smatch_state *state);
132 struct related_list *estate_related(struct smatch_state *state);

134 sval_t estate_min(struct smatch_state *state);
135 sval_t estate_max(struct smatch_state *state);
136 struct symbol *estate_type(struct smatch_state *state);

138 int estate_has_fuzzy_max(struct smatch_state *state);
139 sval_t estate_get_fuzzy_max(struct smatch_state *state);
140 void estate_set_fuzzy_max(struct smatch_state *state, sval_t max);
141 void estate_copy_fuzzy_max(struct smatch_state *new, struct smatch_state *old);
142 void estate_clear_fuzzy_max(struct smatch_state *state);
143 int estate_has_hard_max(struct smatch_state *state);
144 void estate_set_hard_max(struct smatch_state *state);
145 void estate_clear_hard_max(struct smatch_state *state);
146 int estate_get_hard_max(struct smatch_state *state, sval_t *sval);
147 bool estate_capped(struct smatch_state *state);
148 void estate_set_capped(struct smatch_state *state);
```

```
150 int estate_get_single_value(struct smatch_state *state, sval_t *sval);
151 struct smatch_state *get_implied_estate(struct expression *expr);

153 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
154 struct data_info *clone_dinfo_perm(struct data_info *dinfo);
155 struct smatch_state *clone_estate_perm(struct smatch_state *state);

157 /* smatch_extra.c */
158 bool is_impossible_variable(struct expression *expr);
159 struct sm_state *get_extra_sm_state(struct expression *expr);
160 struct smatch_state *get_extra_state(struct expression *expr);
161 void call_extra_mod_hooks(const char *name, struct symbol *sym, struct expressio
162 void set_extra_mod(const char *name, struct symbol *sym, struct expression *expr
163 void set_extra_expr_mod(struct expression *expr, struct smatch_state *state);
164 void set_extra_nomod(const char *name, struct symbol *sym, struct expression *ex
165 void set_extra_nomod_vsl(const char *name, struct symbol *sym, struct var_sym_li
166 void set_extra_expr_nomod(struct expression *expr, struct smatch_state *state);
167 void set_extra_mod_helper(const char *name, struct symbol *sym, struct expressio

169 struct data_info *get_dinfo(struct smatch_state *state);

171 void add_extra_mod_hook(void (*fn)(const char *name, struct symbol *sym, struct
172 void add_extra_nomod_hook(void (*fn)(const char *name, struct symbol *sym, struc
173 int implied_not_equal(struct expression *expr, long long val);
174 int implied_not_equal_name_sym(char *name, struct symbol *sym, long long val);
175 int parent_is_null_var_sym(const char *name, struct symbol *sym);
176 int parent_is_null(struct expression *expr);
177 int parent_is_free_var_sym_strict(const char *name, struct symbol *sym);
178 int parent_is_free_var_sym(const char *name, struct symbol *sym);
179 int parent_is_free(struct expression *expr);

181 struct sm_state *__extra_handle_canonical_loops(struct statement *loop, struct s
182 int __iterator_unchanged(struct sm_state *sm);
183 void __extra_pre_loop_hook_after(struct sm_state *sm,
184                                  struct statement *iterator,
185                                  struct expression *condition);

187 /* smatch_equiv.c */
188 void set_equiv(struct expression *left, struct expression *right);
189 void set_related(struct smatch_state *estate, struct related_list *rlist);
190 struct related_list *get_shared_relations(struct related_list *one,
191                                           struct related_list *two);
192 struct related_list *clone_related_list(struct related_list *related);
193 void remove_from_equiv(const char *name, struct symbol *sym);
194 void remove_from_equiv_expr(struct expression *expr);
195 void set_equiv_state_expr(int id, struct expression *expr, struct smatch_state *

197 /* smatch_function_hooks.c */
198 void function_comparison(struct expression *left, int comparison, struct express

200 /* smatch_expressions.c */
201 struct expression *zero_expr();
202 struct expression *value_expr(long long val);
203 struct expression *member_expression(struct expression *deref, int op, struct id
204 struct expression *preop_expression(struct expression *expr, int op);
205 struct expression *deref_expression(struct expression *expr);
206 struct expression *assign_expression(struct expression *left, int op, struct exp
207 struct expression *binop_expression(struct expression *left, int op, struct expr
208 struct expression *array_element_expression(struct expression *array, struct exp
209 struct expression *symbol_expression(struct symbol *sym);
210 struct expression *string_expression(char *str);
211 struct expression *compare_expression(struct expression *left, int op, struct ex
212 struct expression *unknown_value_expression(struct expression *expr);
213 int is_fake_call(struct expression *expr);
```



```
214 struct expression *gen_expression_from_key(struct expression *arg, const char *k
215 void free_tmp_expressions(void);
216 void expr_set_parent_expr(struct expression *expr, struct expression *parent);
217 void expr_set_parent_stmt(struct expression *expr, struct statement *parent);
218 struct expression *expr_get_parent_expr(struct expression *expr);
219 struct statement *expr_get_parent_stmt(struct expression *expr);

221 /* smatch_param_limit.c */
222 struct smatch_state *get_orig_estate(const char *name, struct symbol *sym);

224 /* smatch_real_absolute.c */
225 struct smatch_state *get_real_absolute_state(struct expression *expr);
226 struct smatch_state *get_real_absolute_state_var_sym(const char *name, struct sy

228 /* smatch_imaginary_absolute.c */
229 void __save_imaginary_state(struct expression *expr, struct range_list *true_rl,
230 int get_imaginary_absolute(struct expression *expr, struct range_list **rl);
```

```
**********************************************************
   47355 Mon Aug  5 08:38:31 2019
new/usr/src/tools/smatch/src/smatch_flow.c
11506 smatch resync
**********************************************************
```

```
   1 /*
   2  * Copyright (C) 2006,2008 Dan Carpenter.
   3  *
   4  * This program is free software; you can redistribute it and/or
   5  * modify it under the terms of the GNU General Public License
   6  * as published by the Free Software Foundation; either version 2
   7  * of the License, or (at your option) any later version.
   8  *
   9  * This program is distributed in the hope that it will be useful,
  10  * but WITHOUT ANY WARRANTY; without even the implied warranty of
  11  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  12  * GNU General Public License for more details.
  13  *
  14  * You should have received a copy of the GNU General Public License
  15  * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
  16  */

  18 #define _GNU_SOURCE 1
  19 #include <unistd.h>
  20 #include <stdio.h>
  21 #include "token.h"
  22 #include "scope.h"
  23 #include "smatch.h"
  24 #include "smatch_expression_stacks.h"
  25 #include "smatch_extra.h"
  26 #include "smatch_slist.h"

  28 int __in_fake_assign;
  29 int __in_fake_struct_assign;
  30 int in_fake_env;
  31 int final_pass;
  32 int __inline_call;
  33 struct expression  *__inline_fn;

  35 static int __smatch_lineno = 0;

  37 static char *base_file;
  38 static const char *filename;
  39 static char *pathname;
  40 static char *full_filename;
  41 static char *full_base_file;
  42 static char *cur_func;
  43 static unsigned int loop_count;
  44 static int last_goto_statement_handled;
  45 int __expr_stmt_count;
  46 int __in_function_def;
  47 static struct expression_list *switch_expr_stack = NULL;
  48 static struct expression_list *post_op_stack = NULL;

  50 static struct ptr_list *backup;

  52 struct expression_list *big_expression_stack;
  53 struct statement_list *big_statement_stack;
  54 struct statement *__prev_stmt;
  55 struct statement *__cur_stmt;
  56 struct statement *__next_stmt;
  57 int __in_pre_condition = 0;
  58 int __bail_on_rest_of_function = 0;
  59 static struct timeval fn_start_time;
  60 static struct timeval outer_fn_start_time;
  61 char *get_function(void) { return cur_func; }
```



```
  62 int get_lineno(void) { return __smatch_lineno; }
  63 int inside_loop(void) { return !!loop_count; }
  64 int definitely_inside_loop(void) { return !!(loop_count & ~0x08000000); }
  65 struct expression *get_switch_expr(void) { return top_expression(switch_expr_sta
  66 int in_expression_statement(void) { return !!__expr_stmt_count; }

  68 static void split_symlist(struct symbol_list *sym_list);
  69 static void split_declaration(struct symbol_list *sym_list);
  70 static void split_expr_list(struct expression_list *expr_list, struct expression
  71 static void add_inline_function(struct symbol *sym);
  72 static void parse_inline(struct expression *expr);

  74 int option_assume_loops = 0;
  75 int option_two_passes = 0;
  76 struct symbol *cur_func_sym = NULL;
  77 struct stree *global_states;

  79 const unsigned long valid_ptr_min = 4096;
  80 unsigned long valid_ptr_max = ULONG_MAX & ~(MTAG_OFFSET_MASK);
  81 const sval_t valid_ptr_min_sval = {
  79 long long valid_ptr_min = 4096;
  80 long long valid_ptr_max = 2117777777;
  81 sval_t valid_ptr_min_sval = {
  82         .type = &ptr_ctype,
  83         {.value = 4096},
  84 };
  85 sval_t valid_ptr_max_sval = {
  86         .type = &ptr_ctype,
  87         {.value = ULONG_MAX & ~(MTAG_OFFSET_MASK)},
  87         {.value = LONG_MAX - 100000},
  88 };
  89 struct range_list *valid_ptr_rl;

  91 void alloc_valid_ptr_rl(void)
  91 static void set_valid_ptr_max(void)
  92 {
  93         valid_ptr_max = sval_type_max(&ulong_ctype).value & ~(MTAG_OFFSET_MASK);
  93         if (type_bits(&ptr_ctype) == 32)
  94                 valid_ptr_max = 2117777777;
  95         else if (type_bits(&ptr_ctype) == 64)
  96                 valid_ptr_max = 211777777777777777LL;

  94         valid_ptr_max_sval.value = valid_ptr_max;
  99 }

 101 static void alloc_valid_ptr_rl(void)
 102 {
  96         valid_ptr_rl = alloc_rl(valid_ptr_min_sval, valid_ptr_max_sval);
  97         valid_ptr_rl = cast_rl(&ptr_ctype, valid_ptr_rl);
  98         valid_ptr_rl = clone_rl_permanent(valid_ptr_rl);
  99 }
______unchanged_portion_omitted_

 364 void __split_expr(struct expression *expr)
 365 {
 366         if (!expr)
 367                 return;

 369         // sm_msg(" Debug expr_type %d %s", expr->type, show_special(expr->op));

 371         if (__in_fake_assign && expr->type != EXPR_ASSIGNMENT)
 372                 return;
 373         if (__in_fake_assign >= 4)  /* don't allow too much nesting */
 374                 return;

 376         push_expression(&big_expression_stack, expr);
```

```
377         set_position(expr->pos);
378         __pass_to_client(expr, EXPR_HOOK);

380         switch (expr->type) {
381         case EXPR_PREOP:
382                 expr_set_parent_expr(expr->unop, expr);

384                 if (expr->op == '*' &&
385                     !prev_expression_is_getting_address(expr))
386                         __pass_to_client(expr, DEREF_HOOK);
387                 __split_expr(expr->unop);
388                 __pass_to_client(expr, OP_HOOK);
389                 break;
390         case EXPR_POSTOP:
391                 expr_set_parent_expr(expr->unop, expr);

393                 __split_expr(expr->unop);
394                 push_expression(&post_op_stack, expr);
395                 break;
396         case EXPR_STATEMENT:
397                 __expr_stmt_count++;
398                 if (expr->statement && !expr->statement) {
399                         stmt_set_parent_stmt(expr->statement,
400                                              last_ptr_list((struct ptr_list *)big_sta
401                 }
402                 __split_stmt(expr->statement);
403                 __expr_stmt_count--;
404                 break;
405         case EXPR_LOGICAL:
406         case EXPR_COMPARE:
407                 expr_set_parent_expr(expr->left, expr);
408                 expr_set_parent_expr(expr->right, expr);

410                 __pass_to_client(expr, LOGIC_HOOK);
411                 __handle_logic(expr);
412                 break;
413         case EXPR_BINOP:
414                 expr_set_parent_expr(expr->left, expr);
415                 expr_set_parent_expr(expr->right, expr);

417                 __pass_to_client(expr, BINOP_HOOK);
418         case EXPR_COMMA:
419                 expr_set_parent_expr(expr->left, expr);
420                 expr_set_parent_expr(expr->right, expr);

422                 __split_expr(expr->left);
423                 __process_post_op_stack();
424                 __split_expr(expr->right);
425                 break;
426         case EXPR_ASSIGNMENT: {
427                 struct expression *right;

429                 expr_set_parent_expr(expr->left, expr);
430                 expr_set_parent_expr(expr->right, expr);

432                 right = strip_expr(expr->right);
433                 if (!right)
434                         break;

436                 __pass_to_client(expr, RAW_ASSIGNMENT_HOOK);

438                 /* foo = !bar() */
439                 if (__handle_condition_assigns(expr))
440                         break;
441                 /* foo = (x < 5 ? foo : 5); */
442                 if (__handle_select_assigns(expr))
```



```
443                         break;
444                 /* foo = ({frob(); frob(); frob(); 1;}) */
445                 if (__handle_expr_statement_assigns(expr))
446                         break;
447                 /* foo = (3, 4); */
448                 if (handle_comma_assigns(expr))
449                         break;
450                 if (handle_postop_assigns(expr))
451                         break;
452                 if (handle__builtin_choose_expr_assigns(expr))
453                         break;

455                 __split_expr(expr->right);
456                 if (outside_of_function())
457                         __pass_to_client(expr, GLOBAL_ASSIGNMENT_HOOK);
458                 else
459                         __pass_to_client(expr, ASSIGNMENT_HOOK);

461                 __fake_struct_member_assignments(expr);

463                 if (expr->op == '=' && right->type == EXPR_CALL)
464                         __pass_to_client(expr, CALL_ASSIGNMENT_HOOK);

466                 if (get_macro_name(right->pos) &&
467                     get_macro_name(expr->pos) != get_macro_name(right->pos))
468                         __pass_to_client(expr, MACRO_ASSIGNMENT_HOOK);

470                 __pass_to_client(expr, ASSIGNMENT_HOOK_AFTER);

472                 __split_expr(expr->left);
473                 break;
474         }
475         case EXPR_DEREF:
476                 expr_set_parent_expr(expr->deref, expr);

478                 __pass_to_client(expr, DEREF_HOOK);
479                 __split_expr(expr->deref);
480                 break;
481         case EXPR_SLICE:
482                 expr_set_parent_expr(expr->base, expr);

484                 __split_expr(expr->base);
485                 break;
486         case EXPR_CAST:
487         case EXPR_FORCE_CAST:
488                 expr_set_parent_expr(expr->cast_expression, expr);

490                 __pass_to_client(expr, CAST_HOOK);
491                 __split_expr(expr->cast_expression);
492                 break;
493         case EXPR_SIZEOF:
494                 if (expr->cast_expression)
495                         __pass_to_client(strip_parens(expr->cast_expression),
496                                          SIZEOF_HOOK);
497                 break;
498         case EXPR_OFFSETOF:
499         case EXPR_ALIGNOF:
507                 evaluate_expression(expr);
500                 break;
501         case EXPR_CONDITIONAL:
502         case EXPR_SELECT:
503                 expr_set_parent_expr(expr->conditional, expr);
504                 expr_set_parent_expr(expr->cond_true, expr);
505                 expr_set_parent_expr(expr->cond_false, expr);

507                 if (known_condition_true(expr->conditional)) {
```

```
508                         __split_expr(expr->cond_true);
509                         break;
510                 }
511                 if (known_condition_false(expr->conditional)) {
512                         __split_expr(expr->cond_false);
513                         break;
514                 }
515                 __pass_to_client(expr, SELECT_HOOK);
516                 __split_whole_condition(expr->conditional);
517                 __split_expr(expr->cond_true);
518                 __push_true_states();
519                 __use_false_states();
520                 __split_expr(expr->cond_false);
521                 __merge_true_states();
522                 break;
523         case EXPR_CALL:
524                 expr_set_parent_expr(expr->fn, expr);

526                 if (sym_name_is("__builtin_constant_p", expr->fn))
527                         break;
528                 if (handle__builtin_choose_expr(expr))
529                         break;
530                 split_expr_list(expr->args, expr);
531                 __split_expr(expr->fn);
532                 if (is_inline_func(expr->fn))
533                         add_inline_function(expr->fn->symbol);
534                 if (inlinable(expr->fn))
535                         __inline_call = 1;
536                 __process_post_op_stack();
537                 __pass_to_client(expr, FUNCTION_CALL_HOOK_BEFORE);
538                 __pass_to_client(expr, FUNCTION_CALL_HOOK);
539                 __inline_call = 0;
540                 if (inlinable(expr->fn)) {
541                         parse_inline(expr);
542                 }
543                 __pass_to_client(expr, CALL_HOOK_AFTER_INLINE);
544                 if (is_noreturn_func(expr->fn))
545                         nullify_path();
546                 handle_builtin_overflow_func(expr);
547                 break;
548         case EXPR_INITIALIZER:
549                 split_expr_list(expr->expr_list, expr);
550                 break;
551         case EXPR_IDENTIFIER:
552                 expr_set_parent_expr(expr->ident_expression, expr);
553                 __split_expr(expr->ident_expression);
554                 break;
555         case EXPR_INDEX:
556                 expr_set_parent_expr(expr->idx_expression, expr);
557                 __split_expr(expr->idx_expression);
558                 break;
559         case EXPR_POS:
560                 expr_set_parent_expr(expr->init_expr, expr);
561                 __split_expr(expr->init_expr);
562                 break;
563         case EXPR_SYMBOL:
564                 __pass_to_client(expr, SYM_HOOK);
565                 break;
566         case EXPR_STRING:
567                 __pass_to_client(expr, STRING_HOOK);
568                 break;
569         default:
570                 break;
571         };
572         pop_expression(&big_expression_stack);
573 }
```

unchanged_portion_omitted_


```
873 /*
874  * This defaults to 60 * 5 == 5 minutes, so we'll just multiply
875  * whatever we're given by 5.
876  */
877 bool taking_too_long(void)
881 static int taking_too_long(void)
878 {
879         if (option_timeout &&
880             (ms_since(&outer_fn_start_time) / 1000) > option_timeout * 5)
883         if ((ms_since(&outer_fn_start_time) / 1000) > 60 * 5) /* five minutes */
881                 return 1;
882         return 0;
883 }
```

unchanged_portion_omitted_

```
1908 void smatch(struct string_list *filelist)
1911 void smatch(int argc, char **argv)
1909 {
1913         struct string_list *filelist = NULL;
1910         struct symbol_list *sym_list;
1911         struct timeval stop, start;
1912         char *path;
1913         int len;

1915         gettimeofday(&start, NULL);

1921         sparse_initialize(argc, argv, &filelist);
1922         set_valid_ptr_max();
1923         alloc_valid_ptr_rl();
1917         FOR_EACH_PTR_NOTAG(filelist, base_file) {
1918                 path = getcwd(NULL, 0);
1919                 free(full_base_file);
1920                 if (path) {
1921                         len = strlen(path) + 1 + strlen(base_file) + 1;
1922                         full_base_file = malloc(len);
1923                         snprintf(full_base_file, len, "%s/%s", path, base_file);
1924                 } else {
1925                         full_base_file = alloc_string(base_file);
1926                 }
1927                 if (option_file_output)
1928                         open_output_files(base_file);
1929                 sym_list = sparse_keep_tokens(base_file);
1930                 split_c_file_functions(sym_list);
1931         } END_FOR_EACH_PTR_NOTAG(base_file);

1933         gettimeofday(&stop, NULL);

1935         set_position(last_pos);
1936         final_pass = 1;
1937         if (option_time)
1938                 sm_msg("time: %lu", stop.tv_sec - start.tv_sec);
1939         if (option_mem)
1940                 sm_msg("mem: %luKb", get_max_memory());
1941 }
```

unchanged_portion_omitted_

```
**********************************************************
   31805 Mon Aug  5 08:38:31 2019
new/usr/src/tools/smatch/src/smatch_function_hooks.c
11506 smatch resync
**********************************************************
```

_____unchanged_portion_omitted_

```
142 void return_implies_state_sval(const char *look_for, sval_t start, sval_t end,
143                          implication_hook *call_back, void *info)
144 {
145         struct fcall_back *cb;
146
147         cb = alloc_fcall_back(RANGED_CALL, call_back, info);
148         cb->range = alloc_range_perm(start, end);
149         add_callback(func_hash, look_for, cb);
150 }
151
152 void select_return_states_hook(int type, return_implies_hook *callback)
153 {
154         struct return_implies_callback *cb = __alloc_return_implies_callback(0);
155
156         cb->type = type;
157         cb->callback = callback;
158         add_ptr_list(&db_return_states_list, cb);
159 }
```

_____unchanged_portion_omitted_

```
247 static int assign_ranged_funcs(const char *fn, struct expression *expr,
248                                 struct call_back_list *call_backs)
249 {
250         struct fcall_back *tmp;
251         struct sm_state *sm;
252         char *var_name;
253         struct symbol *sym;
254         struct smatch_state *estate;
255         struct stree *tmp_stree;
256         struct stree *final_states = NULL;
257         struct range_list *handled_ranges = NULL;
258         struct call_back_list *same_range_call_backs = NULL;
259         struct range_list *rl;
260         int handled = 0;
261
262         if (!call_backs)
263                 return 0;
264
265         var_name = expr_to_var_sym(expr->left, &sym);
266         if (!var_name || !sym)
267                 goto free;
268
269         FOR_EACH_PTR(call_backs, tmp) {
270                 if (tmp->type != RANGED_CALL)
271                         continue;
272
273                 if (in_list_exact_sval(handled_ranges, tmp->range))
274                         continue;
275                 __push_fake_cur_stree();
276                 tack_on(&handled_ranges, tmp->range);
277
278                 same_range_call_backs = get_same_ranged_call_backs(call_backs, t
279                 call_ranged_call_backs(same_range_call_backs, fn, expr->right, e
280                 __free_ptr_list((struct ptr_list **)&same_range_call_backs);
281
282                 rl = alloc_rl(tmp->range->min, tmp->range->max);
283                 rl = cast_rl(get_type(expr->left), rl);
284                 estate = alloc_estate_rl(rl);
271                 estate = alloc_estate_range(tmp->range->min, tmp->range->max);
```



```
285                 set_extra_mod(var_name, sym, expr->left, estate);
286
287                 tmp_stree = __pop_fake_cur_stree();
288                 merge_fake_stree(&final_states, tmp_stree);
289                 free_stree(&tmp_stree);
290                 handled = 1;
291         } END_FOR_EACH_PTR(tmp);
292
293         FOR_EACH_SM(final_states, sm) {
294                 __set_sm(sm);
295         } END_FOR_EACH_SM(sm);
296
297         free_stree(&final_states);
298 free:
299         free_string(var_name);
300         return handled;
301 }
```

_____unchanged_portion_omitted_

```
376 static bool fake_a_param_assignment(struct expression *expr, const char *return_
377 {
378         struct expression *arg, *left, *right, *tmp, *fake_assign;
365         struct expression *arg, *left, *right, *fake_assign;
379         char *p;
380         int param;
381         char buf[256];
382         char *str;
383
384         if (expr->type != EXPR_ASSIGNMENT || expr->op != '=')
385                 return false;
386         left = expr->left;
387         right = expr->right;
388
389         while (right->type == EXPR_ASSIGNMENT)
390                 right = strip_expr(right->right);
391         if (!right || right->type != EXPR_CALL)
392                 return false;
393
394         p = strchr(return_str, '[');
395         if (!p)
396                 return false;
397
398         p++;
399         if (p[0] == '=' && p[1] == '=')
400                 p += 2;
401         if (p[0] != '$')
402                 return false;
403
404         snprintf(buf, sizeof(buf), "%s", p);
405
406         p = buf;
407         p += 1;
408         param = strtol(p, &p, 10);
409
410         p = strchr(p, ']');
411         if (!p || *p != ']')
412                 return false;
413         *p = '\0';
414
415         arg = get_argument_from_call_expr(right->args, param);
416         if (!arg)
417                 return false;
418
419         /* There should be a get_other_name() function which returns an expr */
420         tmp = get_assigned_expr(arg);
421         if (tmp)
```

```
422                 arg = tmp;

424         /*
425          * This is a sanity check to prevent side effects from evaluating stuff
426          * twice.
427          */
428         str = expr_to_chunk_sym_vsl(arg, NULL, NULL);
429         if (!str)
430                 return false;
431         free_string(str);

433         right = gen_expression_from_key(arg, buf);
434         if (!right)  /* Mostly fails for binops like [$0 + 4032] */
435                 return false;
436         fake_assign = assign_expression(left, '=', right);
437         __in_fake_parameter_assign++;
438         __split_expr(fake_assign);
439         __in_fake_parameter_assign--;
440         return true;
441 }

443 static void set_return_assign_state(struct db_callback_info *db_info)
444 static void set_return_state(struct expression *expr, struct db_callback_info *d
445 {
446         struct expression *expr = db_info->expr->left;
447         struct smatch_state *state;

448         if (!db_info->ret_state)
449                 return;

451         state = alloc_estate_rl(cast_rl(get_type(expr), clone_rl(estate_rl(db_in
452         set_extra_expr_mod(expr, state);
453         db_info->ret_state = NULL;
454         fake_a_param_assignment(db_info->expr, db_info->ret_str);
455         db_info->ret_str = NULL;
456 }

458 static void set_other_side_state(struct db_callback_info *db_info)
459 {
460         struct expression *expr = db_info->var_expr;
461         struct smatch_state *state;

463         if (!db_info->ret_state)
464                 return;

466         state = alloc_estate_rl(cast_rl(get_type(expr), clone_rl(estate_rl(db_in
467         set_extra_expr_nomod(expr, state);
468         db_info->ret_state = NULL;
469         db_info->ret_str = NULL;
470 }

472 static void handle_ret_equals_param(char *ret_string, struct range_list *rl, str
473 {
474         char *str;
475         long long param;
476         struct expression *arg;
477         struct range_list *orig;

479         str = strstr(ret_string, "==$");
480         if (!str)
481                 return;
482         str += 3;
483         param = strtoll(str, NULL, 10);
484         arg = get_argument_from_call_expr(call->args, param);
485         if (!arg)
486                 return;
```



```
487         get_absolute_rl(arg, &orig);
488         rl = rl_intersection(orig, rl);
489         if (!rl)
490                 return;
491         set_extra_expr_nomod(arg, alloc_estate_rl(rl));
492 }
______unchanged_portion_omitted_

592 static int db_compare_callback(void *_info, int argc, char **argv, char **azColN
593 {
594         struct db_callback_info *db_info = _info;
595         struct range_list *var_rl = db_info->rl;
596         struct range_list *ret_range;
597         int type, param;
598         char *ret_str, *key, *value;
564         char *key, *value;
599         struct return_implies_callback *tmp;
600         struct stree *stree;
601         int return_id;
602         int comparison;

604         if (argc != 6)
605                 return 0;

607         return_id = atoi(argv[0]);
608         ret_str = argv[1];
609         type = atoi(argv[2]);
610         param = atoi(argv[3]);
611         key = argv[4];
612         value = argv[5];

614         db_info->has_states = 1;
615         if (db_info->prev_return_id != -1 && type == INTERNAL) {
616                 set_other_side_state(db_info);
581                 set_return_state(db_info->var_expr, db_info);
617                 stree = __pop_fake_cur_stree();

619                 if (!db_info->cull)
620                         merge_fake_stree(&db_info->stree, stree);
621                 free_stree(&stree);
622                 __push_fake_cur_stree();
623                 db_info->cull = 0;
624         }
625         db_info->prev_return_id = return_id;

627         if (type == INTERNAL && func_type_mismatch(db_info->expr, value))
628                 db_info->cull = 1;
629         if (db_info->cull)
630                 return 0;
631         if (type == CULL_PATH) {
632                 db_info->cull = 1;
633                 return 0;
634         }

636         if (is_impossible_data(type, db_info->expr, param, key, value)) {
637                 db_info->cull = 1;
638                 return 0;
639         }

641         call_results_to_rl(db_info->expr, get_type(strip_expr(db_info->expr)), r
606         call_results_to_rl(db_info->expr, get_type(strip_expr(db_info->expr)), a
642         ret_range = cast_rl(get_type(db_info->expr), ret_range);
643         if (!ret_range)
644                 ret_range = alloc_whole_rl(get_type(db_info->expr));

646         comparison = db_info->comparison;
```

```
647         if (db_info->left)
648                 comparison = flip_comparison(comparison);

650         if (db_info->true_side) {
651                 if (!possibly_true_rl(var_rl, comparison, ret_range))
652                         return 0;
653                 if (type == PARAM_LIMIT)
654                         param_limit_implications(db_info->expr, param, key, valu
655                 filter_by_comparison(&var_rl, comparison, ret_range);
656                 filter_by_comparison(&ret_range, flip_comparison(comparison), va
657         } else {
658                 if (!possibly_false_rl(var_rl, comparison, ret_range))
659                         return 0;
660                 if (type == PARAM_LIMIT)
661                         param_limit_implications(db_info->expr, param, key, valu
662                 filter_by_comparison(&var_rl, negate_comparison(comparison), ret
663                 filter_by_comparison(&ret_range, flip_comparison(negate_comparis
664         }

666         handle_ret_equals_param(ret_str, ret_range, db_info->expr);
631         handle_ret_equals_param(argv[1], ret_range, db_info->expr);

668         if (type == INTERNAL) {
669                 set_state(-1, "unnull_path", NULL, &true_state);
670                 __add_return_comparison(strip_expr(db_info->expr), ret_str);
671                 __add_return_to_param_mapping(db_info->expr, ret_str);
672                 store_return_state(db_info, ret_str, alloc_estate_rl(clone_rl(va
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
673         }

675         FOR_EACH_PTR(db_info->callbacks, tmp) {
676                 if (tmp->type == type)
677                         tmp->callback(db_info->expr, param, key, value);
678         } END_FOR_EACH_PTR(tmp);

680         return 0;
681 }

683 static void compare_db_return_states_callbacks(struct expression *left, int comp
684 {
685         struct stree *orig_states;
686         struct stree *stree;
687         struct stree *true_states;
688         struct stree *false_states;
689         struct sm_state *sm;
690         struct db_callback_info db_info = {};
691         struct expression *var_expr;
692         struct expression *call_expr;
693         struct range_list *rl;
694         int call_on_left;

696         orig_states = clone_stree(__get_cur_stree());

698         /* legacy cruft.  need to fix call_implies_callbacks(). */
699         call_on_left = 1;
700         call_expr = left;
701         var_expr = right;
702         if (left->type != EXPR_CALL) {
703                 call_on_left = 0;
704                 call_expr = right;
705                 var_expr = left;
706         }

708         get_absolute_rl(var_expr, &rl);
```



```
710         db_info.comparison = comparison;
711         db_info.expr = call_expr;
712         db_info.rl = rl;
713         db_info.left = call_on_left;
714         db_info.callbacks = db_return_states_list;
715         db_info.var_expr = var_expr;

717         call_return_states_before_hooks();

719         db_info.true_side = 1;
720         db_info.stree = NULL;
721         db_info.prev_return_id = -1;
722         __push_fake_cur_stree();
723         sql_select_return_states("return_id, return, type, parameter, key, value
724                                  call_expr, db_compare_callback, &db_info);
725         set_other_side_state(&db_info);
690         set_return_state(db_info.var_expr, &db_info);
726         stree = __pop_fake_cur_stree();
727         if (!db_info.cull)
692         if (!db_info.cull) {
693                 set_return_state(db_info.var_expr, &db_info);
728                 merge_fake_stree(&db_info.stree, stree);
695         }
729         free_stree(&stree);
730         true_states = db_info.stree;
731         if (!true_states && db_info.has_states) {
732                 __push_fake_cur_stree();
733                 set_path_impossible();
734                 true_states = __pop_fake_cur_stree();
735         }

737         nullify_path();
738         __unnullify_path();
739         FOR_EACH_SM(orig_states, sm) {
740                 __set_sm_cur_stree(sm);
741         } END_FOR_EACH_SM(sm);

743         db_info.true_side = 0;
744         db_info.stree = NULL;
745         db_info.prev_return_id = -1;
746         db_info.cull = 0;
747         __push_fake_cur_stree();
748         sql_select_return_states("return_id, return, type, parameter, key, value
749                                  db_compare_callback, &db_info);
750         set_other_side_state(&db_info);
751         stree = __pop_fake_cur_stree();
752         if (!db_info.cull)
718         if (!db_info.cull) {
719                 set_return_state(db_info.var_expr, &db_info);
753                 merge_fake_stree(&db_info.stree, stree);
721         }
754         free_stree(&stree);
755         false_states = db_info.stree;
756         if (!false_states && db_info.has_states) {
757                 __push_fake_cur_stree();
758                 set_path_impossible();
759                 false_states = __pop_fake_cur_stree();
760         }

762         nullify_path();
763         __unnullify_path();
764         FOR_EACH_SM(orig_states, sm) {
765                 __set_sm_cur_stree(sm);
766         } END_FOR_EACH_SM(sm);
```

```
768         free_stree(&orig_states);

770         FOR_EACH_SM(true_states, sm) {
771                 __set_true_false_sm(sm, NULL);
772         } END_FOR_EACH_SM(sm);
773         FOR_EACH_SM(false_states, sm) {
774                 __set_true_false_sm(NULL, sm);
775         } END_FOR_EACH_SM(sm);

777         free_stree(&true_states);
778         free_stree(&false_states);

780         call_return_states_after_hooks(call_expr);

782         FOR_EACH_SM(implied_true, sm) {
783                 __set_true_false_sm(sm, NULL);
784         } END_FOR_EACH_SM(sm);
785         FOR_EACH_SM(implied_false, sm) {
786                 __set_true_false_sm(NULL, sm);
787         } END_FOR_EACH_SM(sm);
788 }
______unchanged_portion_omitted_

823 static void call_ranged_return_hooks(struct db_callback_info *db_info)
824 {
825         struct call_back_list *call_backs;
826         struct expression *expr;
827         struct fcall_back *tmp;
828         char *fn;

830         expr = strip_expr(db_info->expr);
831         while (expr->type == EXPR_ASSIGNMENT)
832                 expr = strip_expr(expr->right);
833         if (expr->type != EXPR_CALL ||
834             expr->fn->type != EXPR_SYMBOL)
835                 return;

837         fn = expr->fn->symbol_name->name;

839         call_backs = search_callback(func_hash, fn);
840         FOR_EACH_PTR(call_backs, tmp) {
841                 struct range_list *range_rl;
809                 struct range_list *range_rl = NULL;

843                 if (tmp->type != RANGED_CALL)
844                         continue;
845                 range_rl = alloc_rl(tmp->range->min, tmp->range->max);
813                 add_range(&range_rl, tmp->range->min, tmp->range->max);
846                 range_rl = cast_rl(estate_type(db_info->ret_state), range_rl);
847                 if (possibly_true_rl(range_rl, SPECIAL_EQUAL, estate_rl(db_info-
815                 if (possibly_true_rl(range_rl, SPECIAL_EQUAL, estate_rl(db_info-
816                         if (!possibly_true_rl(rl_invert(range_rl), SPECIAL_EQUAL
848                         (tmp->u.ranged)(fn, expr, db_info->expr, tmp->info);
818                         else
819                                 db_info->handled = -1;
820                 }
849         } END_FOR_EACH_PTR(tmp);
850 }

852 static int db_assign_return_states_callback(void *_info, int argc, char **argv,
853 {
854         struct db_callback_info *db_info = _info;
855         struct range_list *ret_range;
856         int type, param;
857         char *ret_str, *key, *value;
829         char *key, *value;
```



```
858         struct return_implies_callback *tmp;
859         struct stree *stree;
860         int return_id;

862         if (argc != 6)
863                 return 0;

865         return_id = atoi(argv[0]);
866         ret_str = argv[1];
867         type = atoi(argv[2]);
868         param = atoi(argv[3]);
869         key = argv[4];
870         value = argv[5];

872         if (db_info->prev_return_id != -1 && type == INTERNAL) {
873                 call_ranged_return_hooks(db_info);
874                 set_return_assign_state(db_info);
845                 set_return_state(db_info->expr->left, db_info);
875                 stree = __pop_fake_cur_stree();
876                 if (!db_info->cull)
877                         merge_fake_stree(&db_info->stree, stree);
878                 free_stree(&stree);
879                 __push_fake_cur_stree();
880                 db_info->cull = 0;
881         }
882         db_info->prev_return_id = return_id;

884         if (type == INTERNAL && func_type_mismatch(db_info->expr, value))
885                 db_info->cull = 1;
886         if (db_info->cull)
887                 return 0;
888         if (type == CULL_PATH) {
889                 db_info->cull = 1;
890                 return 0;
891         }
892         if (is_impossible_data(type, db_info->expr, param, key, value)) {
893                 db_info->cull = 1;
894                 return 0;
895         }

897         if (type == PARAM_LIMIT)
898                 param_limit_implications(db_info->expr, param, key, value);

900         db_info->handled = 1;
901         call_results_to_rl(db_info->expr->right, get_type(strip_expr(db_info->ex
872         call_results_to_rl(db_info->expr->right, get_type(strip_expr(db_info->ex
902         if (!ret_range)
903                 ret_range = alloc_whole_rl(get_type(strip_expr(db_info->expr->ri
904         ret_range = cast_rl(get_type(db_info->expr->right), ret_range);

906         if (type == INTERNAL) {
907                 set_state(-1, "unnull_path", NULL, &true_state);
908                 __add_return_comparison(strip_expr(db_info->expr->right), ret_st
909                 __add_comparison_info(db_info->expr->left, strip_expr(db_info->e
910                 __add_return_to_param_mapping(db_info->expr, ret_str);
911                 store_return_state(db_info, ret_str, alloc_estate_rl(ret_range))
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))
912         }

914         FOR_EACH_PTR(db_return_states_list, tmp) {
915                 if (tmp->type == type)
916                         tmp->callback(db_info->expr, param, key, value);
917         } END_FOR_EACH_PTR(tmp);
```

```
 919         return 0;
 920 }

 922 static int db_return_states_assign(struct expression *expr)
 923 {
 924         struct expression *right;
 925         struct sm_state *sm;
 926         struct stree *stree;
 927         struct db_callback_info db_info = {};

 929         right = strip_expr(expr->right);

 931         db_info.prev_return_id = -1;
 932         db_info.expr = expr;
 933         db_info.stree = NULL;
 934         db_info.handled = 0;

 936         call_return_states_before_hooks();

 938         __push_fake_cur_stree();
 939         sql_select_return_states("return_id, return, type, parameter, key, value
 940                         right, db_assign_return_states_callback, &db_info);
 941         if (option_debug) {
 942                 sm_msg("%s return_id %d return_ranges %s",
 943                         db_info.cull ? "culled" : "merging",
 944                         db_info.prev_return_id,
 945                         db_info.ret_state ? db_info.ret_state->name : "'<empty>'
 946         }
 947         if (db_info.handled)
 948                 call_ranged_return_hooks(&db_info);
 949         set_return_assign_state(&db_info);
 920         set_return_state(db_info.expr->left, &db_info);
 950         stree = __pop_fake_cur_stree();
 951         if (!db_info.cull)
 952                 merge_fake_stree(&db_info.stree, stree);
 953         free_stree(&stree);

 955         if (!db_info.stree && db_info.cull) { /* this means we culled everything
 956                 set_extra_expr_mod(expr->left, alloc_estate_whole(get_type(expr-
 957                 set_path_impossible();
 958         }
 959         FOR_EACH_SM(db_info.stree, sm) {
 960                 __set_sm(sm);
 961         } END_FOR_EACH_SM(sm);

 963         free_stree(&db_info.stree);
 964         call_return_states_after_hooks(right);

 966         return db_info.handled;
 967 }
______unchanged_portion_omitted_

1038 static int db_return_states_callback(void *_info, int argc, char **argv, char **
1039 {
1040         struct db_callback_info *db_info = _info;
1041         struct range_list *ret_range;
1042         int type, param;
1043         char *ret_str, *key, *value;
1014         char *key, *value;
1044         struct return_implies_callback *tmp;
1045         struct stree *stree;
1046         int return_id;
1047         char buf[64];

1049         if (argc != 6)
```



```
1050                 return 0;

1052         return_id = atoi(argv[0]);
1053         ret_str = argv[1];
1054         type = atoi(argv[2]);
1055         param = atoi(argv[3]);
1056         key = argv[4];
1057         value = argv[5];

1059         if (db_info->prev_return_id != -1 && type == INTERNAL) {
1060                 stree = __pop_fake_cur_stree();
1061                 if (!db_info->cull)
1062                         merge_fake_stree(&db_info->stree, stree);
1063                 free_stree(&stree);
1064                 __push_fake_cur_stree();
1065                 __unnullify_path();
1066                 db_info->cull = 0;
1067         }
1068         db_info->prev_return_id = return_id;

1070         if (type == INTERNAL && func_type_mismatch(db_info->expr, value))
1071                 db_info->cull = 1;
1072         if (db_info->cull)
1073                 return 0;
1074         if (type == CULL_PATH) {
1075                 db_info->cull = 1;
1076                 return 0;
1077         }
1078         if (is_impossible_data(type, db_info->expr, param, key, value)) {
1079                 db_info->cull = 1;
1080                 return 0;
1081         }

1083         if (type == PARAM_LIMIT)
1084                 param_limit_implications(db_info->expr, param, key, value);

1086         call_results_to_rl(db_info->expr, get_type(strip_expr(db_info->expr)), r
1056         call_results_to_rl(db_info->expr, get_type(strip_expr(db_info->expr)), a
1087         ret_range = cast_rl(get_type(db_info->expr), ret_range);

1089         if (type == INTERNAL) {
1090                 set_state(-1, "unnull_path", NULL, &true_state);
1091                 __add_return_comparison(strip_expr(db_info->expr), ret_str);
1092                 __add_return_to_param_mapping(db_info->expr, ret_str);
1061                 __add_return_comparison(strip_expr(db_info->expr), argv[1]);
1062                 __add_return_to_param_mapping(db_info->expr, argv[1]);
1093         }


1096         FOR_EACH_PTR(db_return_states_list, tmp) {
1097                 if (tmp->type == type)
1098                         tmp->callback(db_info->expr, param, key, value);
1099         } END_FOR_EACH_PTR(tmp);

1101         /*
1102          * We want to store the return values so that we can split the strees
1103          * in smatch_db.c.  This uses set_state() directly because it's not a
1104          * real smatch_extra state.
1105          */
1106         snprintf(buf, sizeof(buf), "return %p", db_info->expr);
1107         set_state(SMATCH_EXTRA, buf, NULL, alloc_estate_rl(ret_range));

1109         return 0;
1110 }
______unchanged_portion_omitted_
```

```
**********************************************************
    9444 Mon Aug  5 08:38:32 2019
new/usr/src/tools/smatch/src/smatch_function_ptrs.c
11506 smatch resync
**********************************************************
```

```
______unchanged_portion_omitted_

 140 char *get_fnptr_name(struct expression *expr)
 141 {
 142         char *name;

 144         if (is_zero(expr))
 145                 return NULL;

 147         expr = strip_expr(expr);

 149         /* (*ptrs[0])(a, b, c) is the same as ptrs[0](a, b, c); */
 150         if (expr->type == EXPR_PREOP && expr->op == '*')
 151                 expr = strip_expr(expr->unop);

 153         name = get_from__symbol_get(expr);
 154         if (name)
 155                 return name;

 157         name = get_array_ptr(expr);
 158         if (name)
 159                 return name;

 161         name = get_returned_ptr(expr);
 162         if (name)
 163                 return name;

 165         name = get_member_name(expr);
 166         if (name)
 167                 return name;

 169         if (expr->type == EXPR_SYMBOL) {
 170                 int param;
 171                 char buf[256];
 172                 struct symbol *sym;
 173                 struct symbol *type;

 175                 param = get_param_num_from_sym(expr->symbol);
 176                 if (param >= 0) {
 177                         snprintf(buf, sizeof(buf), "%s param %d", get_function()
 178                         return alloc_string(buf);
 179                 }

 181                 name =  expr_to_var_sym(expr, &sym);
 182                 if (!name)
 183                         return NULL;
 184                 type = get_type(expr);
 185                 if (type && type->type == SYM_PTR) {
 186                         snprintf(buf, sizeof(buf), "%s %s", ptr_prefix(sym), nam
 187                         free_string(name);
 188                         return alloc_string(buf);
 189                 }
 190                 return name;
 191         }
 192         return expr_to_var(expr);
 193 }
______unchanged_portion_omitted_

 351 static void print_initializer_list(struct expression_list *expr_list,
 352                 struct symbol *struct_type)
 353 {
```



```
 354         struct expression *expr;
 355         struct symbol *base_type;
 356         char struct_name[256];

 358         FOR_EACH_PTR(expr_list, expr) {
 359                 if (expr->type == EXPR_INDEX && expr->idx_expression && expr->id
 360                         print_initializer_list(expr->idx_expression->expr_list,
 361                         continue;
 362                 }
 363                 if (expr->type != EXPR_IDENTIFIER)
 364                         continue;
 365                 if (!expr->expr_ident)
 366                         continue;
 367                 if (!expr->ident_expression ||
 368                     expr->ident_expression->type != EXPR_SYMBOL ||
 369                     !expr->ident_expression->symbol_name)
 370                         continue;
 371                 base_type = get_type(expr->ident_expression);
 372                 if (!base_type || base_type->type != SYM_FN)
 373                         continue;
 374                 snprintf(struct_name, sizeof(struct_name), "(struct %s)->%s",
 375                          struct_type->ident->name, expr->expr_ident->name);
 376                 sql_insert_function_ptr(expr->ident_expression->symbol_name->nam
 377                                         struct_name);
 378         } END_FOR_EACH_PTR(expr);
 379 }

 381 static void global_variable(struct symbol *sym)
 382 {
 383         struct symbol *struct_type;

 385         if (!sym->ident)
 386                 return;
 387         if (!sym->initializer || sym->initializer->type != EXPR_INITIALIZER)
 388                 return;
 389         struct_type = get_base_type(sym);
 390         if (!struct_type)
 391                 return;
 392         if (struct_type->type == SYM_ARRAY) {
 393                 struct_type = get_base_type(struct_type);
 394                 if (!struct_type)
 395                         return;
 396         }
 397         if (struct_type->type != SYM_STRUCT || !struct_type->ident)
 398                 return;
 399         print_initializer_list(sym->initializer->expr_list, struct_type);
 400 }

 402 void register_function_ptrs(int id)
 403 {
 404         my_id = id;

 406         if (!option_info)
 407                 return;

 409         add_hook(&global_variable, BASE_HOOK);
 410         add_hook(&global_variable, DECLARATION_HOOK);
 411         add_hook(&match_passes_function_pointer, FUNCTION_CALL_HOOK);
 412         add_hook(&match_returns_function_pointer, RETURN_HOOK);
 413         add_hook(&match_function_assign, ASSIGNMENT_HOOK);
 414         add_hook(&match_function_assign, GLOBAL_ASSIGNMENT_HOOK);
 415 }
______unchanged_portion_omitted_
```

```
**********************************************************
   25477 Mon Aug  5 08:38:33 2019
new/usr/src/tools/smatch/src/smatch_helper.c
11506 smatch resync
**********************************************************
______unchanged_portion_omitted_

 84 struct smatch_state *merge_str_state(struct smatch_state *s1, struct smatch_stat
 85 {
 86         if (!s1->name || !s2->name)
 87                 return &merged;
 88         if (strcmp(s1->name, s2->name) == 0)
 89                 return s1;
 90         return &merged;
 91 }

 93 struct smatch_state *alloc_state_expr(struct expression *expr)
 94 {
 95         struct smatch_state *state;
 96         char *name;

 89         state = __alloc_smatch_state(0);
 98         expr = strip_expr(expr);
 99         name = expr_to_str(expr);
100         if (!name)
101                 return NULL;

103         state = __alloc_smatch_state(0);
104         state->name = alloc_sname(name);
105         free_string(name);
106         state->data = expr;
107         return state;
108 }
______unchanged_portion_omitted_

162 static void __get_variable_from_expr(struct symbol **sym_ptr, char *buf,
163                                       struct expression *expr, int len,
164                                       int *complicated, int no_parens)
165 {



168         if (!expr) {
169                 /* can't happen on valid code */
170                 *complicated = 1;
171                 return;
172         }

174         switch (expr->type) {
175         case EXPR_DEREF: {
176                 struct expression *deref;
177                 int op;

179                 deref = expr->deref;
180                 op = deref->op;
181                 if (deref->type == EXPR_PREOP && op == '*') {
169                 if (op == '*') {
182                         struct expression *unop = strip_expr(deref->unop);

184                         if (unop->type == EXPR_PREOP && unop->op == '&') {
185                                 deref = unop->unop;
186                                 op = '.';
187                         } else {
188                                 if (!is_pointer(deref) && !is_pointer(deref->uno
189                                         op = '.';
190                                 deref = deref->unop;
177                                 if (!is_pointer(deref))
```



```
178                                         op = '.';
191                         }
192                 }

194                 __get_variable_from_expr(sym_ptr, buf, deref, len, complicated,

196                 if (op == '*')
197                         append(buf, "->", len);
198                 else
199                         append(buf, ".", len);

201                 if (expr->member)
202                         append(buf, expr->member->name, len);
203                 else
204                         append(buf, "unknown_member", len);

206                 return;
207         }
208         case EXPR_SYMBOL:
209                 if (expr->symbol_name)
210                         append(buf, expr->symbol_name->name, len);
211                 if (sym_ptr) {
212                         if (*sym_ptr)
213                                 *complicated = 1;
214                         *sym_ptr = expr->symbol;
215                 }
216                 return;
217         case EXPR_PREOP: {
218                 const char *tmp;

220                 if (get_expression_statement(expr)) {
221                         *complicated = 2;
222                         return;
223                 }

225                 if (expr->op == '(') {
226                         if (!no_parens && expr->unop->type != EXPR_SYMBOL)
227                                 append(buf, "(", len);
228                 } else if (expr->op != '*' || !get_array_expr(expr->unop)) {
229                         tmp = show_special(expr->op);
230                         append(buf, tmp, len);
231                 }
232                 __get_variable_from_expr(sym_ptr, buf, expr->unop,
233                                                 len, complicated, no_parens);

235                 if (expr->op == '(' && !no_parens && expr->unop->type != EXPR_SY
236                         append(buf, ")", len);

238                 if (expr->op == SPECIAL_DECREMENT ||
239                                 expr->op == SPECIAL_INCREMENT)
240                         *complicated = 1;

242                 return;
243         }
244         case EXPR_POSTOP: {
245                 const char *tmp;

247                 __get_variable_from_expr(sym_ptr, buf, expr->unop,
248                                                 len, complicated, no_parens);
249                 tmp = show_special(expr->op);
250                 append(buf, tmp, len);

252                 if (expr->op == SPECIAL_DECREMENT || expr->op == SPECIAL_INCREME
253                         *complicated = 1;
254                 return;
255         }
```

```
256         case EXPR_ASSIGNMENT:
257         case EXPR_COMPARE:
258         case EXPR_LOGICAL:
259         case EXPR_BINOP: {
260                 char tmp[10];
261                 struct expression *array_expr;

263                 *complicated = 1;
264                 array_expr = get_array_expr(expr);
265                 if (array_expr) {
266                         __get_variable_from_expr(sym_ptr, buf, array_expr, len,
267                         append(buf, "[", len);
268                 } else {
269                         __get_variable_from_expr(sym_ptr, buf, expr->left, len,
270                         snprintf(tmp, sizeof(tmp), " %s ", show_special(expr->op
271                         append(buf, tmp, len);
272                 }
273                 __get_variable_from_expr(NULL, buf, expr->right, len, complicate
274                 if (array_expr)
275                         append(buf, "]", len);
276                 return;
277         }
278         case EXPR_VALUE: {
279                 char tmp[25];

281                 *complicated = 1;
282                 snprintf(tmp, 25, "%lld", expr->value);
283                 append(buf, tmp, len);
284                 return;
285         }
286         case EXPR_STRING:
287                 append(buf, "\"", len);
288                 if (expr->string)
289                         append(buf, expr->string->data, len);
290                 append(buf, "\"", len);
291                 return;
292         case EXPR_CALL: {
293                 struct expression *tmp;
294                 int i;

296                 *complicated = 1;
297                 __get_variable_from_expr(NULL, buf, expr->fn, len, complicated,
298                 append(buf, "(", len);
299                 i = 0;
300                 FOR_EACH_PTR(expr->args, tmp) {
301                         if (i++)
302                                 append(buf, ", ", len);
303                         __get_variable_from_expr(NULL, buf, tmp, len, complicate
304                 } END_FOR_EACH_PTR(tmp);
305                 append(buf, ")", len);
306                 return;
307         }
308         case EXPR_CAST:
309         case EXPR_FORCE_CAST:
310                 __get_variable_from_expr(sym_ptr, buf,
311                                          expr->cast_expression, len,
312                                          complicated, no_parens);
313                 return;
314         case EXPR_SIZEOF: {
315                 sval_t sval;
316                 int size;
317                 char tmp[25];

319                 if (expr->cast_type && get_base_type(expr->cast_type)) {
320                         size = type_bytes(get_base_type(expr->cast_type));
321                         snprintf(tmp, 25, "%d", size);
```



```
322                         append(buf, tmp, len);
323                 } else if (get_value(expr, &sval)) {
324                         snprintf(tmp, 25, "%s", sval_to_str(sval));
325                         append(buf, tmp, len);
326                 }
327                 return;
328         }
329         case EXPR_IDENTIFIER:
330                 *complicated = 1;
331                 if (expr->expr_ident)
332                         append(buf, expr->expr_ident->name, len);
333                 return;
334         default:
335                 *complicated = 1;
336                 //printf("unknown type = %d\n", expr->type);
337                 return;
338         }
339 }
```

_____***unchanged_portion_omitted_***

```
521 char *expr_to_chunk_helper(struct expression *expr, struct symbol **sym, struct
522 {
523         struct var_sym_list *tmp_vsl;
524         char *name;
525         struct symbol *tmp;
526         int score;

528         if (vsl)
529                 *vsl = NULL;
530         if (sym)
531                 *sym = NULL;

533         expr = strip_parens(expr);
534         if (!expr)
535                 return NULL;

537         name = expr_to_var_sym(expr, &tmp);
538         if (name && tmp) {
539                 if (sym)
540                         *sym = tmp;
541                 if (vsl)
542                         add_var_sym(vsl, name, tmp);
530                         *vsl = expr_to_vsl(expr);
543                 return name;
544         }
545         free_string(name);

547         score = get_complication_score(expr);
548         if (score <= 0 || score > 2)
549                 return NULL;

551         tmp_vsl = expr_to_vsl(expr);
552         if (vsl) {
553                 *vsl = tmp_vsl;
554                 if (!*vsl)
555                         return NULL;
556         }
557         if (sym) {
558                 if (ptr_list_size((struct ptr_list *)tmp_vsl) == 1) {
559                         struct var_sym *vs;

561                         vs = first_ptr_list((struct ptr_list *)tmp_vsl);
562                         *sym = vs->sym;
563                 }
564         }
```

```
 566         expr = reorder_expr_alphabetically(expr);

 568         return expr_to_str(expr);
 569 }
_____unchanged_portion_omitted_

 864 char *get_member_name(struct expression *expr)
 865 {
 866         char buf[256];
 867         struct symbol *sym;

 869         expr = strip_expr(expr);
 870         if (!expr || expr->type != EXPR_DEREF)
 871                 return NULL;
 872         if (!expr->member)
 873                 return NULL;

 875         sym = get_type(expr->deref);
 876         if (!sym)
 877                 return NULL;
 878         if (sym->type == SYM_UNION) {
 879                 snprintf(buf, sizeof(buf), "(union %s)->%s",
 880                          sym->ident ? sym->ident->name : "anonymous",
 881                          expr->member->name);
 882                 return alloc_string(buf);
 883         }
 884         if (!sym->ident) {
 885                 struct expression *deref;
 886                 char *full, *outer;
 887                 int len;

 889                 /*
 890                  * If we're in an anonymous struct then maybe we can find an
 891                  * outer struct name to use as a name.  This code should be
 892                  * recursive and cleaner.  I am not very proud of it.
 893                  *
 894                  */

 896                 deref = expr->deref;
 897                 if (deref->type != EXPR_DEREF || !deref->member)
 872         if (!sym->ident)
 898                         return NULL;
 899                 sym = get_type(deref->deref);
 900                 if (!sym || sym->type != SYM_STRUCT || !sym->ident)
 901                         return NULL;

 903                 full = expr_to_str(expr);
 904                 if (!full)
 905                         return NULL;
 906                 deref = deref->deref;
 907                 if (deref->type == EXPR_PREOP && deref->op == '*')
 908                         deref = deref->unop;
 909                 outer = expr_to_str(deref);
 910                 if (!outer) {
 911                         free_string(full);
 912                         return NULL;
 913                 }
 914                 len = strlen(outer);
 915                 if (strncmp(outer, full, len) != 0) {
 916                         free_string(full);
 917                         free_string(outer);
 918                         return NULL;
 919                 }
 920                 if (full[len] == '-' && full[len + 1] == '>')
 921                         len += 2;
 922                 if (full[len] == '.')
```



```
 923                         len++;
 924                 snprintf(buf, sizeof(buf), "(struct %s)->%s", sym->ident->name,
 925                 free_string(outer);
 926                 free_string(full);

 928                 return alloc_string(buf);
 929         }
 930         snprintf(buf, sizeof(buf), "(struct %s)->%s", sym->ident->name, expr->me
 931         return alloc_string(buf);
 932 }
_____unchanged_portion_omitted_

1113 int op_remove_assign(int op)
1114 {
1115         switch (op) {
1116         case SPECIAL_ADD_ASSIGN:
1117                 return '+';
1118         case SPECIAL_SUB_ASSIGN:
1119                 return '-';
1120         case SPECIAL_MUL_ASSIGN:
1121                 return '*';
1122         case SPECIAL_DIV_ASSIGN:
1123                 return '/';
1124         case SPECIAL_MOD_ASSIGN:
1125                 return '%';
1126         case SPECIAL_AND_ASSIGN:
1127                 return '&';
1128         case SPECIAL_OR_ASSIGN:
1129                 return '|';
1130         case SPECIAL_XOR_ASSIGN:
1131                 return '^';
1132         case SPECIAL_SHL_ASSIGN:
1133                 return SPECIAL_LEFTSHIFT;
1134         case SPECIAL_SHR_ASSIGN:
1135                 return SPECIAL_RIGHTSHIFT;
1136         default:
1137                 return op;
1138         }
1139 }

1141 int expr_equiv(struct expression *one, struct expression *two)
1142 {
1143         struct symbol *one_sym = NULL;
1144         struct symbol *two_sym = NULL;
1145         char *one_name = NULL;
1146         char *two_name = NULL;
1147         int ret = 0;

1149         if (!one || !two)
1150                 return 0;
1151         if (one->type != two->type)
1152                 return 0;
1153         if (is_fake_call(one) || is_fake_call(two))
1154                 return 0;

1156         one_name = expr_to_str_sym(one, &one_sym);
1157         if (!one_name)
1158                 goto free;
1159         two_name = expr_to_str_sym(two, &two_sym);
1160         if (!two_name)
1161                 goto free;
1162         if (one_sym != two_sym)
1163                 goto free;
1164         /*
1165          * This is a terrible hack because expr_to_str() sometimes gives up in
1166          * the middle and just returns what it has.  If you see a () you know
```

```
1167          * the string is bogus.
1168          */
1169         if (strstr(one_name, "()"))
1170                 goto free;
1171         if (strcmp(one_name, two_name) == 0)
1172                 ret = 1;
1173 free:
1174         free_string(one_name);
1175         free_string(two_name);
1176         return ret;
1177 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    2687 Mon Aug  5 08:38:34 2019
new/usr/src/tools/smatch/src/smatch_ignore.c
11506 smatch resync
**********************************************************
```

```
/*
 * 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;
static struct stree *ignored_from_file;

void add_ignore(int owner, const char *name, struct symbol *sym)
{
        set_state_stree(&ignored, owner, name, sym, &ignore);
}
```

*_____unchanged_portion_omitted_*

```
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);
        if (ret)
                return true;

        name = get_macro_name(expr->pos);
        if (name && get_state_stree(ignored_from_file, owner, name, NULL))
                return true;

        name = get_function();
        if (name && get_state_stree(ignored_from_file, owner, name, NULL))
                return true;

        return false;
        return ret;
}
```

*_____unchanged_portion_omitted_*

```
static void load_ignores(void)
{
        struct token *token;
        const char *name, *str;
```



```
        int owner;
        char buf[64];

        snprintf(buf, sizeof(buf), "%s.ignored_warnings", option_project_str);
        token = get_tokens_file(buf);
        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;
                name = show_ident(token->ident);
                token = token->next;
                owner = id_from_name(name);

                if (token_type(token) != TOKEN_IDENT)
                        break;
                str = show_ident(token->ident);
                token = token->next;

                set_state_stree_perm(&ignored_from_file, owner, str, NULL, &igno
        }
        clear_token_alloc();
}

void register_smatch_ignore(int id)
{
        add_hook(&clear_ignores, AFTER_FUNC_HOOK);
        load_ignores();
}
```

*_____unchanged_portion_omitted_*

****
2250 Mon Aug 5 08:38:34 2019
new/usr/src/tools/smatch/src/smatch_imaginary_absolute.c
11506 smatch resync
****

_____unchanged_portion_omitted_

```
 54 void __save_imaginary_state(struct expression *expr, struct range_list *true_rl,
 55 {
 56         if (__in_pre_condition)
 57                 return;
 58         set_true_false_states_expr(my_id, expr, alloc_estate_rl(true_rl), alloc_
 59 }
```

_____unchanged_portion_omitted_

```
 75 void register_imaginary_absolute(int id)
 76 {
 77         my_id = id;

 79         set_dynamic_states(my_id);
 80         add_unmatched_state_hook(my_id, &empty_state);
 81         add_merge_hook(my_id, &merge_is_empty);
 82         add_modification_hook(my_id, &reset);
 83 }
```

_____unchanged_portion_omitted_

```
**********************************************************
   30977 Mon Aug  5 08:38:35 2019
new/usr/src/tools/smatch/src/smatch_implied.c
11506 smatch resync
**********************************************************
```

```
/*
 * 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
 *
 * Copyright 2019 Joyent, Inc.
 */

/*
 * 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

bool implications_off;
int option_debug_implied = 0;

#define implied_debug 0
#define DIMPLIED(msg...) do { if (implied_debug) printf(msg); } while (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 (!implied_debug && !option_debug)
        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))
```

```
125                 return 0;
126         if (!estate_rl(sm->state))
127                 return 0;

129         orig_rl = cast_rl(rl_type(rl), estate_rl(sm->state));
130         split_comparison_rl(orig_rl, comparison, rl, &true_rl, &false_rl, NULL,

132         true_rl = rl_truncate_cast(estate_type(sm->state), true_rl);
133         false_rl = rl_truncate_cast(estate_type(sm->state), false_rl);
134         if (is_whole_rl(true_rl) || is_whole_rl(false_rl) ||
135             !true_rl || !false_rl ||
136             rl_equiv(orig_rl, true_rl) || rl_equiv(orig_rl, false_rl) ||
137             rl_equiv(estate_rl(sm->state), true_rl) || rl_equiv(estate_rl(sm->st
138                 return 0;

140         if (rl_intersection(true_rl, false_rl)) {
141                 sm_perror("parsing (%s (%s) %s %s)",
142                           sm->name, sm->state->name, show_special(comparison), sho
143                 sm_msg("true_rl = %s false_rl = %s intersection = %s",
144                        show_rl(true_rl), show_rl(false_rl), show_rl(rl_intersect
145                 return 0;
146         }

148         if (implied_debug)
149                 sm_msg("fake_history: %s vs %s.  %s %s %s. --> T: %s F: %s",
146         if (option_debug)
147                 sm_info("fake_history: %s vs %s.  %s %s %s. --> T: %s F: %s",
150                        sm->name, show_rl(rl), sm->state->name, show_special(comp
151                        show_rl(true_rl), show_rl(false_rl));

153         true_sm = clone_sm(sm);
154         false_sm = clone_sm(sm);

156         true_sm->state = clone_partial_estate(sm->state, true_rl);
154         true_sm->state = alloc_estate_rl(cast_rl(estate_type(sm->state), true_rl
157         free_slist(&true_sm->possible);
158         add_possible_sm(true_sm, true_sm);
159         false_sm->state = clone_partial_estate(sm->state, false_rl);
157         false_sm->state = alloc_estate_rl(cast_rl(estate_type(sm->state), false_
160         free_slist(&false_sm->possible);
161         add_possible_sm(false_sm, false_sm);

163         true_stree = clone_stree(sm->pool);
164         false_stree = clone_stree(sm->pool);

166         overwrite_sm_state_stree(&true_stree, true_sm);
167         overwrite_sm_state_stree(&false_stree, false_sm);

169         true_sm->pool = true_stree;
170         false_sm->pool = false_stree;

172         sm->merged = 1;
173         sm->left = true_sm;
174         sm->right = false_sm;

176         return 1;
177 }
```

*unchanged_portion_omitted_*

```
229 /*
230  * If 'foo' == 99 add it that pool to the true pools.  If it's false, add it to
231  * the false pools.  If we're not sure, then we don't add it to either.
232  */
233 static void do_compare(struct sm_state *sm, int comparison, struct range_list *r
234                        struct state_list **true_stack,
235                        struct state_list **maybe_stack,
```



```
236                        struct state_list **false_stack,
237                        int *mixed, struct sm_state *gate_sm)
238 {
239         int istrue;
240         int isfalse;
241         struct range_list *var_rl;

243         if (!sm->pool)
244                 return;

246         var_rl = cast_rl(rl_type(rl), estate_rl(sm->state));

248         istrue = !possibly_false_rl(var_rl, comparison, rl);
249         isfalse = !possibly_true_rl(var_rl, comparison, rl);

251         print_debug_tf(sm, istrue, isfalse);

253         /* give up if we have borrowed implications (smatch_equiv.c) */
254         if (sm->sym != gate_sm->sym ||
255             strcmp(sm->name, gate_sm->name) != 0) {
256                 if (mixed)
257                         *mixed = 1;
258         }

260         if (mixed && !*mixed && !is_merged(sm) && !istrue && !isfalse) {
261                 if (!create_fake_history(sm, comparison, rl))
262                         *mixed = 1;
263         }

265         if (istrue)
266                 add_pool(true_stack, sm);
267         else if (isfalse)
268                 add_pool(false_stack, sm);
269         else
270                 add_pool(maybe_stack, sm);

271 }
```

*unchanged_portion_omitted_*

```
284 /*
285  * separate_pools():
286  * Example code:  if (foo == 99) {
287  *
288  * Say 'foo' is a merged state that has many possible values.  It is the combina
289  * of merges.  separate_pools() iterates through the pools recursively and calls
290  * do_compare() for each time 'foo' was set.
291  */
292 static void __separate_pools(struct sm_state *sm, int comparison, struct range_l
293                        struct state_list **true_stack,
294                        struct state_list **maybe_stack,
295                        struct state_list **false_stack,
296                        struct state_list **checked, int *mixed, struct sm_state
297                        struct timeval *start_time)
295                        struct state_list **checked, int *mixed, struct sm_state
298 {
299         int free_checked = 0;
300         struct state_list *checked_states = NULL;
301         struct timeval now;

303         if (!sm)
304                 return;

306         gettimeofday(&now, NULL);
307         if (now.tv_usec - start_time->tv_usec > 1000000) {
308                 if (implied_debug) {
309                         sm_msg("debug: %s: implications taking too long.  (%s %s
```

```
                                __func__, sm->state->name, show_special(compariso
                }
                if (mixed)
        /*
         * If it looks like this is going to take too long as-is, then don't
         * create even more fake history.
         */
        if (mixed && sm->nr_children > 100)
                        *mixed = 1;

        /*
           Sometimes the implications are just too big to deal with
           so we bail.  Theoretically, bailing out here can cause more false
           positives but won't hide actual bugs.
        */
        if (sm->nr_children > 4000) {
                if (option_debug || option_debug_implied) {
                        static char buf[1028];
                        snprintf(buf, sizeof(buf), "debug: %s: nr_children over
                                __func__, sm->nr_children, sm->name, show_state
                        implied_debug_msg = buf;
        }
                return;
        }

        if (checked == NULL) {
                checked = &checked_states;
                free_checked = 1;
        }
        if (is_checked(*checked, sm))
                return;
        add_ptr_list(checked, sm);

        do_compare(sm, comparison, rl, true_stack, maybe_stack, false_stack, mix

        __separate_pools(sm->left, comparison, rl, true_stack, maybe_stack, fals
        __separate_pools(sm->right, comparison, rl, true_stack, maybe_stack, fal
        __separate_pools(sm->left, comparison, rl, true_stack, maybe_stack, fals
        __separate_pools(sm->right, comparison, rl, true_stack, maybe_stack, fal
        if (free_checked)
                free_slist(checked);
}

static void separate_pools(struct sm_state *sm, int comparison, struct range_lis
                        struct state_list **true_stack,
                        struct state_list **false_stack,
                        struct state_list **checked, int *mixed)
{
        struct state_list *maybe_stack = NULL;
        struct sm_state *tmp;
        struct timeval start_time;

        __separate_pools(sm, comparison, rl, true_stack, &maybe_stack, false_sta

        gettimeofday(&start_time, NULL);
        __separate_pools(sm, comparison, rl, true_stack, &maybe_stack, false_sta

        if (implied_debug) {
        if (option_debug) {
                struct sm_state *sm;

                FOR_EACH_PTR(*true_stack, sm) {
                        sm_msg("TRUE %s [stree %d]", show_sm(sm), get_stree_id(s
                } END_FOR_EACH_PTR(sm);

                FOR_EACH_PTR(maybe_stack, sm) {
```



```
                        sm_msg("MAYBE %s %s[stree %d]",
                               show_sm(sm), sm->merged ? "(merged) ": "", get_st
                        sm_msg("MAYBE %s [stree %d]", show_sm(sm), get_stree_id(
                } END_FOR_EACH_PTR(sm);

                FOR_EACH_PTR(*false_stack, sm) {
                        sm_msg("FALSE %s [stree %d]", show_sm(sm), get_stree_id(
                } END_FOR_EACH_PTR(sm);
        }
        /* if it's a maybe then remove it */
        FOR_EACH_PTR(maybe_stack, tmp) {
                remove_pool(false_stack, tmp->pool);
                remove_pool(true_stack, tmp->pool);
        } END_FOR_EACH_PTR(tmp);

        /* if it's both true and false remove it from both */
        FOR_EACH_PTR(*true_stack, tmp) {
                if (remove_pool(false_stack, tmp->pool))
                        DELETE_CURRENT_PTR(tmp);
        } END_FOR_EACH_PTR(tmp);
}
______unchanged_portion_omitted_

static int going_too_slow(void)
static int taking_too_long(void)
{
        static void *printed;

        if (out_of_memory()) {
                implications_off = true;
        if (out_of_memory())
                return 1;
        }

        if (!option_timeout || time_parsing_function() < option_timeout) {
                implications_off = false;
        if (time_parsing_function() < option_timeout)
                return 0;
        }

        if (!__inline_fn && printed != cur_func_sym) {
                if (!is_skipped_function())
                        sm_perror("turning off implications after %d seconds", o
                        sm_perror("turning off implications after 60 seconds");
                printed = cur_func_sym;
        }
        implications_off = true;
        return 1;
}

static char *sm_state_info(struct sm_state *sm)
{
        static char buf[512];
        int n = 0;

        n += snprintf(buf + n, sizeof(buf) - n, "[stree %d line %d] ",
                      get_stree_id(sm->pool),  sm->line);
        if (n >= sizeof(buf))
                return buf;
        n += snprintf(buf + n, sizeof(buf) - n, "%s ", show_sm(sm));
        if (n >= sizeof(buf))
                return buf;
        n += snprintf(buf + n, sizeof(buf) - n, "left = %s [stree %d] ",
                      sm->left ? sm->left->state->name : "<none>",
                      sm->left ? get_stree_id(sm->left->pool) : -1);
        if (n >= sizeof(buf))
```

```
429                 return buf;
430         n += snprintf(buf + n, sizeof(buf) - n, "right = %s [stree %d]",
431                       sm->right ? sm->right->state->name : "<none>",
432                       sm->right ? get_stree_id(sm->right->pool) : -1);
433         return buf;
434 }

436 /*
437  * NOTE: If a state is in both the keep stack and the remove stack then that is
438  * a bug.  Only add states which are definitely true or definitely false.  If
439  * you have a leaf state that could be both true and false, then create a fake
440  * split history where one side is true and one side is false.  Otherwise, if
441  * you can't do that, then don't add it to either list.
442  */
443 #define RECURSE_LIMIT 300
444 struct sm_state *filter_pools(struct sm_state *sm,
445                               const struct state_list *remove_stack,
446                               const struct state_list *keep_stack,
447                               int *modified, int *recurse_cnt,
448                               struct timeval *start, int *skip, int *bail)
424                               struct timeval *start)
449 {
450         struct sm_state *ret = NULL;
451         struct sm_state *left;
452         struct sm_state *right;
453         int removed = 0;
454         struct timeval now;

456         if (!sm)
457                 return NULL;
458         if (*bail)
459                 return NULL;
434         if (sm->skip_implications)
435                 return sm;
436         if (taking_too_long())
437                 return sm;

460         gettimeofday(&now, NULL);
461         if (now.tv_usec - start->tv_usec > 3000000) {
462                 DIMPLIED("%s: implications taking too long: %s\n", __func__, sm_
463                 *bail = 1;
464                 return NULL;
440         if ((*recurse_cnt)++ > 1000 || now.tv_sec - start->tv_sec > 5) {
441                 if (local_debug || option_debug_implied) {
442                         static char buf[1028];
443                         snprintf(buf, sizeof(buf), "debug: %s: nr_children over
444                                  __func__, sm->nr_children, sm->name, show_state
445                         implied_debug_msg = buf;
465         }
466         if ((*recurse_cnt)++ > RECURSE_LIMIT) {
467                 DIMPLIED("%s: recursed too far:  %s\n", __func__, sm_state_info(
468                 *skip = 1;
469                 return NULL;
447                 sm->skip_implications = 1;
448                 return sm;
470         }

472         if (pool_in_pools(sm->pool, remove_stack)) {
473                 DIMPLIED("%s: remove: %s\n", __func__, sm_state_info(sm));
452                 DIMPLIED("removed [stree %d] %s from %d\n", get_stree_id(sm->poo
474                 *modified = 1;
475                 return NULL;
476         }

478         if (!is_merged(sm) || pool_in_pools(sm->pool, keep_stack) || sm_in_keep_
479                 DIMPLIED("%s: keep %s (%s, %s, %s): %s\n", __func__, sm->state->
```



```
458                 DIMPLIED("kept [stree %d] %s from %d. %s. %s. %s.\n", get_stree_
480                         is_merged(sm) ? "merged" : "not merged",
481                         pool_in_pools(sm->pool, keep_stack) ? "not in keep pools
482                         sm_in_keep_leafs(sm, keep_stack) ? "reachable keep leaf"
483                         sm_state_info(sm));
461                         sm_in_keep_leafs(sm, keep_stack) ? "reachable keep leaf"
484                 return sm;
485         }

487         left = filter_pools(sm->left, remove_stack, keep_stack, &removed, recurs
488         right = filter_pools(sm->right, remove_stack, keep_stack, &removed, recu
489         if (*bail || *skip)
490                 return NULL;
465         DIMPLIED("checking [stree %d] %s from %d (%d) left = %s [stree %d] right
466                  get_stree_id(sm->pool),
467                  show_sm(sm), sm->line, sm->nr_children,
468                  sm->left ? sm->left->state->name : "<none>", sm->left ? get_str
469                  sm->right ? sm->right->state->name : "<none>", sm->right ? get_
470         left = filter_pools(sm->left, remove_stack, keep_stack, &removed, recurs
471         right = filter_pools(sm->right, remove_stack, keep_stack, &removed, recu
491         if (!removed) {
492                 DIMPLIED("%s: kept all: %s\n", __func__, sm_state_info(sm));
473                 DIMPLIED("kept [stree %d] %s from %d\n", get_stree_id(sm->pool),
493                 return sm;
494         }
495         *modified = 1;
496         if (!left && !right) {
497                 DIMPLIED("%s: removed all: %s\n", __func__, sm_state_info(sm));
478                 DIMPLIED("removed [stree %d] %s from %d <none>\n", get_stree_id(
498                 return NULL;
499         }

501         if (!left) {
502                 ret = clone_sm(right);
503                 ret->merged = 1;
504                 ret->right = right;
505                 ret->left = NULL;
506         } else if (!right) {
507                 ret = clone_sm(left);
508                 ret->merged = 1;
509                 ret->left = left;
510                 ret->right = NULL;
511         } else {
512                 if (left->sym != sm->sym || strcmp(left->name, sm->name) != 0) {
513                         left = clone_sm(left);
514                         left->sym = sm->sym;
515                         left->name = sm->name;
516                 }
517                 if (right->sym != sm->sym || strcmp(right->name, sm->name) != 0)
518                         right = clone_sm(right);
519                         right->sym = sm->sym;
520                         right->name = sm->name;
521                 }
522                 ret = merge_sm_states(left, right);
523         }

525         ret->pool = sm->pool;

527         DIMPLIED("%s: partial: %s\n", __func__, sm_state_info(sm));
508         DIMPLIED("partial %s => ", show_sm(sm));
509         DIMPLIED("%s from %d [stree %d]\n", show_sm(ret), sm->line, get_stree_id
528         return ret;
529 }

531 static struct stree *filter_stack(struct sm_state *gate_sm,
532                                   struct stree *pre_stree,
```

```
533                                     const struct state_list *remove_stack,
534                                     const struct state_list *keep_stack)
535 {
536         struct stree *ret = NULL;
537         struct sm_state *tmp;
538         struct sm_state *filtered_sm;
539         int modified;
540         int recurse_cnt;
541         struct timeval start;
542         int skip;
543         int bail = 0;

545         if (!remove_stack)
546                 return NULL;

548         gettimeofday(&start, NULL);
528         if (taking_too_long())
529                 return NULL;

549         FOR_EACH_SM(pre_stree, tmp) {
550                 if (!tmp->merged || sm_in_keep_leafs(tmp, keep_stack))
532                 if (option_debug)
533                         sm_msg("%s: %s", __func__, show_sm(tmp));
534                 if (!tmp->merged)
551                         continue;
536                 if (sm_in_keep_leafs(tmp, keep_stack))
537                         continue;
552                 modified = 0;
553                 recurse_cnt = 0;
554                 skip = 0;
555                 filtered_sm = filter_pools(tmp, remove_stack, keep_stack, &modif
556                 if (going_too_slow())
557                         return NULL;
558                 if (bail)
559                         return ret;  /* Return the implications we figured out b


562                 if (skip || !filtered_sm || !modified)
540                 gettimeofday(&start, NULL);
541                 filtered_sm = filter_pools(tmp, remove_stack, keep_stack, &modif
542                 if (!filtered_sm || !modified)
563                         continue;
564                 /* the assignments here are for borrowed implications */
565                 filtered_sm->name = tmp->name;
566                 filtered_sm->sym = tmp->sym;
567                 avl_insert(&ret, filtered_sm);
548                 if (out_of_memory() || taking_too_long())
549                         return NULL;

568         } END_FOR_EACH_SM(tmp);
569         return ret;
570 }

572 static void separate_and_filter(struct sm_state *sm, int comparison, struct rang
573                 struct stree *pre_stree,
574                 struct stree **true_states,
575                 struct stree **false_states,
576                 int *mixed)
577 {
578         struct state_list *true_stack = NULL;
579         struct state_list *false_stack = NULL;
580         struct timeval time_before;
581         struct timeval time_after;
582         int sec;

584         gettimeofday(&time_before, NULL);
```



```
586         DIMPLIED("checking implications: (%s (%s) %s %s)\n",
587                  sm->name, sm->state->name, show_special(comparison), show_rl(rl

589         if (!is_merged(sm)) {
590                 DIMPLIED("%d '%s' from line %d is not merged.\n", get_lineno(),
570                 DIMPLIED("%d '%s' is not merged.\n", get_lineno(), sm->name);
591                 return;
592         }

574         if (option_debug_implied || option_debug) {
575                 sm_msg("checking implications: (%s %s %s)",
576                        sm->name, show_special(comparison), show_rl(rl));
577         }

594         separate_pools(sm, comparison, rl, &true_stack, &false_stack, NULL, mixe

596         DIMPLIED("filtering true stack.\n");
597         *true_states = filter_stack(sm, pre_stree, false_stack, true_stack);
598         DIMPLIED("filtering false stack.\n");
599         *false_states = filter_stack(sm, pre_stree, true_stack, false_stack);
600         free_slist(&true_stack);
601         free_slist(&false_stack);
602         if (implied_debug) {
603                 printf("These are the implied states for the true path: (%s (%s)
604                        sm->name, sm->state->name, show_special(comparison), show
587         if (option_debug_implied || option_debug) {
588                 printf("These are the implied states for the true path: (%s %s %
589                        sm->name, show_special(comparison), show_rl(rl));
605                 __print_stree(*true_states);
606                 printf("These are the implied states for the false path: (%s (%s
607                        sm->name, sm->state->name, show_special(comparison), show
591                 printf("These are the implied states for the false path: (%s %s
592                        sm->name, show_special(comparison), show_rl(rl));
608                 __print_stree(*false_states);
609         }

611         gettimeofday(&time_after, NULL);
612         sec = time_after.tv_sec - time_before.tv_sec;
613         if (option_timeout && sec > option_timeout) {
598         if (sec > option_timeout) {
599                 sm->nr_children = 4000;
614                 sm_perror("Function too hairy.  Ignoring implications after %d s
615         }
616 }
______unchanged_portion_omitted_

817 static int found_implications;
831 static struct stree *saved_implied_true;
832 static struct stree *saved_implied_false;
833 static struct stree *extra_saved_implied_true;
834 static struct stree *extra_saved_implied_false;

836 static void separate_extra_states(struct stree **implied_true,
837                                   struct stree **implied_false)
838 {
839         struct sm_state *sm;

841         /* We process extra states later to preserve the implications. */
842         FOR_EACH_SM(*implied_true, sm) {
843                 if (sm->owner == SMATCH_EXTRA)
844                         overwrite_sm_state_stree(&extra_saved_implied_true, sm);
845         } END_FOR_EACH_SM(sm);
846         FOR_EACH_SM(extra_saved_implied_true, sm) {
847                 delete_state_stree(implied_true, sm->owner, sm->name, sm->sym);
848         } END_FOR_EACH_SM(sm);
```

```
        FOR_EACH_SM(*implied_false, sm) {
                if (sm->owner == SMATCH_EXTRA)
                        overwrite_sm_state_stree(&extra_saved_implied_false, sm)
        } END_FOR_EACH_SM(sm);
        FOR_EACH_SM(extra_saved_implied_false, sm) {
                delete_state_stree(implied_false, sm->owner, sm->name, sm->sym);
        } END_FOR_EACH_SM(sm);
}

static void get_tf_states(struct expression *expr,
                          struct stree **implied_true,
                          struct stree **implied_false)
{
        if (handled_by_comparison_hook(expr, implied_true, implied_false))
                return;
                goto found;

        if (handled_by_extra_states(expr, implied_true, implied_false)) {
                separate_extra_states(implied_true, implied_false);
                return;
                goto found;
        }

        if (handled_by_stored_conditions(expr, implied_true, implied_false))
                goto found;

                return;
found:
        found_implications = 1;
}

static void save_implications_hook(struct expression *expr)
{
        if (going_too_slow())
        if (taking_too_long())
                return;
        get_tf_states(expr, &saved_implied_true, &saved_implied_false);
}
______unchanged_portion_omitted_

void param_limit_implications(struct expression *expr, int param, char *key, cha
{
        struct expression *arg;
        struct symbol *compare_type;
        char *name;
        struct symbol *sym;
        struct sm_state *sm;
        struct sm_state *tmp;
        struct stree *implied_true = NULL;
        struct stree *implied_false = NULL;
        struct range_list *orig, *limit;

        if (time_parsing_function() > 40)
                return;

        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;

        arg = strip_parens(arg);
```



```
        while (arg->type == EXPR_ASSIGNMENT && arg->op == '=')
                arg = strip_parens(arg->left);

        name = get_variable_from_key(arg, key, &sym);
        if (!name || !sym)
                goto free;

        sm = get_sm_state(SMATCH_EXTRA, name, sym);
        if (!sm || !sm->merged)
                goto free;

        if (strcmp(key, "$") == 0)
                compare_type = get_arg_type(expr->fn, param);
        else
                compare_type = get_member_type_from_key(arg, key);

        orig = estate_rl(sm->state);
        orig = cast_rl(compare_type, orig);

        call_results_to_rl(expr, compare_type, value, &limit);

        separate_and_filter(sm, SPECIAL_EQUAL, limit, __get_cur_stree(), &implie

        FOR_EACH_SM(implied_true, tmp) {
                __set_sm_fake_stree(tmp);
        } END_FOR_EACH_SM(tmp);

        free_stree(&implied_true);
        free_stree(&implied_false);
free:
        free_string(name);
}
______unchanged_portion_omitted_

int assume(struct expression *expr)
{
        int orig_final_pass = final_pass;

        in_fake_env++;
        final_pass = 0;
        __push_fake_cur_stree();
        found_implications = 0;
        __split_whole_condition(expr);
        final_pass = orig_final_pass;
        in_fake_env--;

        return 1;
}
______unchanged_portion_omitted_
```

```
**********************************************************
    6258 Mon Aug  5 08:38:35 2019
new/usr/src/tools/smatch/src/smatch_integer_overflow.c
11506 smatch resync
**********************************************************
```

```
/*
 * 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 link_id;

static struct smatch_state *safe_state(struct expression *expr)
{
        struct smatch_state *state;

        state = __alloc_smatch_state(0);
        expr = strip_expr(expr);
        state->name = alloc_sname("safe");
        state->data = expr;
        return state;
}

static char *save_links(struct expression *expr, struct symbol **sym, struct var
{
        struct var_sym *vs;
        char *name;

        name = expr_to_chunk_sym_vsl(expr, sym, vsl);
        if (!name || !*vsl) {
                free_string(name);
                return NULL;
        }

        FOR_EACH_PTR(*vsl, vs) {
                store_link(link_id, vs->var, vs->sym, name, *sym);
        } END_FOR_EACH_PTR(vs);

        return name;
}

static void match_divide(struct expression *expr)
{
        struct expression *left, *right, *binop;
        struct symbol *type;
        char *name;
        struct symbol *sym;
        struct var_sym_list *vsl;
        sval_t max;
```



```
        if (expr->type != EXPR_COMPARE)
                return;
        if (expr->op != '>' && expr->op != SPECIAL_UNSIGNED_GT &&
            expr->op != SPECIAL_GTE && expr->op != SPECIAL_UNSIGNED_GTE)
                return;

        left = strip_parens(expr->left);
        right = strip_parens(expr->right);

        if (right->type != EXPR_BINOP || right->op != '/')
                return;
        if (!get_value(right->left, &max))
                return;
        if (max.value != INT_MAX && max.value != UINT_MAX &&
            max.value != LLONG_MAX && max.uvalue != ULLONG_MAX)
                return;

        type = get_type(expr);
        if (!type)
                return;
        if (type_bits(type) != 32 && type_bits(type) != 64)
                return;


        binop = binop_expression(left, '*', right->right);

        name = save_links(binop, &sym, &vsl);
        if (!name)
                return;
        set_true_false_states(my_id, name, sym, NULL, safe_state(binop));
        free_string(name);
}

static void match_overflow_to_less_than(struct expression *expr)
{
        struct expression *left, *right;
        struct symbol *type;
        char *name;
        struct symbol *sym;
        struct var_sym_list *vsl;

        if (expr->type != EXPR_COMPARE)
                return;
        if (expr->op != '<' && expr->op != SPECIAL_UNSIGNED_LT)
                return;

        left = strip_parens(expr->left);
        right = strip_parens(expr->right);

        if (left->op != '+')
                return;

        type = get_type(expr);
        if (!type)
                return;
        if (type_bits(type) != 32 && type_bits(type) != 64)
                return;

        if (!expr_equiv(left->left, right) && !expr_equiv(left->right, right))
                return;

        name = save_links(left, &sym, &vsl);
        if (!name)
                return;
        set_true_false_states(my_id, name, sym, NULL, safe_state(left));
```

```
128         free_string(name);
129 }

131 static void match_condition(struct expression *expr)
132 {
133         match_overflow_to_less_than(expr);
134         match_divide(expr);
135 }

137 int can_integer_overflow(struct symbol *type, struct expression *expr)
138 {
139         int op;
140         sval_t lmax, rmax, res;

142         if (!type)
143                 type = &int_ctype;

145         expr = strip_expr(expr);

147         if (expr->type == EXPR_ASSIGNMENT) {
148                 switch(expr->op) {
149                 case SPECIAL_MUL_ASSIGN:
150                         op = '*';
151                         break;
152                 case SPECIAL_ADD_ASSIGN:
153                         op = '+';
154                         break;
155                 case SPECIAL_SHL_ASSIGN:
156                         op = SPECIAL_LEFTSHIFT;
157                         break;
158                 default:
159                         return 0;
160                 }
161         } else if (expr->type == EXPR_BINOP) {
162                 if (expr->op != '*' && expr->op != '+' && expr->op != SPECIAL_LE
163                         return 0;
164                 op = expr->op;
165         } else {
166                 return 0;
167         }

169         get_absolute_max(expr->left, &lmax);
170         get_absolute_max(expr->right, &rmax);

172         if (sval_binop_overflows(lmax, op, rmax))
173                 return 1;

175         res = sval_binop(lmax, op, rmax);
176         if (sval_cmp(res, sval_type_max(type)) > 0)
177                 return 1;
178         return 0;
179 }

181 int can_integer_overflow_expr(struct expression *expr)
182 {
183         struct symbol *type;
184         struct smatch_state *state;
185         char *name;
186         struct symbol *sym;
187         int ret;

189         type = get_type(expr);
190         if (!type)
191                 return 0;

193         if (!can_integer_overflow(type, expr))
```



```
194                 return 0;

196         name = expr_to_known_chunk_sym(expr, &sym);
197         if (!name || !sym)
198                 goto free;

200         state = get_state(my_id, name, sym);
201         if (state && state->data)
202                 ret = 0;
203 free:
204         free_string(name);
205         return ret;
206 }

208 static int get_arg_nr(struct expression *call, struct expression *expr)
209 {
210         struct expression *arg;
211         int i;

213         i = -1;
214         FOR_EACH_PTR(call->args, arg) {
215                 i++;
216                 if (expr_equiv(arg, expr))
217                         return i;
218         } END_FOR_EACH_PTR(arg);

220         return -1;
221 }

223 static void check_links(struct expression *call, struct expression *arg, int nr,
224 {
225         struct var_sym_list *vsl = _vsl;
226         struct var_sym *vs;
227         struct smatch_state *state;
228         struct expression *expr;
229         int left = -1;
230         int right = -1;

232         FOR_EACH_PTR(vsl, vs) {
233                 state = get_state(my_id, vs->var, vs->sym);
234                 if (!state || !state->data)
235                         continue;

237                 expr = state->data;

239                 if (expr_equiv(arg, expr->left)) {
240                         left = nr;
241                         right = get_arg_nr(call, expr->right);
242                 } else if (expr_equiv(arg, expr->right)) {
243                         left = get_arg_nr(call, expr->left);
244                         right = nr;
245                 }

247                 if (left == -1 || right == -1)
248                         continue;

250                 left = -1;
251                 right = -1;
252         } END_FOR_EACH_PTR(vs);
253 }

255 static void match_call_info(struct expression *call)
256 {
257         struct expression *arg;
258         struct sm_state *link;
259         struct stree *done = NULL;
```

```
260         int i;
262         i = -1;
263         FOR_EACH_PTR(call->args, arg) {
264                 i++;
266                 link = get_sm_state_expr(link_id, arg);
267                 if (!link)
268                         continue;
270                 if (get_state_stree(done, my_id, link->state->name, NULL))
271                         continue;
272 //              set_state_stree(&done, my_id, link->state->name, NULL, &undefine
274                 check_links(call, arg, i, link, link->state->data);
275         } END_FOR_EACH_PTR(arg);
277         free_stree(&done);
278 }
280 void register_integer_overflow(int id)
281 {
282         my_id = id;
283         set_dynamic_states(my_id);
284         add_hook(&match_condition, CONDITION_HOOK);
285         add_hook(&match_call_info, FUNCTION_CALL_HOOK);
286 }
288 void register_integer_overflow_links(int id)
289 {
290         link_id = id;
291         set_up_link_functions(my_id, link_id);
292 }
```

```
**********************************************************
   36844 Mon Aug  5 08:38:35 2019
new/usr/src/tools/smatch/src/smatch_kernel_user_data.c
11506 smatch resync
**********************************************************
```

```
  1 /*
  2  * Copyright (C) 2011 Dan Carpenter.
  3  *
  4  * This program is free software; you can redistribute it and/or
  5  * modify it under the terms of the GNU General Public License
  6  * as published by the Free Software Foundation; either version 2
  7  * of the License, or (at your option) any later version.
  8  *
  9  * This program is distributed in the hope that it will be useful,
 10  * but WITHOUT ANY WARRANTY; without even the implied warranty of
 11  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 12  * GNU General Public License for more details.
 13  *
 14  * You should have received a copy of the GNU General Public License
 15  * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 16  */

 18 /*
 19  * There are a couple checks that try to see if a variable
 20  * comes from the user.  It would be better to unify them
 21  * into one place.  Also it we should follow the data down
 22  * the call paths.  Hence this file.
 23  */

 25 #include "smatch.h"
 26 #include "smatch_slist.h"
 27 #include "smatch_extra.h"

 29 static int my_id;
 30 static int my_call_id;

 32 STATE(called);
 33 static bool func_gets_user_data;

 35 static const char *kstr_funcs[] = {
 35 static const char * kstr_funcs[] = {
 36         "kstrtoull", "kstrtoll", "kstrtoul", "kstrtol", "kstrtouint",
 37         "kstrtoint", "kstrtou64", "kstrtos64", "kstrtou32", "kstrtos32",
 38         "kstrtou16", "kstrtos16", "kstrtou8", "kstrtos8", "kstrtoull_from_user"
 39         "kstrtoll_from_user", "kstrtoul_from_user", "kstrtol_from_user",
 40         "kstrtouint_from_user", "kstrtoint_from_user", "kstrtou16_from_user",
 41         "kstrtos16_from_user", "kstrtou8_from_user", "kstrtos8_from_user",
 42         "kstrtou64_from_user", "kstrtos64_from_user", "kstrtou32_from_user",
 43         "kstrtos32_from_user",
 44 };

 46 static const char *returns_user_data[] = {
 47         "simple_strtol", "simple_strtoll", "simple_strtoul", "simple_strtoull",
 48         "kvm_register_read",
 48         "kvm_register_read", "nlmsg_data", "nla_data", "memdup_user",
 49         "kmap_atomic", "skb_network_header",
 49 };

 51 static const char *returns_pointer_to_user_data[] = {
 52         "nlmsg_data", "nla_data", "memdup_user", "kmap_atomic", "skb_network_hea
 53 };

 55 static void set_points_to_user_data(struct expression *expr);

 57 static struct stree *start_states;
 58 static struct stree_stack *saved_stack;
```



```
 59 static void save_start_states(struct statement *stmt)
 60 {
 61         start_states = clone_stree(__get_cur_stree());
 62 }
______unchanged_portion_omitted_

 86 static void pre_merge_hook(struct sm_state *sm)
 87 {
 88         struct smatch_state *user;
 89         struct smatch_state *extra;
 90         struct smatch_state *state;
 91         struct range_list *rl;
 92         sval_t dummy;
 93         sval_t sval_100;

 95         sval_100.value = 100;
 96         sval_100.type = &int_ctype;

 98         user = __get_state(my_id, sm->name, sm->sym);
 99         if (!user || !estate_rl(user))
 94         user = get_state(my_id, sm->name, sm->sym);
 95         if (!user)
100                 return;
101         extra = __get_state(SMATCH_EXTRA, sm->name, sm->sym);
102         if (!extra)
 97         if (!__in_function_def && !estate_rl(sm->state)) {
 98                 /*
 99                  * If the one side is capped and the other side is empty then
100                  * let's just mark it as not-user data because the information
101                  * isn't going to be useful.  How this looks is:
102                  *
103                  * if (user_var > trusted)
104                  *      user_var = trusted;  <-- empty state
105                  * else
106                  *      <-- capped
107                  *
108                  * The problem is that sometimes things are capped to a literal
109                  * and we'd like to keep the state in that case...  Ugh.  I've
110                  * added a check which assumes that everything less than 100 is
111                  * probably capped against a literal.
112                  *
113                  */
114                 if (is_capped_var_sym(sm->name, sm->sym) &&
115                     sval_cmp(estate_max(user), sval_100) > 0)
116                         set_state(my_id, sm->name, sm->sym, alloc_estate_empty()
103                 return;
118         }
119         extra = get_state(SMATCH_EXTRA, sm->name, sm->sym);
120         if (!extra || !estate_rl(extra))
121                 return;
104         rl = rl_intersection(estate_rl(user), estate_rl(extra));
105         if (rl_to_sval(rl, &dummy))
106                 rl = NULL;
107         state = alloc_estate_rl(clone_rl(rl));
108         if (estate_capped(user) || is_capped_var_sym(sm->name, sm->sym))
109                 estate_set_capped(state);
110         set_state(my_id, sm->name, sm->sym, state);
125         set_state(my_id, sm->name, sm->sym, alloc_estate_rl(clone_rl(rl)));
111 }

113 static void extra_nomod_hook(const char *name, struct symbol *sym, struct expres
114 {
115         struct smatch_state *user, *new;
130         struct smatch_state *user;
116         struct range_list *rl;
```

```
118         user = __get_state(my_id, name, sym);
133         user = get_state(my_id, name, sym);
119         if (!user)
120                 return;
121         rl = rl_intersection(estate_rl(user), estate_rl(state));
122         if (rl_equiv(rl, estate_rl(user)))
123                 return;
124         new = alloc_estate_rl(rl);
125         if (estate_capped(user))
126                 estate_set_capped(new);
127         set_state(my_id, name, sym, new);
139         set_state(my_id, name, sym, alloc_estate_rl(rl));
128 }

130 static bool binop_capped(struct expression *expr)
131 {
132         struct range_list *left_rl;
133         int comparison;

135         if (expr->op == '-' && get_user_rl(expr->left, &left_rl)) {
136                 if (user_rl_capped(expr->left))
137                         return true;
138                 comparison = get_comparison(expr->left, expr->right);
139                 if (comparison && show_special(comparison)[0] == '>')
140                         return true;
141                 return false;
142         }

144         if (expr->op == '&' || expr->op == '%') {
145                 if (is_capped(expr->left) || is_capped(expr->right))
146                         return true;
147                 if (user_rl_capped(expr->left) || user_rl_capped(expr->right))
148                         return true;
149                 return false;
150         }

152         if (user_rl_capped(expr->left) &&
153             user_rl_capped(expr->right))
154                 return true;
155         return false;
156 }

158 bool user_rl_capped(struct expression *expr)
159 {
160         struct smatch_state *state;
161         struct range_list *rl;
162         sval_t sval;

164         expr = strip_expr(expr);
165         if (!expr)
166                 return false;
167         if (get_value(expr, &sval))
168                 return true;
169         if (expr->type == EXPR_BINOP)
170                 return binop_capped(expr);
171         if ((expr->type == EXPR_PREOP || expr->type == EXPR_POSTOP) &&
172             (expr->op == SPECIAL_INCREMENT || expr->op == SPECIAL_DECREMENT))
173                 return user_rl_capped(expr->unop);
174         state = get_state_expr(my_id, expr);
175         if (state)
176                 return estate_capped(state);

178         if (get_user_rl(expr, &rl))
179                 return false;  /* uncapped user data */

181         return true;  /* not actually user data */
```



```
182 }

184 static void tag_inner_struct_members(struct expression *expr, struct symbol *mem
185 {
186         struct expression *edge_member;
187         struct symbol *base = get_real_base_type(member);
188         struct symbol *tmp;

190         if (member->ident)
191                 expr = member_expression(expr, '.', member->ident);

193         FOR_EACH_PTR(base->symbol_list, tmp) {
194                 struct symbol *type;

196                 type = get_real_base_type(tmp);
197                 if (!type)
198                         continue;

200                 if (type->type == SYM_UNION || type->type == SYM_STRUCT) {
201                         tag_inner_struct_members(expr, tmp);
202                         continue;
203                 }

205                 if (!tmp->ident)
206                         continue;

208                 edge_member = member_expression(expr, '.', tmp->ident);
209                 set_state_expr(my_id, edge_member, alloc_estate_whole(type));
210         } END_FOR_EACH_PTR(tmp);
211 }
______unchanged_portion_omitted_

392 static bool is_points_to_user_data_fn(struct expression *expr)
393 {
394         int i;

396         expr = strip_expr(expr);
397         if (expr->type != EXPR_CALL || expr->fn->type != EXPR_SYMBOL ||
398             !expr->fn->symbol)
399                 return false;
400         expr = expr->fn;
401         for (i = 0; i < ARRAY_SIZE(returns_pointer_to_user_data); i++) {
402                 if (sym_name_is(returns_pointer_to_user_data[i], expr))
403                         return true;
404         }
405         return false;
406 }

408 static int get_rl_from_function(struct expression *expr, struct range_list **rl)
409 {
410         int i;

412         if (expr->type != EXPR_CALL || expr->fn->type != EXPR_SYMBOL ||
413             !expr->fn->symbol_name || !expr->fn->symbol_name->name)
414                 return 0;

416         for (i = 0; i < ARRAY_SIZE(returns_user_data); i++) {
417                 if (strcmp(expr->fn->symbol_name->name, returns_user_data[i]) ==
418                         *rl = alloc_whole_rl(get_type(expr));
419                         return 1;
420                 }
421         }
422         return 0;
423 }

425 int points_to_user_data(struct expression *expr)
```

```
426 {
427         struct smatch_state *state;
428         struct range_list *rl;
429         char buf[256];
430         struct symbol *sym;
431         char *name;
432         int ret = 0;

434         expr = strip_expr(expr);
435         if (!expr)
436                 return 0;
437         if (is_skb_data(expr))
438                 return 1;
439         if (is_points_to_user_data_fn(expr))
440                 return 1;
441         if (get_rl_from_function(expr, &rl))
442                 return 1;

444         if (expr->type == EXPR_BINOP && expr->op == '+') {
445                 if (points_to_user_data(expr->left))
446                         return 1;
447                 if (points_to_user_data(expr->right))
448                         return 1;
449                 return 0;
450         }

452         name = expr_to_var_sym(expr, &sym);
453         if (!name || !sym)
454                 goto free;
455         snprintf(buf, sizeof(buf), "*%s", name);
456         state = __get_state(my_id, buf, sym);
396         state = get_state(my_id, buf, sym);
457         if (state && estate_rl(state))
458                 ret = 1;
459 free:
460         free_string(name);
461         return ret;
462 }

464 static void set_points_to_user_data(struct expression *expr)
465 {
466         char *name;
467         struct symbol *sym;
468         char buf[256];
469         struct symbol *type;

471         name = expr_to_var_sym(expr, &sym);
472         if (!name || !sym)
473                 goto free;
474         snprintf(buf, sizeof(buf), "*%s", name);
475         type = get_type(expr);
476         if (type && type->type == SYM_PTR)
477                 type = get_real_base_type(type);
478         if (!type || type->type != SYM_BASETYPE)
479                 type = &llong_ctype;
480         set_state(my_id, buf, sym, alloc_estate_whole(type));
414         set_state(my_id, buf, sym, alloc_estate_whole(&llong_ctype));
481 free:
482         free_string(name);
483 }
```

*unchanged_portion_omitted_*

```
558 static bool handle_op_assign(struct expression *expr)
559 {
560         struct expression *binop_expr;
561         struct smatch_state *state;
```



```
562         struct range_list *rl;

564         switch (expr->op) {
565         case SPECIAL_ADD_ASSIGN:
566         case SPECIAL_SUB_ASSIGN:
567         case SPECIAL_AND_ASSIGN:
568         case SPECIAL_MOD_ASSIGN:
569         case SPECIAL_SHL_ASSIGN:
570         case SPECIAL_SHR_ASSIGN:
571         case SPECIAL_OR_ASSIGN:
572         case SPECIAL_XOR_ASSIGN:
573         case SPECIAL_MUL_ASSIGN:
574         case SPECIAL_DIV_ASSIGN:
575                 binop_expr = binop_expression(expr->left,
576                                               op_remove_assign(expr->op),
577                                               expr->right);
578                 if (!get_user_rl(binop_expr, &rl))
579                         return true;

581                 rl = cast_rl(get_type(expr->left), rl);
582                 state = alloc_estate_rl(rl);
583                 if (user_rl_capped(binop_expr))
584                         estate_set_capped(state);
585                 set_state_expr(my_id, expr->left, state);
586                 return true;
587         }
588         return false;
589 }

591 static void match_assign(struct expression *expr)
592 {
593         struct range_list *rl;
594         static struct expression *handled;
595         struct smatch_state *state;
596         struct expression *faked;

598         faked = get_faked_expression();
599         if (faked && faked == handled)
600                 return;
601         if (is_fake_call(expr->right))
602                 goto clear_old_state;
603         if (handle_get_user(expr))
604                 return;
605         if (points_to_user_data(expr->right)) {
606                 handled = expr;
500         if (points_to_user_data(expr->right))
607                 set_points_to_user_data(expr->left);
608         }
609         if (handle_struct_assignment(expr))
610                 return;

612         if (handle_op_assign(expr))
613                 return;
614         if (expr->op != '=')
615                 goto clear_old_state;

617         /* Handled by DB code */
618         if (expr->right->type == EXPR_CALL || __in_fake_parameter_assign)
619                 return;

621         if (!get_user_rl(expr->right, &rl))
622                 goto clear_old_state;

624         rl = cast_rl(get_type(expr->left), rl);
625         state = alloc_estate_rl(rl);
626         if (user_rl_capped(expr->right))
```

```
627                 estate_set_capped(state);
628         set_state_expr(my_id, expr->left, state);
509         set_state_expr(my_id, expr->left, alloc_estate_rl(rl));

630         return;

632 clear_old_state:
633         if (get_state_expr(my_id, expr->left))
634                 set_state_expr(my_id, expr->left, alloc_estate_empty());
635 }
_____unchanged_portion_omitted_

660 static struct range_list *strip_negatives(struct range_list *rl)
541 static void handle_unsigned_lt_gt(struct expression *expr)
661 {
662         sval_t min = rl_min(rl);
663         sval_t minus_one;
664         sval_t over;
665         sval_t max = sval_type_max(rl_type(rl));

667         minus_one.type = rl_type(rl);
668         minus_one.value = INT_MAX + 1ULL;
669         over.type = rl_type(rl);
670         over.value = -1;

672         if (!rl)
673                 return NULL;

675         if (type_unsigned(rl_type(rl)) && type_bits(rl_type(rl)) > 31)
676                 return remove_range(rl, over, max);

678         return remove_range(rl, min, minus_one);
679 }

681 static void handle_compare(struct expression *expr)
682 {
683         struct expression  *left, *right;
684         struct range_list *left_rl = NULL;
685         struct range_list *right_rl = NULL;
686         struct range_list *user_rl;
687         struct smatch_state *capped_state;
688         struct smatch_state *left_true = NULL;
689         struct smatch_state *left_false = NULL;
690         struct smatch_state *right_true = NULL;
691         struct smatch_state *right_false = NULL;
692         struct symbol *type;
693         sval_t sval;
544         struct range_list *left;
545         struct range_list *right;
546         struct range_list *non_negative;
547         sval_t min, minus_one;

695         left = strip_expr(expr->left);
696         right = strip_expr(expr->right);

698         while (left->type == EXPR_ASSIGNMENT)
699                 left = strip_expr(left->left);

701         /*
702          * Conditions are mostly handled by smatch_extra.c, but there are some
703          * times where the exact values are not known so we can't do that.
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.
704          *
705          * Normally, we might consider using smatch_capped.c to supliment smatch
706          * extra but that doesn't work when we merge unknown uncapped kernel
707          * data with unknown capped user data.  The result is uncapped user
708          * data.  We need to keep it separate and say that the user data is
709          * capped.  In the past, I would have marked this as just regular
710          * kernel data (not user data) but we can't do that these days because
711          * we need to track user data for Spectre.
712          *
713          * The other situation which we have to handle is when we do have an
714          * int and we compare against an unknown unsigned kernel variable.  In
715          * that situation we assume that the kernel data is less than INT_MAX.
716          * Otherwise then we get all sorts of array underflow false positives.
717          *
718          */

720         /* Handled in smatch_extra.c */
721         if (get_implied_value(left, &sval) ||
722             get_implied_value(right, &sval))
560         type = get_type(expr);
561         if (!type_unsigned(type))
723                 return;

725         get_user_rl(left, &left_rl);
726         get_user_rl(right, &right_rl);

728         /* nothing to do */
729         if (!left_rl && !right_rl)
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))
730                 return;
731         /* if both sides are user data that's not a good limit */
732         if (left_rl && right_rl)
570         if (get_user_rl(expr->right, &right))
733                 return;

735         if (left_rl)
736                 user_rl = left_rl;
737         else
738                 user_rl = right_rl;
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);

740         type = get_type(expr);
741         if (type_unsigned(type))
742                 user_rl = strip_negatives(user_rl);
743         capped_state = alloc_estate_rl(user_rl);
744         estate_set_capped(capped_state);

746         switch (expr->op) {
747         case '<':
748         case SPECIAL_UNSIGNED_LT:
749         case SPECIAL_LTE:
750         case SPECIAL_UNSIGNED_LTE:
751                 if (left_rl)
752                         left_true = capped_state;
753                 else
```

```
754                         right_false = capped_state;
585                 set_true_false_states_expr(my_id, expr->left,
586                                            alloc_estate_rl(non_negative), NULL);
755                 break;
756         case '>':
757         case SPECIAL_UNSIGNED_GT:
758         case SPECIAL_GTE:
759         case SPECIAL_UNSIGNED_GTE:
760                 if (left_rl)
761                         left_false = capped_state;
762                 else
763                         right_true = capped_state;
592                 set_true_false_states_expr(my_id, expr->left,
593                                            NULL, alloc_estate_rl(non_negative));
764                 break;
765         }

767         set_true_false_states_expr(my_id, left, left_true, left_false);
768         set_true_false_states_expr(my_id, right, right_true, right_false);
769 }

771 static void match_condition(struct expression *expr)
772 {
773         if (expr->type != EXPR_COMPARE)
774                 return;

776         if (expr->op == SPECIAL_EQUAL ||
777             expr->op == SPECIAL_NOTEQUAL) {
778                 handle_eq_noteq(expr);
779                 return;
780         }

782         handle_compare(expr);
609         handle_unsigned_lt_gt(expr);
783 }
```

*unchanged_portion_omitted_*

```
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 }

830 static int has_user_data(struct symbol *sym)
831 {
832         struct sm_state *tmp;

834         FOR_EACH_MY_SM(my_id, __get_cur_stree(), tmp) {
835                 if (tmp->sym == sym)
836                         return 1;
837         } END_FOR_EACH_SM(tmp);
838         return 0;
839 }
```

*unchanged_portion_omitted_*

```
857 static int db_returned_user_rl(struct expression *call, struct range_list **rl)
858 {
859         struct smatch_state *state;
860         char buf[48];
718         struct db_info db_info = {};

720         /* for function pointers assume everything is used */
721         if (call->fn->type != EXPR_SYMBOL)
722                 return 0;
862         if (is_fake_call(call))
863                 return 0;
864         snprintf(buf, sizeof(buf), "return %p", call);
865         state = get_state(my_id, buf, NULL);
866         if (!state || !estate_rl(state))

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))
867                 return 0;
868         *rl = estate_rl(state);
742                 *rl = db_info.rl;
869         return 1;
744         }

746         return 0;
870 }
```

*unchanged_portion_omitted_*

```
877 static int user_data_flag;
878 static int no_user_data_flag;
879 struct range_list *var_user_rl(struct expression *expr)
756 static struct range_list *var_user_rl(struct expression *expr)
880 {
881         struct smatch_state *state;
882         struct range_list *rl;
883         struct range_list *absolute_rl;

885         if (expr->type == EXPR_PREOP && expr->op == '&') {
886                 no_user_data_flag = 1;
```

```
887                 return NULL;
888         }

890         if (expr->type == EXPR_BINOP && expr->op == '%') {
891                 struct range_list *left, *right;

893                 if (!get_user_rl(expr->right, &right))
894                         return NULL;
895                 get_absolute_rl(expr->left, &left);
896                 rl = rl_binop(left, '%', right);
897                 goto found;
898         }

900         if (expr->type == EXPR_BINOP && expr->op == '/') {
772         if (!option_spammy && expr->type == EXPR_BINOP && expr->op == '/') {
901                 struct range_list *left = NULL;
902                 struct range_list *right = NULL;
903                 struct range_list *abs_right;

905                 /*
906                  * The specific bug I'm dealing with is:
907                  *
908                  * foo = capped_user / unknown;
909                  *
910                  * Instead of just saying foo is now entirely user_rl we should
911                  * probably say instead that it is not at all user data.
912                  *
913                  */

915                 get_user_rl(expr->left, &left);
916                 get_user_rl(expr->right, &right);
917                 get_absolute_rl(expr->right, &abs_right);

919                 if (left && !right) {
920                         rl = rl_binop(left, '/', abs_right);
921                         if (sval_cmp(rl_max(left), rl_max(rl)) < 0)
922                                 no_user_data_flag = 1;
923                 }

925                 return NULL;
926         }

928         if (get_rl_from_function(expr, &rl))
929                 goto found;

931         if (get_user_macro_rl(expr, &rl))
932                 goto found;

934         if (comes_from_skb_data(expr)) {
935                 rl = alloc_whole_rl(get_type(expr));
936                 goto found;
937         }

939         state = get_state_expr(my_id, expr);
940         if (state && estate_rl(state)) {
941                 rl = estate_rl(state);
942                 goto found;
943         }

945         if (expr->type == EXPR_CALL && db_returned_user_rl(expr, &rl))
946                 goto found;

948         if (is_array(expr)) {
949                 struct expression *array = get_array_base(expr);

951                 if (!get_state_expr(my_id, array)) {
```



```
952                         no_user_data_flag = 1;
953                         return NULL;
954                 }
955         }

957         if (expr->type == EXPR_PREOP && expr->op == '*' &&
958             is_user_rl(expr->unop)) {
959                 rl = var_to_absolute_rl(expr);
960                 goto found;
961         }

963         return NULL;
964 found:
965         user_data_flag = 1;
966         absolute_rl = var_to_absolute_rl(expr);
967         return clone_rl(rl_intersection(rl, absolute_rl));
968 }

970 static bool is_ptr_subtract(struct expression *expr)
971 {
972         expr = strip_expr(expr);
973         if (!expr)
974                 return false;
975         if (expr->type == EXPR_BINOP && expr->op == '-' &&
976             type_is_ptr(get_type(expr->left))) {
977                 return true;
978         }
979         return false;
980 }

982 int get_user_rl(struct expression *expr, struct range_list **rl)
983 {
984         if (is_ptr_subtract(expr))
985                 return 0;

987         user_data_flag = 0;
988         no_user_data_flag = 0;
989         custom_get_absolute_rl(expr, &var_user_rl, rl);
990         if (!user_data_flag || no_user_data_flag)
991                 *rl = NULL;

993         return !!*rl;
994 }

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 }

996 int is_user_rl(struct expression *expr)
997 {
998         struct range_list *tmp;

1000        return !!get_user_rl(expr, &tmp);
 868        return get_user_rl_spammy(expr, &tmp);
1001 }
______unchanged_portion_omitted_

1015 static char *get_user_rl_str(struct expression *expr, struct symbol *type)
1016 {
```

```
1017         struct range_list *rl;
1018         static char buf[64];

1020         if (!get_user_rl(expr, &rl))
1021                 return NULL;
1022         rl = cast_rl(type, rl);
1023         snprintf(buf, sizeof(buf), "%s%s",
1024                  show_rl(rl), user_rl_capped(expr) ? "[c]" : "");
1025         return buf;
1026 }

1028 static void match_call_info(struct expression *expr)
1029 {
 885         struct range_list *rl;
1030         struct expression *arg;
1031         struct symbol *type;
1032         char *str;
1033         int i;
 888         int i = 0;

1035         i = -1;
1036         FOR_EACH_PTR(expr->args, arg) {
1037                 i++;
1038                 type = get_arg_type(expr->fn, i);
1039                 str = get_user_rl_str(arg, type);
1040                 if (!str)

 895                 if (!get_user_rl(arg, &rl))
1041                         continue;

1043                 sql_insert_caller_info(expr, USER_DATA, i, "$", str);
 898                 rl = cast_rl(type, rl);
 899                 sql_insert_caller_info(expr, USER_DATA3, i, "$", show_rl(rl));
1044         } END_FOR_EACH_PTR(arg);
1045 }
______unchanged_portion_omitted_

1062 static void struct_member_callback(struct expression *call, int param, char *pri
1063 {
1064         struct smatch_state *state;
1065         struct range_list *rl;
1066         struct symbol *type;
1067         char buf[64];

1069         /*
1070          * Smatch uses a hack where if we get an unsigned long we say it's
1071          * both user data and it points to user data.  But if we pass it to a
1072          * function which takes an int, then it's just user data.  There's not
1073          * enough bytes for it to be a pointer.
1074          *
1075          */
1076         type = get_arg_type(call->fn, param);
1077         if (type && type_bits(type) < type_bits(&ptr_ctype))
1078                 return;

1080         if (strcmp(sm->state->name, "") == 0)
1081                 return;

1083         if (strcmp(printed_name, "*$") == 0 &&
1084             is_struct_ptr(sm->sym))
1085                 return;

1087         state = __get_state(SMATCH_EXTRA, sm->name, sm->sym);
 942         state = get_state(SMATCH_EXTRA, sm->name, sm->sym);
1088         if (!state || !estate_rl(state))
1089                 rl = estate_rl(sm->state);
```



```
1090         else
1091                 rl = rl_intersection(estate_rl(sm->state), estate_rl(state));

1093         if (!rl)
1094                 return;

1096         snprintf(buf, sizeof(buf), "%s%s", show_rl(rl),
1097                  estate_capped(sm->state) ? "[c]" : "");
1098         sql_insert_caller_info(call, USER_DATA, param, printed_name, buf);
 948         sql_insert_caller_info(call, USER_DATA3, param, printed_name, show_rl(rl
1099 }

1101 static void db_param_set(struct expression *expr, int param, char *key, char *va
1102 {
1103         struct expression *arg;
1104         char *name;
1105         struct symbol *sym;
1106         struct smatch_state *state;

1108         while (expr->type == EXPR_ASSIGNMENT)
1109                 expr = strip_expr(expr->right);
1110         if (expr->type != EXPR_CALL)
1111                 return;

1113         arg = get_argument_from_call_expr(expr->args, param);
1114         if (!arg)
1115                 return;
1116         name = get_variable_from_key(arg, key, &sym);
1117         if (!name || !sym)
1118                 goto free;

1120         state = get_state(my_id, name, sym);
1121         if (!state)
1122                 goto free;

1124         set_state(my_id, name, sym, alloc_estate_empty());
1125 free:
1126         free_string(name);
1127 }

1129 static bool param_data_capped(const char *value)
1130 {
1131         if (strstr(value, ",c") || strstr(value, "[c]"))
1132                 return true;
1133         return false;
1134 }

1136 static void set_param_user_data(const char *name, struct symbol *sym, char *key,
1137 {
1138         struct range_list *rl = NULL;
1139         struct smatch_state *state;
1140         struct expression *expr;
1141         struct symbol *type;
1142         char fullname[256];
1143         char *key_orig = key;
1144         bool add_star = false;

1146         if (strcmp(key, "**$") == 0) {
1147                 snprintf(fullname, sizeof(fullname), "**%s", name);
1148         } else {
1149                 if (key[0] == '*') {
1150                         add_star = true;
1151                         key++;
1152                 }
 958         if (strcmp(key, "*$") == 0)
 959                 snprintf(fullname, sizeof(fullname), "*%s", name);
```

```
 960         else if (strncmp(key, "$", 1) == 0)
 961                 snprintf(fullname, 256, "%s%s", name, key + 1);
 962         else
 963                 return;

1154                 snprintf(fullname, 256, "%s%s%s", add_star ? "*" : "", name, key
1155         }
 965         type = get_member_type_from_key(symbol_expression(sym), key);

1157         expr = symbol_expression(sym);
1158         type = get_member_type_from_key(expr, key_orig);
 967         /* if the caller passes a void pointer with user data */
 968         if (strcmp(key, "*$") == 0 && type && type != &void_ctype) {
 969                 struct expression *expr = symbol_expression(sym);

1160         /*
1161          * Say this function takes a struct ponter but the caller passes
1162          * this_function(skb->data).  We have two options, we could pass *$
1163          * as user data or we could pass foo->bar, foo->baz as user data.
1164          * The second option is easier to implement so we do that.
1165          *
1166          */
1167         if (strcmp(key_orig, "*$") == 0) {
1168                 struct symbol *tmp = type;

1170                 while (tmp && tmp->type == SYM_PTR)
1171                         tmp = get_real_base_type(tmp);

1173                 if (tmp && (tmp->type == SYM_STRUCT || tmp->type == SYM_UNION))
1174                         tag_as_user_data(symbol_expression(sym));
 971                 tag_as_user_data(expr);
 972                 set_points_to_user_data(expr);
1175                         return;
1176                 }
1177         }

1179         str_to_rl(type, value, &rl);
1180         state = alloc_estate_rl(rl);
1181         if (param_data_capped(value) || is_capped(expr))
1182                 estate_set_capped(state);
1183         set_state(my_id, fullname, sym, state);
1184 }
______unchanged_portion_omitted_

1221 static void store_user_data_return(struct expression *expr, char *key, char *val
1222 {
1223         struct range_list *rl;
1224         struct symbol *type;
1225         char buf[48];

1227         if (strcmp(key, "$") != 0)
1228                 return;

1230         type = get_type(expr);
1231         snprintf(buf, sizeof(buf), "return %p", expr);
1232         call_results_to_rl(expr, type, value, &rl);

1234         set_state(my_id, buf, NULL, alloc_estate_rl(rl));
1235 }

1237 static void set_to_user_data(struct expression *expr, char *key, char *value)
1238 {
1239         struct smatch_state *state;
1240         char *name;
1241         struct symbol *sym;
1242         struct symbol *type;
```



```
1243         struct range_list *rl = NULL;

1245         type = get_member_type_from_key(expr, key);
1246         name = get_variable_from_key(expr, key, &sym);
1247         if (!name || !sym)
1248                 goto free;

1250         call_results_to_rl(expr, type, value, &rl);

1252         state = alloc_estate_rl(rl);
1253         if (param_data_capped(value))
1254                 estate_set_capped(state);
1255         set_state(my_id, name, sym, state);
1029         set_state(my_id, name, sym, alloc_estate_rl(rl));
1256 free:
1257         free_string(name);

1258 }

1260 static void returns_param_user_data(struct expression *expr, int param, char *ke
1261 {
1262         struct expression *arg;
1263         struct expression *call;

1265         call = expr;
1266         while (call->type == EXPR_ASSIGNMENT)
1267                 call = strip_expr(call->right);
1268         if (call->type != EXPR_CALL)
1269                 return;

1271         if (!we_pass_user_data(call))
1272                 return;

1274         if (param == -1) {
1275                 if (expr->type != EXPR_ASSIGNMENT) {
1276                         store_user_data_return(expr, key, value);
1050                 if (expr->type != EXPR_ASSIGNMENT)
1277                         return;
1278                 }
1279                 set_to_user_data(expr->left, key, value);
1280                 return;
1281         }

1283         arg = get_argument_from_call_expr(call->args, param);
1284         if (!arg)
1285                 return;
1286         set_to_user_data(arg, key, value);
1287 }

1289 static void returns_param_user_data_set(struct expression *expr, int param, char
1290 {
1291         struct expression *arg;

1293         func_gets_user_data = true;

1295         if (param == -1) {
1296                 if (expr->type != EXPR_ASSIGNMENT) {
1297                         store_user_data_return(expr, key, value);
1069                 if (expr->type != EXPR_ASSIGNMENT)
1298                         return;
1299                 }
1300                 if (strcmp(key, "*$") == 0) {
1301                         set_points_to_user_data(expr->left);
1302                         tag_as_user_data(expr->left);
1303                 } else {
1304                         set_to_user_data(expr->left, key, value);
```

```
1305                }
1306                return;
1307        }

1309        while (expr->type == EXPR_ASSIGNMENT)
1310                expr = strip_expr(expr->right);
1311        if (expr->type != EXPR_CALL)
1312                return;

1314        arg = get_argument_from_call_expr(expr->args, param);
1315        if (!arg)
1316                return;
1317        set_to_user_data(arg, key, value);
1318 }

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 }

1320 static void param_set_to_user_data(int return_id, char *return_ranges, struct ex
1321 {
1322         struct sm_state *sm;
1323         struct smatch_state *start_state;
1324         struct range_list *rl;
1325         int param;
1326         char *return_str;
1327         const char *param_name;
1328         struct symbol *ret_sym;
1329         bool return_found = false;
1330         bool pointed_at_found = false;
1331         char buf[64];

1333         expr = strip_expr(expr);
1334         return_str = expr_to_str(expr);
1335         ret_sym = expr_to_sym(expr);

1337         FOR_EACH_MY_SM(my_id, __get_cur_stree(), sm) {
1119                 if (has_empty_state(sm))
1120                         continue;

1338                 param = get_param_num_from_sym(sm->sym);
1339                 if (param < 0)
1340                         continue;

1342                 if (!param_was_set_var_sym(sm->name, sm->sym))
1343                         continue;

1345                 /* The logic here was that if we were passed in a user data then
1346                  * we don't record that.  It's like the difference between
1347                  * param_filter and param_set.  When I think about it, I'm not
1348                  * sure it actually works.  It's probably harmless because we
1349                  * checked earlier that we're not returning a parameter...
1350                  * Let's mark this as a TODO.
1351                  */
1352                 start_state = get_state_stree(start_states, my_id, sm->name, sm-
1353                 if (start_state && rl_equiv(estate_rl(sm->state), estate_rl(star
1354                         continue;
```



```
1356                param_name = get_param_name(sm);
1357                if (!param_name)
1358                        continue;
1359                if (strcmp(param_name, "$") == 0)  /* The -1 param is handled af
1360                        continue;

1362                snprintf(buf, sizeof(buf), "%s%s",
1363                         show_rl(estate_rl(sm->state)),
1364                         estate_capped(sm->state) ? "[c]" : "");
1365                sql_insert_return_states(return_id, return_ranges,
1366                                         func_gets_user_data ? USER_DATA_SET : U
1367                                         param, param_name, buf);
1144                                         func_gets_user_data ? USER_DATA3_SET :
1145                                         param, param_name, show_rl(estate_rl(sm
1368        } END_FOR_EACH_SM(sm);

1370        /* This if for "return foo;" where "foo->bar" is user data. */
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        }

1371        FOR_EACH_MY_SM(my_id, __get_cur_stree(), sm) {
1372                if (!ret_sym)
1373                        break;
1374                if (ret_sym != sm->sym)
1375                        continue;

1377                param_name = state_name_to_param_name(sm->name, return_str);
1378                if (!param_name)
1379                        continue;
1380                if (strcmp(param_name, "$") == 0)
1381                        return_found = true;
1382                if (strcmp(param_name, "*$") == 0)
1383                        pointed_at_found = true;
1384                snprintf(buf, sizeof(buf), "%s%s",
1385                         show_rl(estate_rl(sm->state)),
1386                         estate_capped(sm->state) ? "[c]" : "");
1387                sql_insert_return_states(return_id, return_ranges,
1388                                         func_gets_user_data ? USER_DATA_SET : U
1389                                         -1, param_name, buf);
1169                                         func_gets_user_data ? USER_DATA3_SET :
1170                                         -1, param_name, show_rl(estate_rl(sm->s
1390        } END_FOR_EACH_SM(sm);

1392        /* This if for "return ntohl(foo);" */

1393        if (!return_found && get_user_rl(expr, &rl)) {
1394                snprintf(buf, sizeof(buf), "%s%s",
1395                         show_rl(rl), user_rl_capped(expr) ? "[c]" : "");
1396                sql_insert_return_states(return_id, return_ranges,
1397                                         func_gets_user_data ? USER_DATA_SET : U
1398                                         -1, "$", buf);
1176                                         func_gets_user_data ? USER_DATA3_SET :
1177                                         -1, "$", show_rl(rl));
1178                goto free_string;
1399        }

1401        /*
1402         * This is to handle things like return skb->data where we don't set a
1403         * state for that.
1404         */
```

```
1405         if (!pointed_at_found && points_to_user_data(expr)) {
1406                 sql_insert_return_states(return_id, return_ranges,
1407                                          (is_skb_data(expr) || func_gets_user_da
1408                                          USER_DATA_SET : USER_DATA,
1409                                          -1, "*$", "s64min-s64max");
1410         }

1181 free_string:
1412         free_string(return_str);
1413 }

1415 static void returns_param_capped(struct expression *expr, int param, char *key,
1416 {
1417         struct smatch_state *state, *new;
1418         struct symbol *sym;
1419         char *name;

1421         name = return_state_to_var_sym(expr, param, key, &sym);
1422         if (!name || !sym)
1423                 goto free;

1425         state = get_state(my_id, name, sym);
1426         if (!state || estate_capped(state))
1427                 goto free;

1429         new = clone_estate(state);
1430         estate_set_capped(new);

1432         set_state(my_id, name, sym, new);
1433 free:
1434         free_string(name);
1435 }

1437 static struct int_stack *gets_data_stack;
1438 static void match_function_def(struct symbol *sym)
1439 {
1440         func_gets_user_data = false;
1441 }
______unchanged_portion_omitted_

1453 void register_kernel_user_data(int id)
1201 void register_kernel_user_data2(int id)
1454 {
1455         int i;

1457         my_id = id;

1459         if (option_project != PROJ_KERNEL)
1460                 return;

1462         set_dynamic_states(my_id);

1464         add_hook(&match_function_def, FUNC_DEF_HOOK);
1465         add_hook(&match_inline_start, INLINE_FN_START);
1466         add_hook(&match_inline_end, INLINE_FN_END);

1468         add_hook(&save_start_states, AFTER_DEF_HOOK);
1469         add_hook(&free_start_states, AFTER_FUNC_HOOK);
1470         add_hook(&match_save_states, INLINE_FN_START);
1471         add_hook(&match_restore_states, INLINE_FN_END);

1473         add_unmatched_state_hook(my_id, &empty_state);
1474         add_extra_nomod_hook(&extra_nomod_hook);
1475         add_pre_merge_hook(my_id, &pre_merge_hook);
1476         add_merge_hook(my_id, &merge_estates);
```



```
1478         add_function_hook("copy_from_user", &match_user_copy, INT_PTR(0));
1479         add_function_hook("__copy_from_user", &match_user_copy, INT_PTR(0));
1480         add_function_hook("memcpy_fromiovec", &match_user_copy, INT_PTR(0));
1481         for (i = 0; i < ARRAY_SIZE(kstr_funcs); i++)
1482                 add_function_hook(kstr_funcs[i], &match_user_copy, INT_PTR(2));
1483         add_function_hook("usb_control_msg", &match_user_copy, INT_PTR(6));

1485         for (i = 0; i < ARRAY_SIZE(returns_user_data); i++) {
1486                 add_function_assign_hook(returns_user_data[i], &match_user_assig
1487                 add_function_hook(returns_user_data[i], &match_returns_user_rl,
1488         }

1490         add_function_hook("sscanf", &match_sscanf, NULL);

1492         add_hook(&match_syscall_definition, AFTER_DEF_HOOK);

1494         add_hook(&match_assign, ASSIGNMENT_HOOK);
1495         select_return_states_hook(PARAM_SET, &db_param_set);
1496         add_hook(&match_condition, CONDITION_HOOK);

1498         add_hook(&match_call_info, FUNCTION_CALL_HOOK);
1499         add_member_info_callback(my_id, struct_member_callback);
1500         select_caller_info_hook(set_param_user_data, USER_DATA);
1501         select_return_states_hook(USER_DATA, &returns_param_user_data);
1502         select_return_states_hook(USER_DATA_SET, &returns_param_user_data_set);
1503         select_return_states_hook(CAPPED_DATA, &returns_param_capped);
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);
1504         add_split_return_callback(&param_set_to_user_data);
1505 }

1507 void register_kernel_user_data2(int id)
1251 void register_kernel_user_data3(int id)
1508 {
1509         my_call_id = id;

1511         if (option_project != PROJ_KERNEL)
1512                 return;
1513         select_caller_info_hook(set_called, INTERNAL);
1514 }
```

```
**********************************************************
    2738 Mon Aug  5 08:38:36 2019
new/usr/src/tools/smatch/src/smatch_links.c
11506 smatch resync
**********************************************************
______unchanged_portion_omitted_

 100 void set_up_link_functions(int id, int link_id)
 101 {
 102         if (id + 1 != link_id)
 103                 sm_fatal("FATAL ERROR: links need to be registered directly afte

 105         set_dynamic_states(link_id);
 106         add_merge_hook(link_id, &merge_link_states);
 107         add_modification_hook(link_id, &match_link_modify);
 108         // free link at the end of function
 109 }
______unchanged_portion_omitted_
```

****
6072 Mon Aug 5 08:38:36 2019
new/usr/src/tools/smatch/src/smatch_local_values.c
11506 smatch resync
****
_____***unchanged_portion_omitted_***

```
 230 void register_local_values(int id)
 231 {
 232         my_id = id;

 234         if (!option_info)
 235                 return;

 237         set_dynamic_states(my_id);
 238         add_extra_mod_hook(&extra_mod_hook);
 239         add_unmatched_state_hook(my_id, &unmatched_state);
 240         add_merge_hook(my_id, &merge_estates);
 241         all_return_states_hook(&process_states);
 242         add_hook(match_end_file, END_FILE_HOOK);
 243         mem_sql(NULL, NULL, "alter table local_values add column symbol integer;
 244 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   45236 Mon Aug  5 08:38:36 2019
new/usr/src/tools/smatch/src/smatch_math.c
11506 smatch resync
**********************************************************
```

```
/*
 * 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 bool get_rl_sval(struct expression *expr, int implied, int *recurse_cnt,
static bool get_rl_internal(struct expression *expr, int implied, int *recurse_c
static bool handle_variable(struct expression *expr, int implied, int *recurse_c
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 bool get_implied_value_internal(struct expression *expr, int *recurse_cnt
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;
}
```

*unchanged_portion_omitted_*

```
static bool last_stmt_rl(struct statement *stmt, int implied, int *recurse_cnt,
static struct range_list *last_stmt_rl(struct statement *stmt, int implied, int
{
        struct expression *expr;

        if (!stmt)
                return false;
                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 false;
                        return NULL;
        } else if (stmt->type == STMT_EXPRESSION) {
                expr = stmt->expression;
        } else {
                return false;
                return NULL;
        }
        return get_rl_sval(expr, implied, recurse_cnt, res, res_sval);
        return _get_rl(expr, implied, recurse_cnt);
}

static bool handle_expression_statement_rl(struct expression *expr, int implied,
                int *recurse_cnt, struct range_list **res, sval_t *res_sval)
static struct range_list *handle_expression_statement_rl(struct expression *expr
{
        return last_stmt_rl(get_expression_statement(expr), implied, recurse_cnt
        return last_stmt_rl(get_expression_statement(expr), implied, recurse_cnt
}

static bool handle_address(struct expression *expr, int implied, int *recurse_cn
static struct range_list *handle_ampersand_rl(struct expression *expr, int impli
{
        struct range_list *rl;
        static int recursed;
        sval_t sval;

        if (recursed > 10)
                return false;
        if (implied == RL_EXACT)
                return false;

        if (custom_handle_variable) {
                rl = custom_handle_variable(expr);
                if (rl) {
                        *res = rl;
                        return true;
                }
        }

        recursed++;
        if (get_mtag_sval(expr, &sval)) {
                recursed--;
                *res_sval = sval;
                return true;
        }

        if (get_address_rl(expr, res)) {
                recursed--;
                return true;
        }
        recursed--;
        return 0;
        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 bool handle_ampersand_rl(struct expression *expr, int implied, int *recur
static struct range_list *handle_negate_rl(struct expression *expr, int implied,
{
        return handle_address(expr, implied, recurse_cnt, res, res_sval);
```

```
127 }
104         if (known_condition_true(expr->unop))
105                 return rl_zero();
106         if (known_condition_false(expr->unop))
107                 return rl_one();

129 static bool handle_negate_rl(struct expression *expr, int implied, int *recurse_
130 {
131         if (known_condition_true(expr->unop)) {
132                 *res_sval = zero;
133                 return true;
134         }
135         if (known_condition_false(expr->unop)) {
136                 *res_sval = one;
137                 return true;
138         }

140         if (implied == RL_EXACT)
141                 return false;
110                 return NULL;

143         if (implied_condition_true(expr->unop)) {
144                 *res_sval = zero;
145                 return true;
146         }
147         if (implied_condition_false(expr->unop)) {
148                 *res_sval = one;
149                 return true;
150         }

152         *res = alloc_rl(zero, one);
153         return true;
112         if (implied_condition_true(expr->unop))
113                 return rl_zero();
114         if (implied_condition_false(expr->unop))
115                 return rl_one();
116         return alloc_rl(zero, one);
154 }

156 static bool handle_bitwise_negate(struct expression *expr, int implied, int *rec
119 static struct range_list *handle_bitwise_negate(struct expression *expr, int imp
157 {
158         struct range_list *rl;
159         sval_t sval = {};
122         sval_t sval;

161         if (!get_rl_sval(expr->unop, implied, recurse_cnt, &rl, &sval))
162                 return false;
163         if (!sval.type && !rl_to_sval(rl, &sval))
164                 return false;
124         rl = _get_rl(expr->unop, implied, recurse_cnt);
125         if (!rl_to_sval(rl, &sval))
126                 return NULL;
165         sval = sval_preop(sval, '~');
166         sval_cast(get_type(expr->unop), sval);
167         *res_sval = sval;
168         return true;
129         return alloc_rl(sval, sval);
169 }

171 static bool untrusted_type_min(struct expression *expr)
132 static struct range_list *handle_minus_preop(struct expression *expr, int implie
172 {
173         struct range_list *rl;
135         sval_t min, max;
```



```
175         rl = var_user_rl(expr);
176         return rl && sval_is_min(rl_min(rl));
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);
177 }

179 static bool handle_minus_preop(struct expression *expr, int implied, int *recurs
143 static struct range_list *handle_preop_rl(struct expression *expr, int implied,
180 {
181         struct range_list *rl;
182         struct range_list *ret = NULL;
183         struct symbol *type;
184         sval_t neg_one = { 0 };
185         sval_t zero = { 0 };
186         sval_t sval = {};

188         neg_one.value = -1;
189         zero.value = 0;

191         if (!get_rl_sval(expr->unop, implied, recurse_cnt, &rl, &sval))
192                 return false;
193         if (sval.type) {
194                 *res_sval = sval_preop(sval, '-');
195                 return true;
196         }
197         /*
198          * One complication is that -INT_MIN is still INT_MIN because of integer
199          * overflows...  But how many times do we set a time out to INT_MIN?
200          * So normally when we call abs() then it does return a positive value.
201          *
202          */
203         type = rl_type(rl);
204         neg_one.type = zero.type = type;

206         if (sval_is_negative(rl_min(rl))) {
207                 struct range_list *neg;
208                 struct data_range *drange;
209                 sval_t new_min, new_max;

211                 neg = alloc_rl(sval_type_min(type), neg_one);
212                 neg = rl_intersection(rl, neg);

214                 if (sval_is_min(rl_min(neg)) && !sval_is_min(rl_max(neg)))
215                         neg = remove_range(neg, sval_type_min(type), sval_type_m

217                 FOR_EACH_PTR(neg, drange) {
218                         new_min = drange->max;
219                         new_min.value = -new_min.value;
220                         new_max = drange->min;
221                         new_max.value = -new_max.value;
222                         add_range(&ret, new_min, new_max);
223                 } END_FOR_EACH_PTR(drange);

225                 if (untrusted_type_min(expr))
226                         add_range(&ret, sval_type_min(type), sval_type_min(type)
227         }

229         if (!sval_is_negative(rl_max(rl))) {
230                 struct range_list *pos;
231                 struct data_range *drange;
232                 sval_t new_min, new_max;

234                 pos = alloc_rl(zero, sval_type_max(type));
235                 pos = rl_intersection(rl, pos);
```

```
237                 FOR_EACH_PTR(pos, drange) {
238                         new_min = drange->max;
239                         new_min.value = -new_min.value;
240                         new_max = drange->min;
241                         new_max.value = -new_max.value;
242                         add_range(&ret, new_min, new_max);
243                 } END_FOR_EACH_PTR(drange);
244         }

246         *res = ret;
247         return true;
248 }

250 static bool handle_preop_rl(struct expression *expr, int implied, int *recurse_c
251 {
252         switch (expr->op) {
253         case '&':
254                 return handle_ampersand_rl(expr, implied, recurse_cnt, res, res_
147                 return handle_ampersand_rl(expr, implied, recurse_cnt);
255         case '!':
256                 return handle_negate_rl(expr, implied, recurse_cnt, res, res_sva
149                 return handle_negate_rl(expr, implied, recurse_cnt);
257         case '~':
258                 return handle_bitwise_negate(expr, implied, recurse_cnt, res_sva
151                 return handle_bitwise_negate(expr, implied, recurse_cnt);
259         case '-':
260                 return handle_minus_preop(expr, implied, recurse_cnt, res, res_s
153                 return handle_minus_preop(expr, implied, recurse_cnt);
261         case '*':
262                 return handle_variable(expr, implied, recurse_cnt, res, res_sval
155                 return handle_variable(expr, implied, recurse_cnt);
263         case '(':
264                 return handle_expression_statement_rl(expr, implied, recurse_cnt
157                 return handle_expression_statement_rl(expr, implied, recurse_cnt
265         default:
266                 return false;
159                 return NULL;
267         }
268 }

270 static bool handle_divide_rl(struct expression *expr, int implied, int *recurse_
163 static struct range_list *handle_divide_rl(struct expression *expr, int implied,
271 {
272         struct range_list *left_rl = NULL;
273         struct range_list *right_rl = NULL;
165         struct range_list *left_rl, *right_rl;
274         struct symbol *type;

276         type = get_type(expr);

278         get_rl_internal(expr->left, implied, recurse_cnt, &left_rl);
170         left_rl = _get_rl(expr->left, implied, recurse_cnt);
279         left_rl = cast_rl(type, left_rl);
280         get_rl_internal(expr->right, implied, recurse_cnt, &right_rl);
172         right_rl = _get_rl(expr->right, implied, recurse_cnt);
281         right_rl = cast_rl(type, right_rl);

283         if (!left_rl || !right_rl)
284                 return false;
176                 return NULL;

286         if (implied != RL_REAL_ABSOLUTE) {
287                 if (is_whole_rl(left_rl) || is_whole_rl(right_rl))
288                         return false;
180                         return NULL;
```



```
289         }

291         *res = rl_binop(left_rl, '/', right_rl);
292         return true;
183         return rl_binop(left_rl, '/', right_rl);
293 }
______unchanged_portion_omitted_

336 static bool handle_subtract_rl(struct expression *expr, int implied, int *recurs
227 static struct range_list *handle_subtract_rl(struct expression *expr, int implie
337 {
338         struct symbol *type;
339         struct range_list *left_orig, *right_orig;
340         struct range_list *left_rl, *right_rl;
341         sval_t min, max, tmp;
232         sval_t max, min, tmp;
342         int comparison;
343         int offset;

345         type = get_type(expr);

347         offset = handle_offset_subtraction(expr);
348         if (offset >= 0) {
349                 tmp.type = type;
350                 tmp.value = offset;

352                 *res = alloc_rl(tmp, tmp);
353                 return true;
243                 return alloc_rl(tmp, tmp);
354         }

356         comparison = get_comparison(expr->left, expr->right);

358         left_orig = NULL;
359         get_rl_internal(expr->left, implied, recurse_cnt, &left_orig);
248         left_orig = _get_rl(expr->left, implied, recurse_cnt);
360         left_rl = cast_rl(type, left_orig);
361         right_orig = NULL;
362         get_rl_internal(expr->right, implied, recurse_cnt, &right_orig);
250         right_orig = _get_rl(expr->right, implied, recurse_cnt);
363         right_rl = cast_rl(type, right_orig);

365         if ((!left_rl || !right_rl) &&
366             (implied == RL_EXACT || implied == RL_HARD || implied == RL_FUZZY))
367                 return false;
255                 return NULL;

369         if (!left_rl)
370                 left_rl = alloc_whole_rl(type);
371         if (!right_rl)
372                 right_rl = alloc_whole_rl(type);

374         /* negative values complicate everything fix this later */
375         if (sval_is_negative(rl_min(right_rl)))
376                 return false;
264                 return NULL;
377         max = rl_max(left_rl);
378         min = sval_type_min(type);

380         switch (comparison) {
381         case '>':
382         case SPECIAL_UNSIGNED_GT:
383                 min = sval_type_val(type, 1);
384                 max = rl_max(left_rl);
385                 break;
386         case SPECIAL_GTE:
```

```
387         case SPECIAL_UNSIGNED_GTE:
388                 min = sval_type_val(type, 0);
389                 max = rl_max(left_rl);
390                 break;
391         case SPECIAL_EQUAL:
392                 min = sval_type_val(type, 0);
393                 max = sval_type_val(type, 0);
394                 break;
395         case '<':
396         case SPECIAL_UNSIGNED_LT:
397                 max = sval_type_val(type, -1);
398                 break;
399         case SPECIAL_LTE:
400         case SPECIAL_UNSIGNED_LTE:
401                 max = sval_type_val(type, 0);
402                 break;
403         default:
404                 if (!left_orig || !right_orig)
405                         return false;
406                 *res = rl_binop(left_rl, '-', right_rl);
407                 return true;
293                         return NULL;
294                 return rl_binop(left_rl, '-', right_rl);
408         }

410         if (!sval_binop_overflows(rl_min(left_rl), '-', rl_max(right_rl))) {
411                 tmp = sval_binop(rl_min(left_rl), '-', rl_max(right_rl));
412                 if (sval_cmp(tmp, min) > 0)
413                         min = tmp;
414         }

416         if (!sval_is_max(rl_max(left_rl))) {
417                 tmp = sval_binop(rl_max(left_rl), '-', rl_min(right_rl));
418                 if (sval_cmp(tmp, max) < 0)
419                         max = tmp;
420         }

422         if (sval_is_min(min) && sval_is_max(max))
423                 return false;
310                 return NULL;

425         *res = cast_rl(type, alloc_rl(min, max));
426         return true;
312         return cast_rl(type, alloc_rl(min, max));
427 }

429 static bool handle_mod_rl(struct expression *expr, int implied, int *recurse_cnt
315 static struct range_list *handle_mod_rl(struct expression *expr, int implied, in
430 {
431         struct range_list *rl;
432         sval_t left, right, sval;

434         if (implied == RL_EXACT) {
435                 if (!get_implied_value(expr->right, &right))
436                         return false;
322                         return NULL;
437                 if (!get_implied_value(expr->left, &left))
438                         return false;
324                         return NULL;
439                 sval = sval_binop(left, '%', right);
440                 *res = alloc_rl(sval, sval);
441                 return true;
326                 return alloc_rl(sval, sval);
442         }
443         /* if we can't figure out the right side it's probably hopeless */
444         if (!get_implied_value_internal(expr->right, recurse_cnt, &right))
```



```
445                 return false;
329         if (!get_implied_value_internal(expr->right, &right, recurse_cnt))
330                 return NULL;

447         right = sval_cast(get_type(expr), right);
448         right.value--;

450         if (get_rl_internal(expr->left, implied, recurse_cnt, &rl) && rl &&
451             rl_max(rl).uvalue < right.uvalue)
335         rl = _get_rl(expr->left, implied, recurse_cnt);
336         if (rl && rl_max(rl).uvalue < right.uvalue)
452                 right.uvalue = rl_max(rl).uvalue;

454         *res = alloc_rl(sval_cast(right.type, zero), right);
455         return true;
339         return alloc_rl(sval_cast(right.type, zero), right);
456 }

458 static bool handle_bitwise_AND(struct expression *expr, int implied, int *recurs
342 static sval_t sval_lowest_set_bit(sval_t sval)
459 {
344         int i;
345         int found = 0;

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 }

357 static struct range_list *handle_bitwise_AND(struct expression *expr, int implie
358 {
460         struct symbol *type;
461         struct range_list *left_rl, *right_rl;
361         sval_t known;
462         int new_recurse;

464         if (implied != RL_IMPLIED && implied != RL_ABSOLUTE && implied != RL_REA
465                 return false;
365                 return NULL;

467         type = get_type(expr);

469         if (!get_rl_internal(expr->left, implied, recurse_cnt, &left_rl))
470                 left_rl = alloc_whole_rl(type);
369         if (get_implied_value_internal(expr->left, &known, recurse_cnt)) {
370                 sval_t min;

372                 min = sval_lowest_set_bit(known);
373                 left_rl = alloc_rl(min, known);
471         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         }

473         new_recurse = *recurse_cnt;
474         if (*recurse_cnt >= 200)
475                 new_recurse = 100;  /* Let's try super hard to get the mask */
476         if (!get_rl_internal(expr->right, implied, &new_recurse, &right_rl))
477                 right_rl = alloc_whole_rl(type);
478         right_rl = cast_rl(type, right_rl);
391         if (get_implied_value_internal(expr->right, &known, &new_recurse)) {
392                 sval_t min, left_max, mod;

479         *recurse_cnt = new_recurse;

481         *res = rl_binop(left_rl, '&', right_rl);
482         return true;
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);
483 }

485 static bool use_rl_binop(struct expression *expr, int implied, int *recurse_cnt,
426 static struct range_list *use_rl_binop(struct expression *expr, int implied, int
486 {
487         struct symbol *type;
488         struct range_list *left_rl, *right_rl;

490         if (implied != RL_IMPLIED && implied != RL_ABSOLUTE && implied != RL_REA
491                 return false;
432                 return NULL;

493         type = get_type(expr);

495         get_absolute_rl_internal(expr->left, &left_rl, recurse_cnt);
496         get_absolute_rl_internal(expr->right, &right_rl, recurse_cnt);
497         left_rl = cast_rl(type, left_rl);
498         right_rl = cast_rl(type, right_rl);
499         if (!left_rl || !right_rl)
500                 return false;
441                 return NULL;

502         *res = rl_binop(left_rl, expr->op, right_rl);
```



```
503         return true;
443         return rl_binop(left_rl, expr->op, right_rl);
504 }

506 static bool handle_right_shift(struct expression *expr, int implied, int *recurs
446 static struct range_list *handle_right_shift(struct expression *expr, int implie
507 {
508         struct range_list *left_rl, *right_rl;
448         struct range_list *left_rl;
449         sval_t right;
509         sval_t min, max;

511         if (implied == RL_EXACT || implied == RL_HARD)
512                 return false;
453                 return NULL;

514         if (get_rl_internal(expr->left, implied, recurse_cnt, &left_rl)) {
455         left_rl = _get_rl(expr->left, implied, recurse_cnt);
456         if (left_rl) {
515                 max = rl_max(left_rl);
516                 min = rl_min(left_rl);
517         } else {
518                 if (implied == RL_FUZZY)
519                         return false;
461                         return NULL;
520                 max = sval_type_max(get_type(expr->left));
521                 min = sval_type_val(get_type(expr->left), 0);
522         }

524         if (get_rl_internal(expr->right, implied, recurse_cnt, &right_rl) &&
525             !sval_is_negative(rl_min(right_rl))) {
526                 min = sval_binop(min, SPECIAL_RIGHTSHIFT, rl_max(right_rl));
527                 max = sval_binop(max, SPECIAL_RIGHTSHIFT, rl_min(right_rl));
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);
528         } else if (!sval_is_negative(min)) {
529                 min.value = 0;
530                 max = sval_type_max(max.type);
531         } else {
532                 return false;
473                 return NULL;
533         }

535         *res = alloc_rl(min, max);
536         return true;
476         return alloc_rl(min, max);
537 }

539 static bool handle_left_shift(struct expression *expr, int implied, int *recurse
479 static struct range_list *handle_left_shift(struct expression *expr, int implied
540 {
541         struct range_list *left_rl, *rl;
481         struct range_list *left_rl, *res;
542         sval_t right;
483         sval_t min, max;
484         int add_zero = 0;

544         if (implied == RL_EXACT || implied == RL_HARD)
545                 return false;
487                 return NULL;
546         /* this is hopeless without the right side */
547         if (!get_implied_value_internal(expr->right, recurse_cnt, &right))
548                 return false;
549         if (!get_rl_internal(expr->left, implied, recurse_cnt, &left_rl)) {
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 {
550                 if (implied == RL_FUZZY)
551                         return false;
552                 left_rl = alloc_whole_rl(get_type(expr->left));
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;
553         }

555         rl = rl_binop(left_rl, SPECIAL_LEFTSHIFT, alloc_rl(right, right));
556         if (!rl)
557                 return false;
558         *res = rl;
559         return true;
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;
560 }

562 static bool handle_known_binop(struct expression *expr, sval_t *res)
515 static struct range_list *handle_known_binop(struct expression *expr)
563 {
564         sval_t left, right;

566         if (!get_value(expr->left, &left))
567                 return false;
520                 return NULL;
568         if (!get_value(expr->right, &right))
569                 return false;
570         *res = sval_binop(left, expr->op, right);
571         return true;
522                 return NULL;
523         left = sval_binop(left, expr->op, right);
524         return alloc_rl(left, left);
572 }
_____unchanged_portion_omitted_

616 static bool handle_binop_rl_helper(struct expression *expr, int implied, int *re
569 static struct range_list *handle_binop_rl(struct expression *expr, int implied,
617 {
571         struct smatch_state *state;
618         struct symbol *type;
619         struct range_list *left_rl = NULL;
620         struct range_list *right_rl = NULL;
621         struct range_list *rl;
573         struct range_list *left_rl, *right_rl, *rl;
622         sval_t min, max;

624         type = get_promoted_type(get_type(expr->left), get_type(expr->right));
625         get_rl_internal(expr->left, implied, recurse_cnt, &left_rl);
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);
626         left_rl = cast_rl(type, left_rl);
627         get_rl_internal(expr->right, implied, recurse_cnt, &right_rl);
597         right_rl = _get_rl(expr->right, implied, recurse_cnt);
628         right_rl = cast_rl(type, right_rl);

629         if (!left_rl && !right_rl)
630                 return false;
601                 return NULL;

632         rl = handle_implied_binop(left_rl, expr->op, right_rl);
633         if (rl) {
634                 *res = rl;
635                 return true;
636         }
604         if (rl)
605                 return rl;

638         switch (expr->op) {
639         case '%':
640                 return handle_mod_rl(expr, implied, recurse_cnt, res);
609                 return handle_mod_rl(expr, implied, recurse_cnt);
641         case '&':
642                 return handle_bitwise_AND(expr, implied, recurse_cnt, res);
611                 return handle_bitwise_AND(expr, implied, recurse_cnt);
643         case '|':
644         case '^':
645                 return use_rl_binop(expr, implied, recurse_cnt, res);
614                 return use_rl_binop(expr, implied, recurse_cnt);
646         case SPECIAL_RIGHTSHIFT:
647                 return handle_right_shift(expr, implied, recurse_cnt, res);
616                 return handle_right_shift(expr, implied, recurse_cnt);
648         case SPECIAL_LEFTSHIFT:
649                 return handle_left_shift(expr, implied, recurse_cnt, res);
618                 return handle_left_shift(expr, implied, recurse_cnt);
650         case '-':
651                 return handle_subtract_rl(expr, implied, recurse_cnt, res);
620                 return handle_subtract_rl(expr, implied, recurse_cnt);
652         case '/':
653                 return handle_divide_rl(expr, implied, recurse_cnt, res);
622                 return handle_divide_rl(expr, implied, recurse_cnt);
654         }

656         if (!left_rl || !right_rl)
657                 return false;
626                 return NULL;

659         if (sval_binop_overflows(rl_min(left_rl), expr->op, rl_min(right_rl)))
660                 return false;
629                 return NULL;
```

```
661         if (sval_binop_overflows(rl_max(left_rl), expr->op, rl_max(right_rl)))
662                 return false;
631                 return NULL;

664         min = sval_binop(rl_min(left_rl), expr->op, rl_min(right_rl));
665         max = sval_binop(rl_max(left_rl), expr->op, rl_max(right_rl));

667         *res = alloc_rl(min, max);
668         return true;

636         return alloc_rl(min, max);
670 }

672 static bool handle_binop_rl(struct expression *expr, int implied, int *recurse_c
673 {
674         struct smatch_state *state;
675         struct range_list *rl;
676         sval_t val;

678         if (handle_known_binop(expr, &val)) {
679                 *res_sval = val;
680                 return true;
681         }
682         if (implied == RL_EXACT)
683                 return false;

685         if (custom_handle_variable) {
686                 rl = custom_handle_variable(expr);
687                 if (rl) {
688                         *res = rl;
689                         return true;
690                 }
691         }

693         state = get_extra_state(expr);
694         if (state && !is_whole_rl(estate_rl(state))) {
695                 if (implied != RL_HARD || estate_has_hard_max(state)) {
696                         *res = clone_rl(estate_rl(state));
697                         return true;
698                 }
699         }

701         return handle_binop_rl_helper(expr, implied, recurse_cnt, res, res_sval)
702 }

704 static int do_comparison(struct expression *expr)
705 {
706         struct range_list *left_ranges = NULL;
707         struct range_list *right_ranges = NULL;
708         int poss_true, poss_false;
709         struct symbol *type;

711         type = get_type(expr);
712         get_absolute_rl(expr->left, &left_ranges);
713         get_absolute_rl(expr->right, &right_ranges);

715         left_ranges = cast_rl(type, left_ranges);
716         right_ranges = cast_rl(type, right_ranges);

718         poss_true = possibly_true_rl(left_ranges, expr->op, right_ranges);
719         poss_false = possibly_false_rl(left_ranges, expr->op, right_ranges);

721         if (!poss_true && !poss_false)
722                 return 0x0;
723         if (poss_true && !poss_false)
724                 return 0x1;
```



```
725         if (!poss_true && poss_false)
726                 return 0x2;
727         return 0x3;
728 }

730 static bool handle_comparison_rl(struct expression *expr, int implied, int *recu
665 static struct range_list *handle_comparison_rl(struct expression *expr, int impl
731 {
732         sval_t left, right;
733         int cmp;
668         int res;

735         if (expr->op == SPECIAL_EQUAL && expr->left->type == EXPR_TYPE) {
736                 struct symbol *left, *right;

738                 if (expr->right->type != EXPR_TYPE)
739                         return false;

741                 left = get_real_base_type(expr->left->symbol);
742                 right = get_real_base_type(expr->right->symbol);
743                 if (type_bits(left) == type_bits(right) &&
744                     type_positive_bits(left) == type_positive_bits(right))
745                         *res_sval = one;
746                 else
747                         *res_sval = zero;
748                 return true;
674                 right = get_real_base_type(expr->left->symbol);
675                 if (left == right)
676                         return rl_one();
677                 return rl_zero();
749         }

751         if (get_value(expr->left, &left) && get_value(expr->right, &right)) {
752                 struct data_range tmp_left, tmp_right;

754                 tmp_left.min = left;
755                 tmp_left.max = left;
756                 tmp_right.min = right;
757                 tmp_right.max = right;
758                 if (true_comparison_range(&tmp_left, expr->op, &tmp_right))
759                         *res_sval = one;
760                 else
761                         *res_sval = zero;
762                 return true;
688                         return rl_one();
689                 return rl_zero();
763         }

765         if (implied == RL_EXACT)
766                 return false;
693                 return NULL;

768         cmp = do_comparison(expr);
769         if (cmp == 1) {
770                 *res_sval = one;
771                 return true;
772         }
773         if (cmp == 2) {
774                 *res_sval = zero;
775                 return true;
776         }
695         res = do_comparison(expr);
696         if (res == 1)
697                 return rl_one();
698         if (res == 2)
699                 return rl_zero();
```

```
778         *res = alloc_rl(zero, one);
779         return true;
701         return alloc_rl(zero, one);
780 }

782 static bool handle_logical_rl(struct expression *expr, int implied, int *recurse
704 static struct range_list *handle_logical_rl(struct expression *expr, int implied
783 {
784         sval_t left, right;
785         int left_known = 0;
786         int right_known = 0;

788         if (implied == RL_EXACT) {
789                 if (get_value(expr->left, &left))
790                         left_known = 1;
791                 if (get_value(expr->right, &right))
792                         right_known = 1;
793         } else {
794                 if (get_implied_value_internal(expr->left, recurse_cnt, &left))
716                 if (get_implied_value_internal(expr->left, &left, recurse_cnt))
795                         left_known = 1;
796                 if (get_implied_value_internal(expr->right, recurse_cnt, &right)
718                 if (get_implied_value_internal(expr->right, &right, recurse_cnt)
797                         right_known = 1;
798         }

800         switch (expr->op) {
801         case SPECIAL_LOGICAL_OR:
802                 if (left_known && left.value)
803                         goto one;
725                         return rl_one();
804                 if (right_known && right.value)
805                         goto one;
727                         return rl_one();
806                 if (left_known && right_known)
807                         goto zero;
729                         return rl_zero();
808                 break;
809         case SPECIAL_LOGICAL_AND:
810                 if (left_known && right_known) {
811                         if (left.value && right.value)
812                                 goto one;
813                         goto zero;
734                                 return rl_one();
735                         return rl_zero();
814                 }
815                 break;
816         default:
817                 return false;
739                 return NULL;
818         }

820         if (implied == RL_EXACT)
821                 return false;
743                 return NULL;

823         *res = alloc_rl(zero, one);
824         return true;

826 zero:
827         *res_sval = zero;
828         return true;
829 one:
830         *res_sval = one;
831         return true;
```



```
745         return alloc_rl(zero, one);
832 }

834 static bool handle_conditional_rl(struct expression *expr, int implied, int *rec
748 static struct range_list *handle_conditional_rl(struct expression *expr, int imp
835 {
836         struct expression *cond_true;
837         struct range_list *true_rl, *false_rl;
838         struct symbol *type;
839         int final_pass_orig = final_pass;

841         cond_true = expr->cond_true;
842         if (!cond_true)
843                 cond_true = expr->conditional;

845         if (known_condition_true(expr->conditional))
846                 return get_rl_sval(cond_true, implied, recurse_cnt, res, res_sva
760                 return _get_rl(cond_true, implied, recurse_cnt);
847         if (known_condition_false(expr->conditional))
848                 return get_rl_sval(expr->cond_false, implied, recurse_cnt, res,
762                 return _get_rl(expr->cond_false, implied, recurse_cnt);

850         if (implied == RL_EXACT)
851                 return false;
765                 return NULL;

853         if (implied_condition_true(expr->conditional))
854                 return get_rl_sval(cond_true, implied, recurse_cnt, res, res_sva
768                 return _get_rl(cond_true, implied, recurse_cnt);
855         if (implied_condition_false(expr->conditional))
856                 return get_rl_sval(expr->cond_false, implied, recurse_cnt, res,
770                 return _get_rl(expr->cond_false, implied, recurse_cnt);


858         /* this becomes a problem with deeply nested conditional statements */
859         if (low_on_memory())
860                 return false;
775                 return NULL;

862         type = get_type(expr);

864         __push_fake_cur_stree();
865         final_pass = 0;
866         __split_whole_condition(expr->conditional);
867         true_rl = NULL;
868         get_rl_internal(cond_true, implied, recurse_cnt, &true_rl);
782         true_rl = _get_rl(cond_true, implied, recurse_cnt);
869         __push_true_states();
870         __use_false_states();
871         false_rl = NULL;
872         get_rl_internal(expr->cond_false, implied, recurse_cnt, &false_rl);
785         false_rl = _get_rl(expr->cond_false, implied, recurse_cnt);
873         __merge_true_states();
874         __free_fake_cur_stree();
875         final_pass = final_pass_orig;

877         if (!true_rl || !false_rl)
878                 return false;
791                 return NULL;
879         true_rl = cast_rl(type, true_rl);
880         false_rl = cast_rl(type, false_rl);

882         *res = rl_union(true_rl, false_rl);
883         return true;
795         return rl_union(true_rl, false_rl);
884 }
```

```
886 static bool get_fuzzy_max_helper(struct expression *expr, sval_t *max)
798 static int get_fuzzy_max_helper(struct expression *expr, sval_t *max)
887 {
888         struct smatch_state *state;
889         sval_t sval;

891         if (get_hard_max(expr, &sval)) {
892                 *max = sval;
893                 return true;
805                 return 1;
894         }

896         state = get_extra_state(expr);
897         if (!state || !estate_has_fuzzy_max(state))
898                 return false;
810                 return 0;
899         *max = sval_cast(get_type(expr), estate_get_fuzzy_max(state));
900         return true;
812         return 1;
901 }

903 static bool get_fuzzy_min_helper(struct expression *expr, sval_t *min)
815 static int get_fuzzy_min_helper(struct expression *expr, sval_t *min)
904 {
905         struct smatch_state *state;
906         sval_t sval;

908         state = get_extra_state(expr);
909         if (!state || !estate_rl(state))
910                 return false;
822                 return 0;

912         sval = estate_min(state);
913         if (sval_is_negative(sval) && sval_is_min(sval))
914                 return false;
826                 return 0;

916         if (sval_is_max(sval))
917                 return false;
829                 return 0;

919         *min = sval_cast(get_type(expr), sval);
920         return true;
832         return 1;
921 }
_____unchanged_portion_omitted_

964 static bool handle_variable(struct expression *expr, int implied, int *recurse_c
876 static struct range_list *handle_variable(struct expression *expr, int implied,
965 {
966         struct smatch_state *state;
967         struct range_list *rl;
968         sval_t sval, min, max;
969         struct symbol *type;

971         if (get_const_value(expr, &sval)) {
972                 *res_sval = sval;
973                 return true;
974         }
883         if (get_const_value(expr, &sval))
884                 return alloc_rl(sval, sval);

976         if (implied == RL_EXACT)
977                 return false;
```



```
979         if (custom_handle_variable) {
980                 rl = custom_handle_variable(expr);
981                 if (rl) {
982                         if (!rl_to_sval(rl, res_sval))
983                                 *res = rl;
984                 } else {
985                         *res = var_to_absolute_rl(expr);
888                 if (!rl)
889                         return var_to_absolute_rl(expr);
890                 return rl;
986                 }
987                 return true;
988         }

990         if (get_mtag_sval(expr, &sval)) {
991                 *res_sval = sval;
992                 return true;
993         }
893         if (implied == RL_EXACT)
894                 return NULL;

896         if (get_mtag_sval(expr, &sval))
897                 return alloc_rl(sval, sval);

995         type = get_type(expr);
996         if (type &&
997             (type->type == SYM_ARRAY ||
998              type->type == SYM_FN))
999                 return handle_address(expr, implied, recurse_cnt, res, res_sval)
900         if (type && type->type == SYM_FN)
901                 return alloc_rl(fn_ptr_min, fn_ptr_max);

1001        /* FIXME: call rl_to_sval() on the results */

1003        switch (implied) {
1004        case RL_HARD:
1005        case RL_IMPLIED:
1006        case RL_ABSOLUTE:
1007                state = get_extra_state(expr);
1008                if (!state) {
 908                if (!state || !state->data) {
1009                        if (implied == RL_HARD)
1010                                return false;
1011                        if (get_local_rl(expr, res))
1012                                return true;
1013                        if (get_mtag_rl(expr, res))
1014                                return true;
1015                        if (get_db_type_rl(expr, res))
1016                                return true;
1017                        if (is_array(expr) && get_array_rl(expr, res))
1018                                return true;
1019                        return false;
 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;
1020                }
1021                if (implied == RL_HARD && !estate_has_hard_max(state))
1022                        return false;
1023                *res = clone_rl(estate_rl(state));
```

```
1024                return true;
 922                        return NULL;
 923                return clone_rl(estate_rl(state));
1025        case RL_REAL_ABSOLUTE: {
1026                struct smatch_state *abs_state;

1028                state = get_extra_state(expr);
1029                abs_state = get_real_absolute_state(expr);

1031                if (estate_rl(state) && estate_rl(abs_state)) {
1032                        *res = clone_rl(rl_intersection(estate_rl(state),
 931                        return clone_rl(rl_intersection(estate_rl(state),
1033                                                        estate_rl(abs_state)));
1034                        return true;
1035                } else if (estate_rl(state)) {
1036                        *res = clone_rl(estate_rl(state));
1037                        return true;
 934                        return clone_rl(estate_rl(state));
1038                } else if (estate_is_empty(state)) {
1039                        /*
1040                         * FIXME: we don't handle empty extra states correctly.
1041                         *
1042                         * The real abs rl is supposed to be filtered by the
1043                         * extra state if there is one.  We don't bother keeping
1044                         * the abs state in sync all the time because we know it
1045                         * will be filtered later.
1046                         *
1047                         * It's not totally obvious to me how they should be
1048                         * handled.  Perhaps we should take the whole rl and
1049                         * filter by the imaginary states.  Perhaps we should
1050                         * just go with the empty state.
1051                         *
1052                         * Anyway what we currently do is return NULL here and
1053                         * that gets translated into the whole range in
1054                         * get_real_absolute_rl().
1055                         *
1056                         */
1057                        return false;
 954                        return NULL;
1058                } else if (estate_rl(abs_state)) {
1059                        *res = clone_rl(estate_rl(abs_state));
1060                        return true;
 956                        return clone_rl(estate_rl(abs_state));
1061                }

1063                if (get_local_rl(expr, res))
1064                        return true;
1065                if (get_mtag_rl(expr, res))
1066                        return true;
1067                if (get_db_type_rl(expr, res))
1068                        return true;
1069                if (is_array(expr) && get_array_rl(expr, res))
1070                        return true;
1071                return false;
 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;
1072        }
1073        case RL_FUZZY:
1074                if (!get_fuzzy_min_helper(expr, &min))
```



```
1075                        min = sval_type_min(get_type(expr));
1076                if (!get_fuzzy_max_helper(expr, &max))
1077                        return false;
 973                        return NULL;
1078                /* fuzzy ranges are often inverted */
1079                if (sval_cmp(min, max) > 0) {
1080                        sval = min;
1081                        min = max;
1082                        max = sval;
1083                }
1084                *res = alloc_rl(min, max);
1085                return true;
 980                return alloc_rl(min, max);
1086        }
1087        return false;
 982        return NULL;
1088 }
______unchanged_portion_omitted_

1134 static bool handle_strlen(struct expression *expr, int implied, int *recurse_cnt
1029 static struct range_list *handle_strlen(struct expression *expr, int implied, in
1135 {
1031        struct range_list *rl;
1136        struct expression *arg, *tmp;
1137        sval_t tag;
1138        sval_t ret = { .type = &ulong_ctype };
1139        struct range_list *rl;

1036        if (implied == RL_EXACT)
1037                return NULL;

1141        arg = get_argument_from_call_expr(expr->args, 0);
1142        if (!arg)
1143                return false;
1041                return NULL;
1144        if (arg->type == EXPR_STRING) {
1145                ret.value = arg->string->length - 1;
1146                *res_sval = ret;
1147                return true;
1044                return alloc_rl(ret, ret);
1148        }
1149        if (implied == RL_EXACT)
1150                return false;
1151        if (get_implied_value(arg, &tag) &&
1152            (tmp = fake_string_from_mtag(tag.uvalue))) {
1153                ret.value = tmp->string->length - 1;
1154                *res_sval = ret;
1155                return true;
1049                return alloc_rl(ret, ret);
1156        }

1158        if (implied == RL_HARD || implied == RL_FUZZY)
1159                return false;
1053                return NULL;

1161        if (get_implied_return(expr, &rl)) {
1162                *res = rl;
1163                return true;
1164        }
1055        if (get_implied_return(expr, &rl))
1056                return rl;

1166        return false;
1058        return NULL;
1167 }
```

```
static bool handle_builtin_constant_p(struct expression *expr, int implied, int
static struct range_list *handle_builtin_constant_p(struct expression *expr, int
{
        struct expression *arg;
        struct range_list *rl;
        sval_t sval;

        arg = get_argument_from_call_expr(expr->args, 0);
        if (get_rl_internal(arg, RL_EXACT, recurse_cnt, &rl))
                *res_sval = one;
        else
                *res_sval = zero;
        return true;
        rl = _get_rl(arg, RL_EXACT, recurse_cnt);
        if (rl_to_sval(rl, &sval))
                return rl_one();
        return rl_zero();
}

static bool handle__builtin_choose_expr(struct expression *expr, int implied, in
static struct range_list *handle__builtin_choose_expr(struct expression *expr, i
{
        struct expression *const_expr, *expr1, *expr2;
        sval_t sval;

        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 false;
                return NULL;
        if (sval.value)
                return get_rl_sval(expr1, implied, recurse_cnt, res, res_sval);
        else
                return get_rl_sval(expr2, implied, recurse_cnt, res, res_sval);
                return _get_rl(expr1, implied, recurse_cnt);
        return _get_rl(expr2, implied, recurse_cnt);
}

static bool handle_call_rl(struct expression *expr, int implied, int *recurse_cn
static struct range_list *handle_call_rl(struct expression *expr, int implied, i
{
        struct range_list *rl;

        if (sym_name_is("__builtin_constant_p", expr->fn))
                return handle_builtin_constant_p(expr, implied, recurse_cnt, res
                return handle_builtin_constant_p(expr, implied, recurse_cnt);

        if (sym_name_is("__builtin_choose_expr", expr->fn))
                return handle__builtin_choose_expr(expr, implied, recurse_cnt, r
                return handle__builtin_choose_expr(expr, implied, recurse_cnt);

        if (sym_name_is("__builtin_expect", expr->fn) ||
            sym_name_is("__builtin_bswap16", expr->fn) ||
            sym_name_is("__builtin_bswap32", expr->fn) ||
            sym_name_is("__builtin_bswap64", expr->fn)) {
                struct expression *arg;

                arg = get_argument_from_call_expr(expr->args, 0);
                return get_rl_sval(arg, implied, recurse_cnt, res, res_sval);
                return _get_rl(arg, implied, recurse_cnt);
        }

        if (sym_name_is("strlen", expr->fn))
                return handle_strlen(expr, implied, recurse_cnt, res, res_sval);
```



```
                return handle_strlen(expr, implied, recurse_cnt);

        if (implied == RL_EXACT || implied == RL_HARD || implied == RL_FUZZY)
                return false;
                return NULL;

        if (custom_handle_variable) {
                rl = custom_handle_variable(expr);
                if (rl) {
                        *res = rl;
                        return true;
                if (rl)
                        return rl;
                }
        }

        /* Ugh...  get_implied_return() sets *rl to NULL on failure */
        if (get_implied_return(expr, &rl)) {
                *res = rl;
                return true;
        }
        rl = db_return_vals(expr);
        if (rl) {
                *res = rl;
                return true;
        }
        return false;
        if (get_implied_return(expr, &rl))
                return rl;
        return db_return_vals(expr);
}

static bool handle_cast(struct expression *expr, int implied, int *recurse_cnt,
static struct range_list *handle_cast(struct expression *expr, int implied, int
{
        struct range_list *rl;
        struct symbol *type;
        sval_t sval = {};

        type = get_type(expr);
        if (get_rl_sval(expr->cast_expression, implied, recurse_cnt, &rl, &sval)
                if (sval.type)
                        *res_sval = sval_cast(type, sval);
                else
                        *res = cast_rl(type, rl);
                return true;
        }
        if (implied == RL_ABSOLUTE || implied == RL_REAL_ABSOLUTE) {
                *res = alloc_whole_rl(type);
                return true;
        }
        rl = _get_rl(expr->cast_expression, implied, recurse_cnt);
        if (rl)
                return cast_rl(type, rl);
        if (implied == RL_ABSOLUTE || implied == RL_REAL_ABSOLUTE)
                return alloc_whole_rl(type);
        if (implied == RL_IMPLIED && type &&
            type_bits(type) > 0 && type_bits(type) < 32) {
                *res = alloc_whole_rl(type);
                return true;
        }
        return false;
            type_bits(type) > 0 && type_bits(type) < 32)
                return alloc_whole_rl(type);
        return NULL;
}
```

```
1272 static bool get_offset_from_down(struct expression *expr, int implied, int *recu
1144 static struct range_list *_get_rl(struct expression *expr, int implied, int *rec
1273 {
1274         struct expression *index;
1275         struct symbol *type = expr->in;
1276         struct range_list *rl;
1277         struct symbol *field;
1278         int offset = 0;
1279         sval_t sval = { .type = ssize_t_ctype };
1280         sval_t tmp_sval = {};

1282         /*
1283          * FIXME:  I don't really know what I'm doing here.  I wish that I
1284          * could just get rid of the __builtin_offset() function and use:
1285          * "&((struct bpf_prog *)NULL)->insns[fprog->len]" instead...
1286          * Anyway, I have done the minimum ammount of work to get that
1287          * expression to work.
1288          *
1289          */

1291         if (expr->op != '.' || !expr->down ||
1292             expr->down->type != EXPR_OFFSETOF ||
1293             expr->down->op != '[' ||
1294             !expr->down->index)
1295                 return false;

1297         index = expr->down->index;

1299         examine_symbol_type(type);
1300         type = get_real_base_type(type);
1301         if (!type)
1302                 return false;
1303         field = find_identifier(expr->ident, type->symbol_list, &offset);
1304         if (!field)
1305                 return false;

1307         type = get_real_base_type(field);
1308         if (!type || type->type != SYM_ARRAY)
1309                 return false;
1310         type = get_real_base_type(type);

1312         if (get_implied_value_internal(index, recurse_cnt, &sval)) {
1313                 res_sval->type = ssize_t_ctype;
1314                 res_sval->value = offset + sval.value * type_bytes(type);
1315                 return true;
1316         }

1318         if (!get_rl_sval(index, implied, recurse_cnt, &rl, &tmp_sval))
1319                 return false;

1321         /*
1322          * I'm not sure why get_rl_sval() would return an sval when
1323          * get_implied_value_internal() failed but it does when I
1324          * parse drivers/net/ethernet/mellanox/mlx5/core/en/monitor_stats.c
1325          *
1326          */
1327         if (tmp_sval.type) {
1328                 res_sval->type = ssize_t_ctype;
1329                 res_sval->value = offset + sval.value * type_bytes(type);
1330                 return true;
1331         }

1333         sval.value = type_bytes(type);
1334         rl = rl_binop(rl, '*', alloc_rl(sval, sval));
1335         sval.value = offset;
```



```
1336         *res = rl_binop(rl, '+', alloc_rl(sval, sval));
1337         return true;
1338 }

1340 static bool get_offset_from_in(struct expression *expr, int implied, int *recurs
1341 {
1342         struct symbol *type = get_real_base_type(expr->in);
1343         struct symbol *field;
1344         int offset = 0;

1346         if (expr->op != '.' || !type || !expr->ident)
1347                 return false;

1349         field = find_identifier(expr->ident, type->symbol_list, &offset);
1350         if (!field)
1351                 return false;

1353         res_sval->type = size_t_ctype;
1354         res_sval->value = offset;

1356         return true;
1357 }

1359 static bool handle_offsetof_rl(struct expression *expr, int implied, int *recurs
1360 {
1361         if (get_offset_from_down(expr, implied, recurse_cnt, res, res_sval))
1362                 return true;

1364         if (get_offset_from_in(expr, implied, recurse_cnt, res, res_sval))
1365                 return true;

1367         evaluate_expression(expr);
1368         if (expr->type == EXPR_VALUE) {
1369                 *res_sval = sval_from_val(expr, expr->value);
1370                 return true;
1371         }
1372         return false;
1373 }

1375 static bool get_rl_sval(struct expression *expr, int implied, int *recurse_cnt,
1376 {
1377         struct range_list *rl = (void *)-1UL;
1378         struct symbol *type;
1379         sval_t sval = {};
1148         sval_t sval;

1381         type = get_type(expr);
1382         expr = strip_parens(expr);
1383         if (!expr)
1384                 return false;
1153                 return NULL;

1386         if (++(*recurse_cnt) >= 200)
1387                 return false;
1156                 return NULL;

1389         switch(expr->type) {
1390         case EXPR_CAST:
1391         case EXPR_FORCE_CAST:
1392         case EXPR_IMPLIED_CAST:
1393                 handle_cast(expr, implied, recurse_cnt, &rl, &sval);
1162                 rl = handle_cast(expr, implied, recurse_cnt);
1394                 goto out_cast;
1395         }

1397         expr = strip_expr(expr);
```

```
1398         if (!expr)
1399                 return false;
1168                 return NULL;

1401         switch (expr->type) {
1402         case EXPR_VALUE:
1403                 sval = sval_from_val(expr, expr->value);
1173                 rl = alloc_rl(sval, sval);
1404                 break;
1405         case EXPR_PREOP:
1406                 handle_preop_rl(expr, implied, recurse_cnt, &rl, &sval);
1176                 rl = handle_preop_rl(expr, implied, recurse_cnt);
1407                 break;
1408         case EXPR_POSTOP:
1409                 get_rl_sval(expr->unop, implied, recurse_cnt, &rl, &sval);
1179                 rl = _get_rl(expr->unop, implied, recurse_cnt);
1410                 break;
1411         case EXPR_BINOP:
1412                 handle_binop_rl(expr, implied, recurse_cnt, &rl, &sval);
1182                 rl = handle_binop_rl(expr, implied, recurse_cnt);
1413                 break;
1414         case EXPR_COMPARE:
1415                 handle_comparison_rl(expr, implied, recurse_cnt, &rl, &sval);
1185                 rl = handle_comparison_rl(expr, implied, recurse_cnt);
1416                 break;
1417         case EXPR_LOGICAL:
1418                 handle_logical_rl(expr, implied, recurse_cnt, &rl, &sval);
1188                 rl = handle_logical_rl(expr, implied, recurse_cnt);
1419                 break;
1420         case EXPR_PTRSIZEOF:
1421         case EXPR_SIZEOF:
1422                 sval = handle_sizeof(expr);
1193                 rl = alloc_rl(sval, sval);
1423                 break;
1424         case EXPR_SELECT:
1425         case EXPR_CONDITIONAL:
1426                 handle_conditional_rl(expr, implied, recurse_cnt, &rl, &sval);
1197                 rl = handle_conditional_rl(expr, implied, recurse_cnt);
1427                 break;
1428         case EXPR_CALL:
1429                 handle_call_rl(expr, implied, recurse_cnt, &rl, &sval);
1200                 rl = handle_call_rl(expr, implied, recurse_cnt);
1430                 break;
1431         case EXPR_STRING:
1203                 rl = NULL;
1432                 if (get_mtag_sval(expr, &sval))
1205                         rl = alloc_rl(sval, sval);
1433                         break;
1434                 if (implied == RL_EXACT)
1435                         break;
1436                 rl = alloc_rl(valid_ptr_min_sval, valid_ptr_max_sval);
1437                 break;
1438         case EXPR_OFFSETOF:
1439                 handle_offsetof_rl(expr, implied, recurse_cnt, &rl, &sval);
1440                 break;
1441         case EXPR_ALIGNOF:
1442                 evaluate_expression(expr);
1443                 if (expr->type == EXPR_VALUE)
1444                         sval = sval_from_val(expr, expr->value);
1445                 break;
1446         default:
1447                 handle_variable(expr, implied, recurse_cnt, &rl, &sval);
1208                 rl = handle_variable(expr, implied, recurse_cnt);
1448         }

1450 out_cast:
```



```
1451         if (rl == (void *)-1UL)
1452                 rl = NULL;

1454         if (sval.type || (rl && rl_to_sval(rl, &sval))) {
1455                 *sval_res = sval;
1456                 return true;
1457         }
1458         if (implied == RL_EXACT)
1459                 return false;

1461         if (rl) {
1462                 *res = rl;
1463                 return true;
1464         }
1465         if (type && (implied == RL_ABSOLUTE || implied == RL_REAL_ABSOLUTE)) {
1466                 *res = alloc_whole_rl(type);
1467                 return true;
1468         }
1469         return false;
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;
1470 }

1472 static bool get_rl_internal(struct expression *expr, int implied, int *recurse_c
1473 {
1474         struct range_list *rl = NULL;
1475         sval_t sval = {};

1477         if (!get_rl_sval(expr, implied, recurse_cnt, &rl, &sval))
1478                 return false;

1480         if (sval.type)
1481                 *res = alloc_rl(sval, sval);
1482         else
1483                 *res = rl;
1484         return true;
1485 }

1487 static bool get_rl_helper(struct expression *expr, int implied, struct range_lis
1488 {
1489         struct range_list *rl = NULL;
1490         sval_t sval = {};
1491         int recurse_cnt = 0;

1493         if (get_value(expr, &sval)) {
1494                 *res = alloc_rl(sval, sval);
1495                 return true;
1496         }

1498         if (!get_rl_sval(expr, implied, &recurse_cnt, &rl, &sval))
1499                 return false;

1501         if (sval.type)
1502                 *res = alloc_rl(sval, sval);
1503         else
1504                 *res = rl;
1505         return true;
1506 }

1508 struct {
1509         struct expression *expr;
1510         sval_t sval;
1221         struct range_list *rl;
```

```
} cached_results[24];
_____unchanged_portion_omitted_

/*
 * Don't cache EXPR_VALUE because values are fast already.
 *
 */
static bool get_value_literal(struct expression *expr, sval_t *res_sval)
{
        struct expression *tmp;
        int recurse_cnt = 0;

        tmp = strip_expr(expr);
        if (!tmp || tmp->type != EXPR_VALUE)
                return false;

        return get_rl_sval(expr, RL_EXACT, &recurse_cnt, NULL, res_sval);
}

/* returns 1 if it can get a value literal or else returns 0 */
int get_value(struct expression *expr, sval_t *res_sval)
int get_value(struct expression *expr, sval_t *sval)
{
        struct range_list *(*orig_custom_fn)(struct expression *expr);
        struct range_list *rl;
        int recurse_cnt = 0;
        sval_t sval = {};
        sval_t tmp;
        int i;

        if (get_value_literal(expr, res_sval))
                return 1;

        /*
         * This only handles RL_EXACT because other expr statements can be
         * different at different points.  Like the list iterator, for example.
         */
        for (i = 0; i < ARRAY_SIZE(cached_results); i++) {
                if (expr == cached_results[i].expr) {
                        if (cached_results[i].sval.type) {
                                *res_sval = cached_results[i].sval;
                                return true;
                if (expr == cached_results[i].expr)
                        return rl_to_sval(cached_results[i].rl, sval);
                        }
                        return false;
                }
        }

        orig_custom_fn = custom_handle_variable;
        custom_handle_variable = NULL;
        get_rl_sval(expr, RL_EXACT, &recurse_cnt, NULL, &sval);

        rl = _get_rl(expr, RL_EXACT, &recurse_cnt);
        if (!rl_to_sval(rl, &tmp))
                rl = NULL;
        custom_handle_variable = orig_custom_fn;

        cached_results[cache_idx].expr = expr;
        cached_results[cache_idx].sval = sval;
        cached_results[cache_idx].rl = rl;
        cache_idx = (cache_idx + 1) % ARRAY_SIZE(cached_results);

        if (!sval.type)
        if (!rl)
                return 0;
```



```
        *res_sval = sval;
        *sval = tmp;
        return 1;
}

static bool get_implied_value_internal(struct expression *expr, int *recurse_cnt
static int get_implied_value_internal(struct expression *expr, sval_t *sval, int
{
        struct range_list *rl;

        res_sval->type = NULL;

        if (!get_rl_sval(expr, RL_IMPLIED, recurse_cnt, &rl, res_sval))
                return false;
        if (!res_sval->type && !rl_to_sval(rl, res_sval))
                return false;
        return true;
        rl =  _get_rl(expr, RL_IMPLIED, recurse_cnt);
        if (!rl_to_sval(rl, sval))
                return 0;
        return 1;
}

int get_implied_value(struct expression *expr, sval_t *sval)
{
        struct range_list *rl;
        int recurse_cnt = 0;

        if (!get_rl_helper(expr, RL_IMPLIED, &rl) ||
            !rl_to_sval(rl, sval))
        rl =  _get_rl(expr, RL_IMPLIED, &recurse_cnt);
        if (!rl_to_sval(rl, sval))
                return 0;
        return 1;
}

int get_implied_min(struct expression *expr, sval_t *sval)
{
        struct range_list *rl;
        int recurse_cnt = 0;

        if (!get_rl_helper(expr, RL_IMPLIED, &rl) || !rl)
        rl =  _get_rl(expr, RL_IMPLIED, &recurse_cnt);
        if (!rl)
                return 0;
        *sval = rl_min(rl);
        return 1;
}

int get_implied_max(struct expression *expr, sval_t *sval)
{
        struct range_list *rl;
        int recurse_cnt = 0;

        if (!get_rl_helper(expr, RL_IMPLIED, &rl) || !rl)
        rl =  _get_rl(expr, RL_IMPLIED, &recurse_cnt);
        if (!rl)
                return 0;
        *sval = rl_max(rl);
        return 1;
}

int get_implied_rl(struct expression *expr, struct range_list **rl)
{
        if (!get_rl_helper(expr, RL_IMPLIED, rl) || !*rl)
```

```
1623                    return 0;
1313            int recurse_cnt = 0;

1315            *rl = _get_rl(expr, RL_IMPLIED, &recurse_cnt);
1316            if (*rl)
1624                    return 1;
1318            return 0;
1625 }

1627 static int get_absolute_rl_internal(struct expression *expr, struct range_list *
1628 {
1629            *rl = NULL;
1630            get_rl_internal(expr, RL_ABSOLUTE, recurse_cnt, rl);
1323            *rl = _get_rl(expr, RL_ABSOLUTE, recurse_cnt);
1631            if (!*rl)
1632                    *rl = alloc_whole_rl(get_type(expr));
1633            return 1;
1634 }

1636 int get_absolute_rl(struct expression *expr, struct range_list **rl)
1637 {
1638            *rl = NULL;
1639             get_rl_helper(expr, RL_ABSOLUTE, rl);
1331            int recurse_cnt = 0;

1333            *rl = _get_rl(expr, RL_ABSOLUTE, &recurse_cnt);
1640            if (!*rl)
1641                    *rl = alloc_whole_rl(get_type(expr));
1642            return 1;
1643 }

1645 int get_real_absolute_rl(struct expression *expr, struct range_list **rl)
1646 {
1647            *rl = NULL;
1648            get_rl_helper(expr, RL_REAL_ABSOLUTE, rl);
1341            int recurse_cnt = 0;

1343            *rl = _get_rl(expr, RL_REAL_ABSOLUTE, &recurse_cnt);
1649            if (!*rl)
1650                    *rl = alloc_whole_rl(get_type(expr));
1651            return 1;
1652 }

1654 int custom_get_absolute_rl(struct expression *expr,
1655                            struct range_list *(*fn)(struct expression *expr),
1656                            struct range_list **rl)
1657 {
1658            int ret;
1353            int recurse_cnt = 0;

1660            *rl = NULL;
1661            custom_handle_variable = fn;
1662            ret = get_rl_helper(expr, RL_REAL_ABSOLUTE, rl);
1357            *rl = _get_rl(expr, RL_REAL_ABSOLUTE, &recurse_cnt);
1663            custom_handle_variable = NULL;
1664            return ret;
1359            return 1;
1665 }
______unchanged_portion_omitted_

1678 int get_hard_max(struct expression *expr, sval_t *sval)
1679 {
1680            struct range_list *rl;
1376            int recurse_cnt = 0;

1682            if (!get_rl_helper(expr, RL_HARD, &rl) || !rl)
```



```
1378            rl =  _get_rl(expr, RL_HARD, &recurse_cnt);
1379            if (!rl)
1683                    return 0;
1684            *sval = rl_max(rl);
1685            return 1;
1686 }

1688 int get_fuzzy_min(struct expression *expr, sval_t *sval)
1689 {
1690            struct range_list *rl;
1691            sval_t tmp;
1389            int recurse_cnt = 0;

1693            if (!get_rl_helper(expr, RL_FUZZY, &rl) || !rl)
1391            rl =  _get_rl(expr, RL_FUZZY, &recurse_cnt);
1392            if (!rl)
1694                    return 0;
1695            tmp = rl_min(rl);
1696            if (sval_is_negative(tmp) && sval_is_min(tmp))
1697                    return 0;
1698            *sval = tmp;
1699            return 1;
1700 }

1702 int get_fuzzy_max(struct expression *expr, sval_t *sval)
1703 {
1704            struct range_list *rl;
1705            sval_t max;
1405            int recurse_cnt = 0;

1707            if (!get_rl_helper(expr, RL_FUZZY, &rl) || !rl)
1407            rl =  _get_rl(expr, RL_FUZZY, &recurse_cnt);
1408            if (!rl)
1708                    return 0;
1709            max = rl_max(rl);
1710            if (max.uvalue > INT_MAX - 10000)
1711                    return 0;
1712            *sval = max;
1713            return 1;
1714 }

1716 int get_absolute_min(struct expression *expr, sval_t *sval)
1717 {
1718            struct range_list *rl;
1719            struct symbol *type;
1421            int recurse_cnt = 0;

1721            type = get_type(expr);
1722            if (!type)
1723                    type = &llong_ctype;  // FIXME: this is wrong but places assume
1724            rl = NULL;
1725            get_rl_helper(expr, RL_REAL_ABSOLUTE, &rl);
1426            rl = _get_rl(expr, RL_REAL_ABSOLUTE, &recurse_cnt);
1726            if (rl)
1727                    *sval = rl_min(rl);
1728            else
1729                    *sval = sval_type_min(type);

1731            if (sval_cmp(*sval, sval_type_min(type)) < 0)
1732                    *sval = sval_type_min(type);
1733            return 1;
1734 }

1736 int get_absolute_max(struct expression *expr, sval_t *sval)
1737 {
1738            struct range_list *rl;
```

```
1739         struct symbol *type;
1441         int recurse_cnt = 0;

1741         type = get_type(expr);
1742         if (!type)
1743                 type = &llong_ctype;
1744         rl = NULL;
1745         get_rl_helper(expr, RL_REAL_ABSOLUTE, &rl);
1446         rl = _get_rl(expr, RL_REAL_ABSOLUTE, &recurse_cnt);
1746         if (rl)
1747                 *sval = rl_max(rl);
1748         else
1749                 *sval = sval_type_max(type);

1751         if (sval_cmp(sval_type_max(type), *sval) < 0)
1752                 *sval = sval_type_max(type);
1753         return 1;
1754 }
______unchanged_portion_omitted_

1555 int can_integer_overflow(struct symbol *type, struct expression *expr)
1556 {
1557         int op;
1558         sval_t lmax, rmax, res;

1560         if (!type)
1561                 type = &int_ctype;

1563         expr = strip_expr(expr);

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 }
```

****
1391 Mon Aug 5 08:38:37 2019
new/usr/src/tools/smatch/src/smatch_mem_tracker.c
11506 smatch resync
****

```
/*
 * 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;

unsigned long get_mem_kb(void)
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 0;
                return;
        fread(buf, 1, sizeof(buf), file);
        fclose(file);

        size = strtoul(buf, NULL, 10);
        size = size * sysconf(_SC_PAGESIZE) / 1024;
        return size;
}

static void match_end_func(struct symbol *sym)
{
        unsigned long size;

        if (option_mem) {
                size = get_mem_kb();
                if (size > max_size)
                        max_size = size;
        }
}
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    7239 Mon Aug  5 08:38:37 2019
new/usr/src/tools/smatch/src/smatch_modification_hooks.c
11506 smatch resync
**********************************************************
```

```
______unchanged_portion_omitted_

 43 static modification_hook **hooks;
 44 static modification_hook **hooks_late;

 46 ALLOCATOR(modification_data, "modification data");

 48 static int my_id;
 49 static struct smatch_state *alloc_my_state(struct expression *expr, struct smatc
 50 {
 51         struct smatch_state *state;
 52         struct modification_data *data;
 53         char *name;

 55         state = __alloc_smatch_state(0);
 55         expr = strip_expr(expr);
 56         name = expr_to_str(expr);
 57         if (!name)
 58                 return NULL;

 60         state = __alloc_smatch_state(0);
 61         state->name = alloc_sname(name);
 62         free_string(name);

 64         data = __alloc_modification_data(0);
 65         data->prev = prev;
 66         data->cur = expr;
 67         state->data = data;

 69         return state;
 70 }
______unchanged_portion_omitted_

157 static void db_param_add(struct expression *expr, int param, char *key, char *va
158 {
159         struct expression *arg, *gen_expr;
160         char *name, *other_name;
161         struct symbol *sym, *other_sym;

163         while (expr->type == EXPR_ASSIGNMENT)
164                 expr = strip_expr(expr->right);
165         if (expr->type != EXPR_CALL)
166                 return;

168         arg = get_argument_from_call_expr(expr->args, param);
169         if (!arg)
170                 return;

172         gen_expr = gen_expression_from_key(arg, key);
173         if (gen_expr)
174                 update_mtag_data(gen_expr);

176         name = get_variable_from_key(arg, key, &sym);
177         if (!name || !sym)
178                 goto free;

180         __in_fake_assign++;
181         call_modification_hooks_name_sym(name, sym, expr, BOTH);
182         __in_fake_assign--;

184         other_name = get_other_name_sym(name, sym, &other_sym);
```



```
181         other_name = map_long_to_short_name_sym(name, sym, &other_sym);
185         if (other_name) {
186                 __in_fake_assign++;
187                 call_modification_hooks_name_sym(other_name, other_sym, expr, BO
188                 __in_fake_assign--;
189                 free_string(other_name);
190         }

192 free:
193         free_string(name);
194 }
______unchanged_portion_omitted_

281 void register_modification_hooks(int id)
282 {
283         my_id = id;

285         set_dynamic_states(my_id);

287         hooks = malloc((num_checks + 1) * sizeof(*hooks));
288         memset(hooks, 0, (num_checks + 1) * sizeof(*hooks));
289         hooks_late = malloc((num_checks + 1) * sizeof(*hooks));
290         memset(hooks_late, 0, (num_checks + 1) * sizeof(*hooks));

292         add_hook(&match_assign_early, ASSIGNMENT_HOOK);
293         add_hook(&unop_expr_early, OP_HOOK);
294         add_hook(&asm_expr_early, ASM_HOOK);
295 }
______unchanged_portion_omitted_
```

```
**********************************************************
   11413 Mon Aug  5 08:38:38 2019
new/usr/src/tools/smatch/src/smatch_mtag.c
11506 smatch resync
**********************************************************
```

```
/*
 * 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;
}

const struct {
        const char *name;
        int size_arg;
} allocator_info[] = {
        { "kmalloc", 0 },
        { "kzalloc", 0 },
        { "devm_kmalloc", 1},
        { "devm_kzalloc", 1},
};

static bool is_mtag_call(struct expression *expr)
static void alloc_assign(const char *fn, struct expression *expr, void *unused)
{
        struct expression *arg;
        int i;
        sval_t sval;

        if (expr->type != EXPR_CALL ||
            expr->fn->type != EXPR_SYMBOL ||
            !expr->fn->symbol)
                return false;

        for (i = 0; i < ARRAY_SIZE(allocator_info); i++) {
                if (strcmp(expr->fn->symbol->ident->name, allocator_info[i].name
                        break;
        }
        if (i == ARRAY_SIZE(allocator_info))
                return false;

        arg = get_argument_from_call_expr(expr->args, allocator_info[i].size_arg
        if (!get_implied_value(arg, &sval))
                return false;

        return true;
}

struct smatch_state *swap_mtag_return(struct expression *expr, struct smatch_sta
{
        struct expression *left, *right;
        char *left_name, *right_name;
        struct symbol *left_sym;
        struct range_list *rl;
        char buf[256];
```

```
113         mtag_t tag;
114         sval_t tag_sval;

116         if (!expr || expr->type != EXPR_ASSIGNMENT || expr->op != '=')
117                 return state;

119         if (!estate_rl(state) || strcmp(state->name, "0,4096-ptr_max") != 0)
120                 return state;
 94         // FIXME:  This should only happen when the size is not a paramter of
 95         // the caller
 96         return;

 98         if (expr->type != EXPR_ASSIGNMENT || expr->op != '=')
 99                 return;
122         left = strip_expr(expr->left);
123         right = strip_expr(expr->right);
102         if (right->type != EXPR_CALL || right->fn->type != EXPR_SYMBOL)
103                 return;

125         if (!is_mtag_call(right))
126                 return state;

128         left_name = expr_to_str_sym(left, &left_sym);
129         if (!left_name || !left_sym)
130                 return state;
131         right_name = expr_to_str(right);

133         snprintf(buf, sizeof(buf), "%s %s %s %s", get_filename(), get_function()
134                  left_name, right_name);
135         tag = str_to_tag(buf);
136         tag_sval.type = estate_type(state);
137         tag_sval.uvalue = tag;

139         rl = rl_filter(estate_rl(state), valid_ptr_rl);
140         rl = clone_rl(rl);
141         add_range(&rl, tag_sval, tag_sval);
112         sql_insert_mtag_about(tag, left_name, right_name);

143         sql_insert_mtag_about(tag, left_name, buf);
114         if (left_name && left_sym)
115                 set_state(my_id, left_name, left_sym, alloc_tag_state(tag));

145         free_string(left_name);
146         free_string(right_name);

148         return alloc_estate_rl(rl);
149 }
```

_____***unchanged_portion_omitted_***

```
184 bool get_symbol_mtag(struct symbol *sym, mtag_t *tag)
154 int get_deref_mtag(struct expression *expr, mtag_t *tag)
185 {
186         char buf[256];
156         mtag_t container_tag, member_tag;
157         int offset;

188         if (!sym || !sym->ident)
189                 return false;
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          */
```



```
191         if (get_toplevel_mtag(sym, tag))
192                 return true;
167         if (!get_mtag(expr->unop, &container_tag))
168                 return 0;

194         if (get_param_num_from_sym(sym) >= 0)
195                 return false;
170         offset = get_member_offset_from_deref(expr);
171         if (offset < 0)
172                 return 0;

197         snprintf(buf, sizeof(buf), "%s %s %s",
198                  get_filename(), get_function(), sym->ident->name);
199         *tag = str_to_tag(buf);
200         return true;
174         if (!mtag_map_select_tag(container_tag, -offset, &member_tag))
175                 return 0;

177         *tag = member_tag;
178         return 1;
201 }
```

_____***unchanged_portion_omitted_***

```
193 static void db_returns_buf_size(struct expression *expr, int param, char *unused
194 {
195         struct expression *call;
196         struct range_list *rl;

198         if (expr->type != EXPR_ASSIGNMENT)
199                 return;
200         call = strip_expr(expr->right);

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 }

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;

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;

221         tag = strtoul(value, NULL, 10);

223         if (!create_mtag_alias(tag, call, &alias))
224                 return;

226         arg = get_argument_from_call_expr(call->args, param);
227         if (!arg)
228                 return;

230         name = get_variable_from_key(arg, key, &sym);
231         if (!name || !sym)
232                 goto free;

234         set_state(my_id, name, sym, alloc_tag_state(alias));
235 free:
```

```
236         free_string(name);
237 }

239 static void match_call_info(struct expression *expr)
240 {
241         struct smatch_state *state;
242         struct expression *arg;
243         int i = -1;

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 }

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 }

215 static int get_array_mtag_offset(struct expression *expr, mtag_t *tag, int *offs
216 {
217         struct expression *array, *offset_expr;
218         struct symbol *type;
219         sval_t sval;
220         int start_offset;

222         if (!is_array(expr))
223                 return 0;

225         array = get_array_base(expr);
226         if (!array)
227                 return 0;
228         type = get_type(array);
229         if (!type || type->type != SYM_ARRAY)
230                 return 0;
231         type = get_real_base_type(type);
232         if (!type_bytes(type))
233                 return 0;

235         if (!expr_to_mtag_offset(array, tag, &start_offset))
288         if (!get_mtag(array, tag))
236                 return 0;

238         offset_expr = get_array_offset(expr);
239         if (!get_value(offset_expr, &sval))
240                 return 0;
241         *offset = start_offset + sval.value * type_bytes(type);
294         *offset = sval.value * type_bytes(type);

243         return 1;
244 }
```



```
246 struct range_list *swap_mtag_seed(struct expression *expr, struct range_list *rl
299 static int get_implied_mtag_offset(struct expression *expr, mtag_t *tag, int *of
247 {
248         char buf[256];
249         char *name;
301         struct smatch_state *state;
302         struct symbol *type;
250         sval_t sval;
251         mtag_t tag;

253         if (!rl_to_sval(rl, &sval))
254                 return rl;
255         if (sval.type->type != SYM_PTR || sval.uvalue != MTAG_SEED)
256                 return rl;
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;

258         name = expr_to_str(expr);
259         snprintf(buf, sizeof(buf), "%s %s %s", get_filename(), get_function(), n
260         free_string(name);
261         tag = str_to_tag(buf);
262         sval.value = tag;
263         return alloc_rl(sval, sval);
312         *tag = sval.uvalue & ~MTAG_OFFSET_MASK;
313         *offset = sval.uvalue & MTAG_OFFSET_MASK;
314         return 1;
264 }

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 }

266 int create_mtag_alias(mtag_t tag, struct expression *expr, mtag_t *new)
267 {
268         char buf[256];
269         int lines_from_start;
270         char *str;

272         /*
273          * We need the alias to be unique.  It's not totally required that it
274          * be the same from one DB build to then next, but it makes debugging
275          * a bit simpler.
276          *
277          */

279         if (!cur_func_sym)
280                 return 0;

282         lines_from_start = expr->pos.line - cur_func_sym->pos.line;
283         str = expr_to_str(expr);
284         snprintf(buf, sizeof(buf), "%lld %d %s", tag, lines_from_start, str);
285         free_string(str);

287         *new = str_to_tag(buf);
288         sql_insert_mtag_alias(tag, *new);

290         return 1;
291 }
```



```
293 static int get_implied_mtag_offset(struct expression *expr, mtag_t *tag, int *of
294 {
295         struct smatch_state *state;
296         struct symbol *type;
297         sval_t sval;

299         type = get_type(expr);
300         if (!type_is_ptr(type))
301                 return 0;
302         state = get_extra_state(expr);
303         if (!state || !estate_get_single_value(state, &sval) || sval.value == 0)
304                 return 0;

306         *tag = sval.uvalue & ~MTAG_OFFSET_MASK;
307         *offset = sval.uvalue & MTAG_OFFSET_MASK;
308         return 1;
309 }

311 /*
312  * The point of this function is to give you the mtag and the offset so
313  * you can look up the data in the DB.  It takes an expression.
314  *
315  * So say you give it "foo->bar".  Then it would give you the offset of "bar"
316  * and the implied value of "foo".  Or if you lookup "*foo" then the offset is
317  * zero and we look up the implied value of "foo.  But if the expression is
318  * foo, then if "foo" is a global variable, then we get the mtag and the offset
319  * is zero.  If "foo" is a local variable, then there is nothing to look up in
320  * the mtag_data table because that's handled by smatch_extra.c to this returns
321  * false.
322  *
323  */
324 int expr_to_mtag_offset(struct expression *expr, mtag_t *tag, int *offset)
325 {
326         *tag = 0;
327         *offset = 0;

329         if (bits_in_pointer != 64)
330                 return 0;

332         expr = strip_expr(expr);
333         if (!expr)
334                 return 0;

336         if (is_array(expr))
337                 return get_array_mtag_offset(expr, tag, offset);

339         if (expr->type == EXPR_PREOP && expr->op == '*') {
340                 expr = strip_expr(expr->unop);
341                 return get_implied_mtag_offset(expr, tag, offset);
342         } else if (expr->type == EXPR_DEREF) {
343                 int tmp, tmp_offset = 0;

345                 while (expr->type == EXPR_DEREF) {
346                         tmp = get_member_offset_from_deref(expr);
347                         if (tmp < 0)
429         if (expr->type ==  EXPR_DEREF) {
430                 *offset = get_member_offset_from_deref(expr);
431                 if (*offset < 0)
348                                 return 0;
349                         tmp_offset += tmp;
350                         expr = expr->deref;
433                 return get_mtag(expr->deref, tag);
351                 }
352                 *offset = tmp_offset;
353                 if (expr->type == EXPR_PREOP && expr->op == '*') {
```

```
354                         expr = strip_expr(expr->unop);

356                         if (get_implied_mtag_offset(expr, tag, &tmp_offset)) {
357                                 // FIXME:  look it up recursively?
358                                 if (tmp_offset)
359                                         return 0;
436         if (get_implied_mtag_offset(expr, tag, offset))
360                                 return 1;
361                         }
362                         return 0;
363                 } else if (expr->type == EXPR_SYMBOL) {
364                         return get_symbol_mtag(expr->symbol, tag);
365                 }
366                 return 0;
367         } else if (expr->type == EXPR_SYMBOL) {
368                 return get_symbol_mtag(expr->symbol, tag);
369         }
370         return 0;

439         return get_mtag(expr, tag);
371 }

373 /*
374  * This function takes an address and returns an sval.  Let's take some
375  * example things you might pass to it:
376  * foo->bar:
377  *   If we were only called from smatch_math, we wouldn't need to bother with
378  *   this because it's already been looked up in smatch_extra.c but this is
379  *   also called from other places so we have to check smatch_extra.c.
380  * &foo
381  *   If "foo" is global return the mtag for "foo".
382  * &foo.bar
383  *   If "foo" is global return the mtag for "foo" + the offset of ".bar".
384  * It also handles string literals.
385  *
386  */
387 int get_mtag_sval(struct expression *expr, sval_t *sval)
388 {
389         struct symbol *type;
390         mtag_t tag;
391         int offset = 0;

393         if (bits_in_pointer != 64)
394                 return 0;

396         expr = strip_expr(expr);

398         type = get_type(expr);
399         if (!type_is_ptr(type))
400                 return 0;
401         /*
402          * There are several options:
457          * There are only three options:
403          *
404          * If the expr is a string literal, that's an address/mtag.
405          * SYM_ARRAY and SYM_FN are mtags.  There are "&foo" type addresses.
406          * And there are saved pointers "p = &foo;"
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;
407          *
408          */
```



```
410         if (expr->type == EXPR_STRING && get_string_mtag(expr, &tag))
411                 goto found;

413         if (expr->type == EXPR_SYMBOL &&
414             (type->type == SYM_ARRAY || type->type == SYM_FN) &&
415             get_toplevel_mtag(expr->symbol, &tag))
471         if (type->type == SYM_ARRAY && get_toplevel_mtag(expr->symbol, &tag))
416                 goto found;

418         if (expr->type == EXPR_PREOP && expr->op == '&') {
419                 expr = strip_expr(expr->unop);
420                 if (expr_to_mtag_offset(expr, &tag, &offset))
421                         goto found;
422                 return 0;
423         }

425         if (get_implied_mtag_offset(expr, &tag, &offset))
426                 goto found;

477         if (expr->type != EXPR_PREOP || expr->op != '&')
428         return 0;
429 found:
430         if (offset >= MTAG_OFFSET_MASK)
479         expr = strip_expr(expr->unop);

481         if (!expr_to_mtag_offset(expr, &tag, &offset))
431                 return 0;
483         if (offset > MTAG_OFFSET_MASK)
484                 offset = MTAG_OFFSET_MASK;

486 found:
433         sval->type = type;
434         sval->uvalue = tag | offset;

436         return 1;
437 }

493 static struct expression *remove_dereference(struct expression *expr)
494 {
495         expr = strip_expr(expr);

497         if (expr->type == EXPR_PREOP && expr->op == '*')
498                 return strip_expr(expr->unop);
499         return preop_expression(expr, '&');
500 }

502 int get_mtag_addr_sval(struct expression *expr, sval_t *sval)
503 {
504         return get_mtag_sval(remove_dereference(expr), sval);
505 }

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;

514         FOR_EACH_MY_SM(my_id, __get_cur_stree(), sm) {
515                 if (!sm->state->data)
516                         continue;

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 }
```

```
439 void register_mtag(int id)
440 {
441         my_id = id;


444         /*
445          * The mtag stuff only works on 64 systems because we store the
446          * information in the pointer itself.
447          * bit 63   : set for alias mtags
448          * bit 62-12: mtag hash
449          * bit 11-0 : offset
450          *
451          */
```

```
545         if (bits_in_pointer != 64)
546                 return;
```

```
453         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);
454 }
```

_____unchanged_portion_omitted_

```
**********************************************************
    5770 Mon Aug  5 08:38:38 2019
new/usr/src/tools/smatch/src/smatch_mtag_data.c
11506 smatch resync
**********************************************************
______unchanged_portion_omitted_

 39 static struct range_list *select_orig(mtag_t tag, int offset)
 39 static struct range_list *select_orig_rl(sval_t sval)
 40 {
 41         struct range_list *rl = NULL;
 42         mtag_t tag = sval.uvalue & ~MTAG_OFFSET_MASK;
 43         int offset = sval.uvalue & MTAG_OFFSET_MASK;

 43         mem_sql(&save_rl, &rl, "select value from mtag_data where tag = %lld and
 44                 tag, offset);
 45         return rl;
 46 }
______unchanged_portion_omitted_

 72 static void insert_mtag_data(mtag_t tag, int offset, struct range_list *rl)
 74 void insert_mtag_data(sval_t sval, struct range_list *rl)
 73 {
 76         mtag_t tag = sval.uvalue & ~MTAG_OFFSET_MASK;
 77         int offset = sval.uvalue & MTAG_OFFSET_MASK;

 74         rl = clone_rl_permanent(rl);

 76         mem_sql(NULL, NULL, "delete from mtag_data where tag = %lld and offset =
 77                 tag, offset, DATA_VALUE);
 78         mem_sql(NULL, NULL, "insert into mtag_data values (%lld, %d, %d, '%lu');
 79                 tag, offset, DATA_VALUE, (unsigned long)rl);
 80 }

 82 void update_mtag_data(struct expression *expr)
 83 {
 84         struct range_list *orig, *new, *rl;
 85         struct symbol *type;
 86         char *name;
 87         mtag_t tag;
 88         int offset;
 91         sval_t sval;

 90         name = expr_to_var(expr);
 91         if (is_kernel_param(name)) {
 92                 free_string(name);
 93                 return;
 94         }
 95         free_string(name);

 97         if (!expr_to_mtag_offset(expr, &tag, &offset))
100         if (!get_mtag_addr_sval(expr, &sval))
 98                 return;

100         type = get_type(expr);
101         if ((offset == 0) &&
102             (!type || type == &void_ctype ||
103              type->type == SYM_STRUCT || type->type == SYM_UNION || type->type =
104                 return;

106         get_absolute_rl(expr, &rl);

108         orig = select_orig(tag, offset);
105         orig = select_orig_rl(sval);
109         new = rl_union(orig, rl);
110         insert_mtag_data(tag, offset, new);
```



```
107         insert_mtag_data(sval, new);
111 }

113 static void match_global_assign(struct expression *expr)
114 {
115         struct range_list *rl;
116         mtag_t tag;
117         int offset;
113         sval_t sval;
118         char *name;

120         name = expr_to_var(expr->left);
121         if (is_kernel_param(name)) {
122                 free_string(name);
123                 return;
124         }
125         free_string(name);

127         if (!expr_to_mtag_offset(expr->left, &tag, &offset))
123         if (!get_mtag_addr_sval(expr->left, &sval))
128                 return;

130         get_absolute_rl(expr->right, &rl);
131         insert_mtag_data(tag, offset, rl);
127         insert_mtag_data(sval, rl);
132 }
______unchanged_portion_omitted_

177 struct db_cache_results {
178         mtag_t tag;
174         sval_t sval;
179         struct range_list *rl;
180 };
181 static struct db_cache_results cached_results[8];

183 static int get_rl_from_mtag_offset(mtag_t tag, int offset, struct symbol *type,
179 static int get_rl_from_mtag_sval(sval_t sval, struct symbol *type, struct range_
184 {
185         struct db_info db_info = {};
186         mtag_t merged = tag | offset;
182         mtag_t tag;
183         int offset;
187         static int idx;
188         int ret;
189         int i;

191         if (!type || type == &void_ctype ||
192             (type->type == SYM_STRUCT || type->type == SYM_ARRAY || type->type =
193                 return 0;

195         for (i = 0; i < ARRAY_SIZE(cached_results); i++) {
196                 if (merged == cached_results[i].tag) {
189                 if (sval.uvalue == cached_results[i].sval.uvalue) {
197                         if (cached_results[i].rl) {
198                                 *rl = cached_results[i].rl;
199                                 return 1;
200                         }
201                         return 0;
202                 }
203         }

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;
```

```
        }
        db_info.type = type;

        run_sql(get_vals, &db_info,
                "select value from mtag_data where tag = %lld and offset = %d an
                tag, offset, DATA_VALUE);
        if (!db_info.rl || is_whole_rl(db_info.rl)) {
                db_info.rl = NULL;
                ret = 0;
                goto update_cache;
        }

        *rl = db_info.rl;
        ret = 1;

update_cache:
        cached_results[idx].tag = merged;
        cached_results[idx].sval = sval;
        cached_results[idx].rl = db_info.rl;
        idx = (idx + 1) % ARRAY_SIZE(cached_results);

        return ret;
}
```

*unchanged_portion_omitted_*

```
int get_mtag_rl(struct expression *expr, struct range_list **rl)
{
        struct symbol *type;
        mtag_t tag;
        int offset;
        sval_t sval;

        if (!expr_to_mtag_offset(expr, &tag, &offset))
        if (!get_mtag_addr_sval(expr, &sval))
                return 0;
        if (offset >= MTAG_OFFSET_MASK)
                return 0;

        type = get_type(expr);
        if (!type)
                return 0;

        return get_rl_from_mtag_offset(tag, offset, type, rl);
        return get_rl_from_mtag_sval(sval, type, rl);
}
```

*unchanged_portion_omitted_*

```
**********************************************************
    1625 Mon Aug  5 08:38:39 2019
new/usr/src/tools/smatch/src/smatch_mtag_map.c
11506 smatch resync
**********************************************************
```

```
 1 /*
 2  * Copyright (C) 2017 Oracle.  All rights reserved.
 3  *
 4  * This program is free software; you can redistribute it and/or
 5  * modify it under the terms of the GNU General Public License
 6  * as published by the Free Software Foundation; either version 2
 7  * of the License, or (at your option) any later version.
 8  *
 9  * This program is distributed in the hope that it will be useful,
10  * but WITHOUT ANY WARRANTY; without even the implied warranty of
11  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
12  * GNU General Public License for more details.
13  *
14  * You should have received a copy of the GNU General Public License
15  * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
16  */

18 /*
19  * This basically stores when a pointer is stored as a struct member.
20  *
21  */

23 #include "smatch.h"
24 #include "smatch_slist.h"
25 #include "smatch_extra.h"

27 static int my_id;

29 static void match_assign(struct expression *expr)
30 {
31         struct expression *left, *right;
32         mtag_t left_tag;
32         mtag_t left_tag, right_tag;
33         int offset;
34         sval_t sval;

36         if (expr->op != '=')
37                 return;

39         left = strip_expr(expr->left);
40         right = strip_expr(expr->right);

42         if (!type_is_ptr(get_type(right)))
41         if (left->type != EXPR_DEREF)
43                 return;
44         if (!get_implied_value(right, &sval))

44         offset = get_member_offset_from_deref(left);
45         if (offset < 0)
45                 return;
46         if (sval_cmp(sval, valid_ptr_min_sval) < 0 ||
47             sval_cmp(sval, valid_ptr_max_sval) > 0)

48         if (!get_mtag(left->deref, &left_tag))
48                 return;
49         if (sval.uvalue & MTAG_OFFSET_MASK)
50         if (!get_mtag(right, &right_tag))
50                 return;

52         if (!expr_to_mtag_offset(left, &left_tag, &offset))
53                 return;
```



```
55         sql_insert_mtag_map(sval.uvalue, -offset, left_tag);
53         sql_insert_mtag_map(right_tag, -offset, left_tag);
56 }
______unchanged_portion_omitted_
```

****
6995 Mon Aug 5 08:38:39 2019
new/usr/src/tools/smatch/src/smatch_nul_terminator.c
11506 smatch resync
****
_____unchanged_portion_omitted_

```
250 static void match_strnlen_test(struct expression *expr)
251 {
252         struct expression *left, *tmp, *arg;
253         int cnt;

255         if (expr->type != EXPR_COMPARE)
256                 return;
257         if (expr->op != SPECIAL_EQUAL && expr->op != SPECIAL_NOTEQUAL)
258                 return;

260         left = strip_expr(expr->left);
261         cnt = 0;
262         while ((tmp = get_assigned_expr(left))) {
263                 if (cnt++ > 3)
264                         break;
265                 left = tmp;
266         }

268         if (left->type != EXPR_CALL)
269                 return;
270         if (!sym_name_is("strnlen", left->fn))
271                 return;
272         arg = get_argument_from_call_expr(left->args, 0);
273         set_true_false_states_expr(my_id, arg,
274                         (expr->op == SPECIAL_EQUAL) ? &terminated : NULL,
275                         (expr->op == SPECIAL_NOTEQUAL) ? &terminated : NULL);
276         if (get_param_num(arg) >= 0)
277                 set_true_false_states_expr(param_set_id, arg,
278                                 (expr->op == SPECIAL_EQUAL) ? &terminated : NULL
279                                 (expr->op == SPECIAL_NOTEQUAL) ? &terminated : N
280 }

282 void register_nul_terminator(int id)
283 {
284         my_id = id;

286         add_hook(&match_nul_assign, ASSIGNMENT_HOOK);
287         add_hook(&match_string_assign, ASSIGNMENT_HOOK);

289         add_hook(&match_call_info, FUNCTION_CALL_HOOK);
290         add_member_info_callback(my_id, struct_member_callback);
291         add_split_return_callback(&split_return_info);

293         select_caller_info_hook(caller_info_terminated, TERMINATED);
294         select_return_states_hook(TERMINATED, return_info_terminated);

296         add_hook(&match_strnlen_test, CONDITION_HOOK);
297 }
```

_____unchanged_portion_omitted_

```
**********************************************************
    9433 Mon Aug  5 08:38:39 2019
new/usr/src/tools/smatch/src/smatch_param_compare_limit.c
11506 smatch resync
**********************************************************
```

_____*unchanged_portion_omitted_*

```
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[248];
184         static char right_buf[248];
183         static char left_buf[256];
184         static char right_buf[256];
185         static char info_buf[256];
186         const char *tmp_name;

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;

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;

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);

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

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);

233         } END_FOR_EACH_SM(tmp);
234 }
```

_____*unchanged_portion_omitted_*

```
279 static int split_op_param_key(char *value, int *op, int *param, char **key)
280 {
281         static char buf[256];
282         char *p;

284         if (!parse_comparison(&value, op))
285                 return 0;

287         snprintf(buf, sizeof(buf), "%s", value);
287         snprintf(buf, sizeof(buf), value);

289         p = buf;
290         if (*p++ != '$')
291                 return 0;

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;

303         return 1;
304 }
```

_____*unchanged_portion_omitted_*

```
356 void register_param_compare_limit(int id)
357 {
358         compare_id = id;

360         set_dynamic_states(compare_id);
361         add_merge_hook(compare_id, &merge_compare_states);
362         add_split_return_callback(&print_return_comparison);

364         select_return_states_hook(COMPARE_LIMIT, &db_return_comparison);
365 }

367 void register_param_compare_limit_links(int id)
368 {
369         link_id = id;

371         set_dynamic_states(link_id);
372         add_merge_hook(link_id, &merge_links);

373 }
```

_____*unchanged_portion_omitted_*

****
5256 Mon Aug 5 08:38:40 2019
new/usr/src/tools/smatch/src/smatch_param_filter.c
11506 smatch resync
****
_____***unchanged_portion_omitted_***

```
 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         if (is_ignored_kernel_data(param_name)) {
 152                 insert_string(totally_filtered, (char *)sm->name);
 153                 return;
 154         }

 156         sql_insert_return_states(return_id, return_ranges, PARAM_FILTER, param,
 157                         param_name, show_rl(estate_rl(sm->state)));
 158 }
```

_____***unchanged_portion_omitted_***

```
 202 void register_param_filter(int id)
 203 {
 204         my_id = id;

 206         set_dynamic_states(my_id);
 207         add_hook(&save_start_states, AFTER_DEF_HOOK);
 208         add_hook(&free_start_states, AFTER_FUNC_HOOK);

 210         add_extra_mod_hook(&extra_mod_hook);
 211         add_unmatched_state_hook(my_id, &unmatched_state);
 212         add_pre_merge_hook(my_id, &pre_merge_hook);
 213         add_merge_hook(my_id, &merge_estates);

 215         add_hook(&match_save_states, INLINE_FN_START);
 216         add_hook(&match_restore_states, INLINE_FN_END);

 218         add_split_return_callback(&print_return_value_param);
 219 }
```

_____***unchanged_portion_omitted_***

****
5733 Mon Aug 5 08:38:40 2019
new/usr/src/tools/smatch/src/smatch_param_limit.c
11506 smatch resync
****
_____***unchanged_portion_omitted_***

```
 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;

 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;

 147                 param_name = get_param_name(tmp);
 148                 if (!param_name)
 149                         continue;

 151                 state = __get_state(my_id, tmp->name, tmp->sym);
 152                 if (!state)
 153                         state = tmp->state;

 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;

 161                 if (is_ignored_kernel_data(param_name))
 162                         continue;

 164                 rl = generify_mtag_range(state);
 165                 sql_insert_return_states(return_id, return_ranges, PARAM_LIMIT,
 166                                          param, param_name, show_rl(rl));
 167         } END_FOR_EACH_SM(tmp);
 168 }
```
_____***unchanged_portion_omitted_***

```
 195 void register_param_limit(int id)
 196 {
 197         my_id = id;

 199         set_dynamic_states(my_id);
 200         add_hook(&save_start_states, AFTER_DEF_HOOK);
 201         add_hook(&free_start_states, AFTER_FUNC_HOOK);

 203         add_extra_mod_hook(&extra_mod_hook);
 204         add_unmatched_state_hook(my_id, &unmatched_state);
 205         add_merge_hook(my_id, &merge_estates);

 207         add_hook(&match_save_states, INLINE_FN_START);
 208         add_hook(&match_restore_states, INLINE_FN_END);

 210         add_split_return_callback(&print_return_value_param);
 211 }
```
_____***unchanged_portion_omitted_***

```
**********************************************************
    6249 Mon Aug  5 08:38:40 2019
new/usr/src/tools/smatch/src/smatch_param_set.c
11506 smatch resync
**********************************************************
```

```
______unchanged_portion_omitted_

 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;

 168         FOR_EACH_MY_SM(my_id, __get_cur_stree(), sm) {
 169                 if (!estate_rl(sm->state))
 170                         continue;
 171                 extra = get_state(SMATCH_EXTRA, sm->name, sm->sym);
 172                 if (extra) {
 173                         rl = rl_intersection(estate_rl(sm->state), estate_rl(ext
 174                         if (!rl)
 175                                 continue;
 176                 } else {
 177                         rl = estate_rl(sm->state);
 178                 }

 180                 param = get_param_num_from_sym(sm->sym);
 181                 if (param < 0)
 182                         continue;
 183                 param_name = get_param_name(sm);
 184                 if (!param_name)
 185                         continue;
 186                 if (strcmp(param_name, "$") == 0) {
 187                         insert_string(&set_list, (char *)sm->name);
 188                         continue;
 189                 }
 190                 if (is_recursive_member(param_name)) {
 191                         insert_string(&set_list, (char *)sm->name);
 192                         continue;
 193                 }

 195                 if (is_ignored_kernel_data(param_name)) {
 192                 if (rl_to_sval(rl, &sval)) {
 196                         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 }
______unchanged_portion_omitted_

 259 void register_param_set(int id)
 260 {
 261         my_id = id;

 263         set_dynamic_states(my_id);
 264         add_extra_mod_hook(&extra_mod_hook);
 265         add_hook(match_array_assignment, ASSIGNMENT_HOOK);
 266         add_unmatched_state_hook(my_id, &unmatched_state);
 267         add_merge_hook(my_id, &merge_estates);
 268         add_split_return_callback(&print_return_value_param);
 269 }
______unchanged_portion_omitted_
```

```
**********************************************************
    5853 Mon Aug  5 08:38:41 2019
new/usr/src/tools/smatch/src/smatch_param_to_mtag_data.c
11506 smatch resync
**********************************************************
______unchanged_portion_omitted_
```

```
 79 static bool is_local_var(struct expression *expr)
 80 {
 81         struct symbol *sym;

 83         if (!expr || expr->type != EXPR_SYMBOL)
 84                 return false;
 85         sym = expr->symbol;
 86         if (!(sym->ctype.modifiers & MOD_TOPLEVEL))
 87                 return true;
 88         return false;
 89 }

 91 static void match_assign(struct expression *expr)
 92 {
 93         struct expression *left;
 94         struct symbol *right_sym;
 95         char *name;
 96         mtag_t tag;
 97         int offset;
 98         int param;

100         if (expr->op != '=')
101                 return;
102         left = strip_expr(expr->left);
103         if (is_local_var(left))
104                 return;
105         right_sym = expr_to_sym(expr->right);
106         if (!right_sym)
107                 return;

109         param = get_param_num_from_sym(right_sym);
110         if (param < 0)
111                 return;
112         // FIXME:  modify param_has_filter_data() to take a name/sym
113         if (!expr_to_mtag_offset(left, &tag, &offset))
114                 return;
115         name = expr_to_str(left);
116         if (!name)
117                 return;
118         set_state_expr(my_id, expr->right, alloc_tag_data_state(tag, name, offse
119         free_string(name);
120 }

108 #if 0
109 static void save_mtag_to_map(struct expression *expr, mtag_t tag, int offset, in
110 {
111         struct expression *arg, *gen_expr;
112         mtag_t arg_tag;

114         arg = get_argument_from_call_expr(expr->args, param);
115         if (!arg)
116                 return;

118         gen_expr = gen_expression_from_key(arg, key);
119         if (!gen_expr)
120                 return;

122         if (!get_mtag(gen_expr, &arg_tag))
123                 arg_tag = 0;
```



```
125         if (local_debug)
126                 sm_msg("finding mtag for '%s' %lld", expr_to_str(gen_expr), arg_
127 }
128 #endif

122 static void propogate_assignment(struct expression *expr, mtag_t tag, int offset
123 {
124         struct expression *arg;
125         int orig_param;
126         char buf[32];
127         char *name;
128         struct symbol *sym;

130         arg = get_argument_from_call_expr(expr->args, param);
131         if (!arg)
132                 return;
133         name = get_variable_from_key(arg, key, &sym);
134         if (!name || !sym)
135                 goto free;

137         orig_param = get_param_num_from_sym(sym);
138         if (orig_param < 0)
139                 goto free;

141         snprintf(buf, sizeof(buf), "$->[%d]", offset);
142         set_state(my_id, name, sym, alloc_tag_data_state(tag, buf, offset));
143 free:
144         free_string(name);
145 }

147 static void assign_to_alias(struct expression *expr, int param, mtag_t tag, int
148 {
149         struct expression *arg, *gen_expr;
150         struct range_list *rl;
151         mtag_t arg_tag;
152         mtag_t alias;
153         int arg_offset;

155         arg = get_argument_from_call_expr(expr->args, param);
156         if (!arg)
157                 return;

159         gen_expr = gen_expression_from_key(arg, key);
160         if (!gen_expr)
161                 return;

163         get_absolute_rl(gen_expr, &rl);

165         if (!create_mtag_alias(tag, expr, &alias))
166                 return;

168 //      insert_mtag_data(alias, offset, rl);

170         // FIXME:  is arg_offset handled correctly?
171         if (expr_to_mtag_offset(gen_expr, &arg_tag, &arg_offset) && arg_offset =
177         if (get_mtag(gen_expr, &arg_tag))
172                 sql_insert_mtag_map(arg_tag, -offset, alias);
173 }
```

______unchanged_portion_omitted_

```
222 void register_param_to_mtag_data(int id)
223 {
224         my_id = id;

226         set_dynamic_states(my_id);
```

```
 227         add_hook(&match_assign, ASSIGNMENT_HOOK);
 228         select_return_states_hook(MTAG_ASSIGN, &call_does_mtag_assign);
 229         add_merge_hook(my_id, &merge_tag_info);
 230         add_split_return_callback(&print_stored_to_mtag);
 231 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    2652 Mon Aug  5 08:38:41 2019
new/usr/src/tools/smatch/src/smatch_param_used.c
11506 smatch resync
**********************************************************
```

```
   1 /*
   2  * Copyright (C) 2015 Oracle.
   3  *
   4  * This program is free software; you can redistribute it and/or
   5  * modify it under the terms of the GNU General Public License
   6  * as published by the Free Software Foundation; either version 2
   7  * of the License, or (at your option) any later version.
   8  *
   9  * This program is distributed in the hope that it will be useful,
  10  * but WITHOUT ANY WARRANTY; without even the implied warranty of
  11  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  12  * GNU General Public License for more details.
  13  *
  14  * You should have received a copy of the GNU General Public License
  15  * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
  16  */

  18 #include "smatch.h"
  19 #include "smatch_slist.h"

  21 static int my_id;

  23 static struct stree *used_stree;
  24 static struct stree_stack *saved_stack;

  26 STATE(used);

  28 static void get_state_hook(int owner, const char *name, struct symbol *sym)
  29 {
  30         int arg;

  32         if (!option_info)
  33                 return;
  34         if (__in_fake_assign || __in_fake_parameter_assign || __in_function_def)
  34         if (__in_fake_assign)
  35                 return;

  37         arg = get_param_num_from_sym(sym);
  38         if (arg >= 0)
  39                 set_state_stree(&used_stree, my_id, name, sym, &used);
  40 }

  42 static void set_param_used(struct expression *call, struct expression *arg, char
  43 {
  44         struct symbol *sym;
  45         char *name;
  46         int arg_nr;

  48         name = get_variable_from_key(arg, key, &sym);
  49         if (!name || !sym)
  50                 goto free;

  52         arg_nr = get_param_num_from_sym(sym);
  53         if (arg_nr >= 0)
  54                 set_state_stree(&used_stree, my_id, name, sym, &used);
  54                 set_state(my_id, name, sym, &used);
  55 free:
  56         free_string(name);
  57 }

  59 static void process_states(void)
```



```
  60 {
  61         struct sm_state *tmp;
  62         int arg;
  63         const char *name;

  65         FOR_EACH_SM(used_stree, tmp) {
  66                 arg = get_param_num_from_sym(tmp->sym);
  67                 if (arg < 0)
  68                         continue;
  69                 name = get_param_name(tmp);
  70                 if (!name)
  71                         continue;
  72                 if (is_recursive_member(name))
  73                         continue;

  75                 if (is_ignored_kernel_data(name))
  76                         continue;

  78                 sql_insert_return_implies(PARAM_USED, arg, name, "");
  79         } END_FOR_EACH_SM(tmp);

  81         free_stree(&used_stree);
  82 }
```

_____*unchanged_portion_omitted_*

```
**********************************************************
   13407 Mon Aug  5 08:38:41 2019
new/usr/src/tools/smatch/src/smatch_parse_call_math.c
11506 smatch resync
**********************************************************
______unchanged_portion_omitted_
```

```
117 static int read_rl_from_var(struct expression *call, const char *p, const char *
117 static int read_rl_from_var(struct expression *call, char *p, char **end, struct
118 {
119         struct expression *arg;
120         struct smatch_state *state;
121         long param;
122         char *name;
123         struct symbol *sym;
124         char buf[256];
125         int star;

127         p++;
128         param = strtol(p, (char **)&p, 10);
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, const char *p, const char **end
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, (char **)end, 10);
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 const char *read_op(const char *p)
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, const char *math, struct range_l
202 int parse_call_math_rl(struct expression *call, char *math, struct range_list **
203 {
204         struct range_list *tmp;
205         const char *c;
205         char *c;

207         /* try to implement shunting yard algorithm. */

209         c = math;
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 }
```

*unchanged_portion_omitted_*

```
347 static int is_mtag_sval(sval_t sval)
348 {
349         if (!is_ptr_type(sval.type))
350                 return 0;
351         if (sval_cmp(sval, valid_ptr_min_sval) >= 0 &&
352             sval_cmp(sval, valid_ptr_max_sval) <= 0)
353                 return 1;
354         return 0;
355 }

357 static int format_expr_helper(char *buf, int remaining, struct expression *expr)
358 {
359         sval_t sval;
360         int ret;
361         char *cur;

363         if (!expr)
364                 return 0;

366         cur = buf;

368         if (expr->type == EXPR_BINOP) {
369                 ret = format_expr_helper(cur, remaining, expr->left);
370                 if (ret == 0)
371                         return 0;
372                 remaining -= ret;
373                 if (remaining <= 0)
374                         return 0;
375                 cur += ret;

377                 ret = snprintf(cur, remaining, " %s ", show_special(expr->op));
378                 remaining -= ret;
379                 if (remaining <= 0)
380                         return 0;
381                 cur += ret;

383                 ret = format_expr_helper(cur, remaining, expr->right);
384                 if (ret == 0)
385                         return 0;
386                 remaining -= ret;
387                 if (remaining <= 0)
388                         return 0;
389                 cur += ret;
390                 return cur - buf;
391         }

393         if (!param_was_set(expr) && get_implied_value(expr, &sval) && !is_mtag_s
383         if (get_implied_value(expr, &sval)) {
394                 ret = snprintf(cur, remaining, "%s", sval_to_str(sval));
395                 remaining -= ret;
396                 if (remaining <= 0)
397                         return 0;
```



```
398                 return ret;
399         }

401         if (expr->type == EXPR_CALL)
402                 return format_call_to_param_mapping(cur, remaining, expr);

404         return format_variable_helper(cur, remaining, expr);
405 }
```

*unchanged_portion_omitted_*

```
442 char *get_value_in_terms_of_parameter_math_var_sym(const char *name, struct symb
443 {
444         struct expression *tmp, *expr;
445         char buf[256] = "";
446         int ret;
447         int cnt = 0;
448         sval_t sval;

450         expr = get_assigned_expr_name_sym(name, sym);
451         if (!expr)
452                 return NULL;
453         while ((tmp = get_assigned_expr(expr))) {
454                 expr = strip_expr(tmp);
455                 if (++cnt > 3)
456                         break;
457         }

459         if (get_implied_value(expr, &sval))
460                 return NULL;

462         ret = format_expr_helper(buf, sizeof(buf), expr);
463         if (ret == 0)
464                 return NULL;

466         return alloc_sname(buf);

468 }
```

*unchanged_portion_omitted_*

```
486 static char *swap_format(struct expression *call, char *format)
487 {
488         char buf[256];
489         sval_t sval;
490         long param;
491         struct expression *arg;
492         char *p;
493         char *out;
494         int ret;

496         if (format[0] == '$' && format[2] == '\0') {
497                 param = strtol(format + 1, NULL, 10);
498                 arg = get_argument_from_call_expr(call->args, param);
499                 if (!arg)
500                         return NULL;
501                 return format_expr(arg);
502         }

504         buf[0] = '\0';
505         p = format;
506         out = buf;
507         while (*p) {
508                 if (*p == '$') {
509                         p++;
510                         param = strtol(p, (char **)&p, 10);
496                         param = strtol(p, &p, 10);
511                         arg = get_argument_from_call_expr(call->args, param);
```

```
512                         if (!arg)
513                                 return NULL;
514                         param = get_arg_number(arg);
515                         if (param >= 0) {
516                                 ret = snprintf(out, buf + sizeof(buf) - out, "$%
517                                 out += ret;
518                                 if (out >= buf + sizeof(buf))
519                                         return NULL;
520                         } else if (get_implied_value(arg, &sval)) {
521                                 ret = snprintf(out, buf + sizeof(buf) - out, "%s
522                                 out += ret;
523                                 if (out >= buf + sizeof(buf))
524                                         return NULL;
525                         } else {
526                                 return NULL;
527                         }
528                 }
529                 *out = *p;
530                 p++;
531                 out++;
532         }
533         if (buf[0] == '\0')
534                 return NULL;
535         *out = '\0';
536         return alloc_sname(buf);
537 }
```

*unchanged_portion_omitted_*

```
654 void register_parse_call_math(int id)
655 {
656         int i;

658         my_id = id;

660         set_dynamic_states(my_id);

662         for (i = 0; i < ARRAY_SIZE(alloc_functions); i++)
663                 add_function_assign_hook(alloc_functions[i].func, &match_alloc,
664                                          INT_PTR(alloc_functions[i].param));
665         add_hook(&match_call_assignment, CALL_ASSIGNMENT_HOOK);
666         add_split_return_callback(print_returned_allocations);
667 }
```

*unchanged_portion_omitted_*

****
2172 Mon Aug 5 08:38:42 2019
new/usr/src/tools/smatch/src/smatch_passes_array_size.c
11506 smatch resync
****
_____unchanged_portion_omitted_

```
 39 static void match_call(struct expression *expr)
 40 {
 41         struct expression *arg;
 42         struct symbol *type, *arg_type;
 42         struct symbol *type;
 43         int size, bytes;
 44         int i, nr;
 45         char buf[16];
 46         char elem_count[8];
 47         char byte_count[8];

 49         snprintf(elem_count, sizeof(elem_count), "%d", ELEM_COUNT);
 50         snprintf(byte_count, sizeof(byte_count), "%d", BYTE_COUNT);

 52         i = -1;
 53         FOR_EACH_PTR(expr->args, arg) {
 54                 i++;
 55                 type = get_type(arg);
 56                 if (!type || (type->type != SYM_PTR && type->type != SYM_ARRAY))
 57                         continue;
 58                 arg_type = get_arg_type(expr->fn, i);
 59                 if (arg_type != type)
 60                         continue;

 62                 size = get_array_size(arg);
 63                 if (size > 0) {
 64                         nr = find_param_eq(expr, size);
 65                         if (nr >= 0) {
 66                                 snprintf(buf, sizeof(buf), "==$%d", nr);
 67                                 sql_insert_caller_info(expr, ELEM_COUNT, i, buf,
 58                                 snprintf(buf, sizeof(buf), "%d", nr);
 59                                 sql_insert_caller_info(expr, ARRAYSIZE_ARG, i, b
 68                                 continue;
 69                         }
 70                 }
 71                 bytes = get_array_size_bytes(arg);
 72                 if (bytes > 0) {
 73                         nr = find_param_eq(expr, bytes);
 74                         if (nr >= 0) {
 75                                 snprintf(buf, sizeof(buf), "==$%d", nr);
 76                                 sql_insert_caller_info(expr, BYTE_COUNT, i, buf,
 67                                 snprintf(buf, sizeof(buf), "%d", nr);
 68                                 sql_insert_caller_info(expr, SIZEOF_ARG, i, buf,
 77                                 continue;
 78                         }
 79                 }
 80         } END_FOR_EACH_PTR(arg);
 81 }
```

_____unchanged_portion_omitted_

```
**********************************************************
   53282 Mon Aug  5 08:38:42 2019
new/usr/src/tools/smatch/src/smatch_ranges.c
11506 smatch resync
**********************************************************
```

```
/*
 * 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);

static bool is_err_ptr(sval_t sval)
{
        if (option_project != PROJ_KERNEL)
                return false;
        if (!type_is_ptr(sval.type))
                return false;
        if (sval.uvalue < -4095ULL)
                return false;
        return true;
}

static char *get_err_pointer_str(struct data_range *drange)
{
        static char buf[20];

        /*
         * The kernel has error pointers where you do essentially:
         *
         * return (void *)(unsigned long)-12;
         *
         * But what I want here is to print -12 instead of the unsigned version
         * of that.
         *
         */
        if (!is_err_ptr(drange->min))
                return NULL;

        if (drange->min.value == drange->max.value)
                snprintf(buf, sizeof(buf), "(%lld)", drange->min.value);
        else
                snprintf(buf, sizeof(buf), "(%lld)-(%lld)", drange->min.value, d
        return buf;
}
```



```
char *show_rl(struct range_list *list)
{
        struct data_range *prev_drange = NULL;
        struct data_range *tmp;
        char full[255];
        char *p = full;
        char *prev = full;
        char *err_ptr;
        int remain;
        char full[512];
        int i = 0;

        full[0] = '\0';

        full[sizeof(full) - 1] = '\0';
        FOR_EACH_PTR(list, tmp) {
                remain = full + sizeof(full) - p;
                if (remain < 48) {
                        snprintf(prev, full + sizeof(full) - prev, ",%s-%s",
                                 sval_to_str(prev_drange->min),
                                 sval_to_str(sval_type_max(prev_drange->min.type
                        break;
                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;
                }
                prev_drange = tmp;
                prev = p;

                err_ptr = get_err_pointer_str(tmp);
                if (err_ptr) {
                        p += snprintf(p, remain, "%s%s", i++ ? "," : "", err_ptr
                } else if (sval_cmp(tmp->min, tmp->max) == 0) {
                        p += snprintf(p, remain, "%s%s", i++ ? "," : "",
                                      sval_to_str(tmp->min));
                } else {
                        p += snprintf(p, remain, "%s%s-%s", i++ ? "," : "",
                                      sval_to_str(tmp->min),
                                      sval_to_str(tmp->max));
                }
                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);
}
```

*unchanged_portion_omitted_*

```
static int truncates_nicely(struct symbol *type, sval_t min, sval_t max)
{
        unsigned long long mask;
        int bits = type_bits(type);

        if (bits >= type_bits(min.type))
                return 0;

        mask = -1ULL << bits;
        return (min.uvalue & mask) == (max.uvalue & mask);
}
```

```
144 static void add_range_t(struct symbol *type, struct range_list **rl, sval_t min,
145 {
146         /* If we're just adding a number, cast it and add it */
147         if (sval_cmp(min, max) == 0) {
148                 add_range(rl, sval_cast(type, min), sval_cast(type, max));
149                 return;
150         }

152         /* If the range is within the type range then add it */
153         if (sval_fits(type, min) && sval_fits(type, max)) {
154                 add_range(rl, sval_cast(type, min), sval_cast(type, max));
155                 return;
156         }

158         if (truncates_nicely(type, min, max)) {
159                 add_range(rl, sval_cast(type, min), sval_cast(type, max));
160                 return;
161         }

163         /*
164          * If the range we are adding has more bits than the range type then
165          * add the whole range type.  Eg:
166          * 0x8000000000000000 - 0xf000000000000000 -> cast to int
167          *
100          * This isn't totally the right thing to do.  We could be more granular.
168          */
169         if (sval_too_big(type, min) || sval_too_big(type, max)) {
170                 add_range(rl, sval_type_min(type), sval_type_max(type));
171                 return;
172         }

174         /* Cast negative values to high positive values */
175         if (sval_is_negative(min) && type_unsigned(type)) {
176                 if (sval_is_positive(max)) {
177                         if (sval_too_high(type, max)) {
178                                 add_range(rl, sval_type_min(type), sval_type_max
179                                 return;
180                         }
181                         add_range(rl, sval_type_val(type, 0), sval_cast(type, ma
182                         max = sval_type_max(type);
183                 } else {
184                         max = sval_cast(type, max);
185                 }
186                 min = sval_cast(type, min);
187                 add_range(rl, min, max);
188         }

190         /* Cast high positive numbers to negative */
191         if (sval_unsigned(max) && sval_is_negative(sval_cast(type, max))) {
192                 if (!sval_is_negative(sval_cast(type, min))) {
193                         add_range(rl, sval_cast(type, min), sval_type_max(type))
194                         min = sval_type_min(type);
195                 } else {
196                         min = sval_cast(type, min);
197                 }
198                 max = sval_cast(type, max);
199                 add_range(rl, min, max);
200         }

202         add_range(rl, sval_cast(type, min), sval_cast(type, max));
203         return;
204 }

206 static int str_to_comparison_arg_helper(const char *str,
207                 struct expression *call, int *comparison,
208                 struct expression **arg, const char **endp)
```



```
141                 struct expression **arg, char **endp)
209 {
210         int param;
211         const char *c = str;
144         char *c = (char *)str;

213         if (*c != '[')
214                 return 0;
215         c++;

217         if (*c == '<') {
218                 c++;
219                 if (*c == '=') {
220                         *comparison = SPECIAL_LTE;
221                         c++;
222                 } else {
223                         *comparison = '<';
224                 }
225         } else if (*c == '=') {
226                 c++;
227                 c++;
228                 *comparison = SPECIAL_EQUAL;
229         } else if (*c == '>') {
230                 c++;
231                 if (*c == '=') {
232                         *comparison = SPECIAL_GTE;
233                         c++;
234                 } else {
235                         *comparison = '>';
236                 }
237         } else if (*c == '!') {
238                 c++;
239                 c++;
240                 *comparison = SPECIAL_NOTEQUAL;
241         } else if (*c == '$') {
242                 *comparison = SPECIAL_EQUAL;
243         } else {
244                 return 0;
245         }

247         if (*c != '$')
248                 return 0;
249         c++;

251         param = strtoll(c, (char **)&c, 10);
252         if (*c == ',' || *c == ']')
182         param = strtoll(c, &c, 10);
183         if (*c == ']')
253                 c++; /* skip the ']' character */
254         if (endp)
255                 *endp = (char *)c;

257         if (!call)
258                 return 0;
259         *arg = get_argument_from_call_expr(call->args, param);
260         if (!*arg)
261                 return 0;
262         if (*c == '-' && *(c + 1) == '>') {
263                 char buf[256];
264                 int n;

266                 n = snprintf(buf, sizeof(buf), "$%s", c);
267                 if (n >= sizeof(buf))
268                         return 0;
269                 if (buf[n - 1] == ']')
270                         buf[n - 1] = '\0';
```

```
                *arg = gen_expression_from_key(*arg, buf);
                while (*c && *c != ']')
                        c++;
        }
        return 1;
}
```

*unchanged_portion_omitted_*

```
static int get_val_from_key(int use_max, struct symbol *type, const char *c, str
static int get_val_from_key(int use_max, struct symbol *type, char *c, struct ex
{
        struct expression *arg;
        int comparison;
        sval_t ret, tmp;

        if (use_max)
                ret = sval_type_max(type);
        else
                ret = sval_type_min(type);

        if (!str_to_comparison_arg_helper(c, call, &comparison, &arg, endp)) {
                *sval = ret;
                return 0;
        }

        if (use_max && get_implied_max(arg, &tmp)) {
                ret = tmp;
                if (comparison == '<') {
                        tmp.value = 1;
                        ret = sval_binop(ret, '-', tmp);
                }
        }
        if (!use_max && get_implied_min(arg, &tmp)) {
                ret = tmp;
                if (comparison == '>') {
                        tmp.value = 1;
                        ret = sval_binop(ret, '+', tmp);
                }
        }

        *sval = ret;
        return 1;
}
```

*unchanged_portion_omitted_*

```
static struct range_list *filter_by_comparison_call(const char *c, struct expres
static struct range_list *filter_by_comparison_call(char *c, struct expression *
{
        struct symbol *type;
        struct expression *arg;
        struct range_list *casted_start, *right_orig;
        int comparison;

        if (!str_to_comparison_arg_helper(c, call, &comparison, &arg, endp))
                return start_rl;

        if (!get_implied_rl(arg, &right_orig))
                return start_rl;

        type = &int_ctype;
        if (type_positive_bits(rl_type(start_rl)) > type_positive_bits(type))
                type = rl_type(start_rl);
        if (type_positive_bits(rl_type(right_orig)) > type_positive_bits(type))
                type = rl_type(right_orig);

        casted_start = cast_rl(type, start_rl);
```



```
        right_orig = cast_rl(type, right_orig);

        filter_by_comparison(&casted_start, comparison, right_orig);
        return cast_rl(rl_type(start_rl), casted_start);
}

static sval_t parse_val(int use_max, struct expression *call, struct symbol *typ
static sval_t parse_val(int use_max, struct expression *call, struct symbol *typ
{
        const char *start = c;
        char *start = c;
        sval_t ret;

        if (!strncmp(start, "max", 3)) {
                ret = sval_type_max(type);
                c += 3;
        } else if (!strncmp(start, "u64max", 6)) {
                ret = sval_type_val(type, ULLONG_MAX);
                c += 6;
        } else if (!strncmp(start, "s64max", 6)) {
                ret = sval_type_val(type, LLONG_MAX);
                c += 6;
        } else if (!strncmp(start, "u32max", 6)) {
                ret = sval_type_val(type, UINT_MAX);
                c += 6;
        } else if (!strncmp(start, "s32max", 6)) {
                ret = sval_type_val(type, INT_MAX);
                c += 6;
        } else if (!strncmp(start, "u16max", 6)) {
                ret = sval_type_val(type, USHRT_MAX);
                c += 6;
        } else if (!strncmp(start, "s16max", 6)) {
                ret = sval_type_val(type, SHRT_MAX);
                c += 6;
        } else if (!strncmp(start, "min", 3)) {
                ret = sval_type_min(type);
                c += 3;
        } else if (!strncmp(start, "s64min", 6)) {
                ret = sval_type_val(type, LLONG_MIN);
                c += 6;
        } else if (!strncmp(start, "s32min", 6)) {
                ret = sval_type_val(type, INT_MIN);
                c += 6;
        } else if (!strncmp(start, "s16min", 6)) {
                ret = sval_type_val(type, SHRT_MIN);
                c += 6;
        } else if (!strncmp(start, "long_min", 8)) {
                ret = sval_type_val(type, LONG_MIN);
                c += 8;
        } else if (!strncmp(start, "long_max", 8)) {
                ret = sval_type_val(type, LONG_MAX);
                c += 8;
        } else if (!strncmp(start, "ulong_max", 9)) {
                ret = sval_type_val(type, ULONG_MAX);
                c += 9;
        } else if (!strncmp(start, "ptr_max", 7)) {
                ret = sval_type_val(type, valid_ptr_max);
                c += 7;
        } else if (start[0] == '[') {
                /* this parses [==p0] comparisons */
                get_val_from_key(1, type, start, call, &c, &ret);
        } else if (type_positive_bits(type) == 64) {
                ret = sval_type_val(type, strtoull(start, (char **)&c, 0));
                ret = sval_type_val(type, strtoull(start, &c, 0));
        } else {
                ret = sval_type_val(type, strtoll(start, (char **)&c, 0));
```

```
403                 ret = sval_type_val(type, strtoll(start, &c, 0));
473         }
474         *endp = c;
475         return ret;
476 }

478 static const char *jump_to_call_math(const char *value)
409 static char *jump_to_call_math(char *value)
479 {
480         const char *c = value;
411         char *c = value;

482         while (*c && *c != '[')
483                 c++;

485         if (!*c)
486                 return NULL;
487         c++;
488         if (*c == '<' || *c == '=' || *c == '>' || *c == '!')
489                 return NULL;

491         return c;
492 }

494 static void str_to_rl_helper(struct expression *call, struct symbol *type, const
425 static void str_to_rl_helper(struct expression *call, struct symbol *type, char
495 {
496         struct range_list *rl_tmp = NULL;
497         sval_t prev_min, min, max;
498         const char *c;
428         sval_t min, max;
429         char *c;

500         prev_min = sval_type_min(type);
501         min = sval_type_min(type);
502         max = sval_type_max(type);
503         c = str;
504         while (*c != '\0' && *c != '[') {
505                 if (*c == '+') {
506                         if (sval_cmp(min, sval_type_min(type)) != 0)
507                                 min = max;
508                         max = sval_type_max(type);
509                         add_range_t(type, &rl_tmp, min, max);
510                         break;
511                 }
512                 if (*c == '(')
513                         c++;
514                 min = parse_val(0, call, type, c, &c);
515                 if (!sval_fits(type, min))
516                         min = sval_type_min(type);
517                 max = min;
518                 if (*c == ')')
519                         c++;
520                 if (*c == '\0' || *c == '[') {
521                         add_range_t(type, &rl_tmp, min, min);
522                         break;
523                 }
524                 if (*c == ',') {
525                         add_range_t(type, &rl_tmp, min, min);
526                         c++;
527                         continue;
528                 }
529                 if (*c == '+') {
530                         min = prev_min;
531                         max = sval_type_max(type);
532                         add_range_t(type, &rl_tmp, min, max);
```



```
460                         min = sval_type_max(type);
533                         c++;
534                         if (*c == '[' || *c == '\0')
535                                 break;
536                 }
537                 if (*c != '-') {
538                         sm_msg("debug XXX: trouble parsing %s c = %s", str, c);
539                         break;
540                 }
541                 c++;
542                 if (*c == '(')
543                         c++;
544                 max = parse_val(1, call, type, c, &c);
545                 if (!sval_fits(type, max))
546                         max = sval_type_max(type);
547                 if (*c == '+') {
548                         max = sval_type_max(type);
549                         add_range_t(type, &rl_tmp, min, max);
550                         c++;
551                         if (*c == '[' || *c == '\0')
552                                 break;
553                 }
554                 prev_min = max;
555                 add_range_t(type, &rl_tmp, min, max);
556                 if (*c == ')')
557                         c++;
558                 if (*c == ',')
559                         c++;
560         }

562         *rl = rl_tmp;
563         *endp = c;
564 }

566 static void str_to_dinfo(struct expression *call, struct symbol *type, const cha
488 static void str_to_dinfo(struct expression *call, struct symbol *type, char *val
567 {
568         struct range_list *math_rl;
569         const char *call_math;
570         const char *c;
491         char *call_math;
492         char *c;
571         struct range_list *rl = NULL;

573         if (!type)
574                 type = &llong_ctype;

576         if (strcmp(value, "empty") == 0)
577                 return;

579         if (strncmp(value, "[==$", 4) == 0) {
580                 struct expression *arg;
581                 int comparison;

583                 if (!str_to_comparison_arg(value, call, &comparison, &arg))
584                         return;
585                 if (!get_implied_rl(arg, &rl))
586                         return;
587                 goto cast;
588         }

590         str_to_rl_helper(call, type, value, &c, &rl);
591         if (*c == '\0')
592                 goto cast;

594         call_math = jump_to_call_math(value);
```

```
595         if (call_math && parse_call_math_rl(call, call_math, &math_rl)) {
596                 rl = rl_intersection(rl, math_rl);
597                 goto cast;
598         }

600         /*
601          * For now if we already tried to handle the call math and couldn't
602          * figure it out then bail.
603          */
604         if (jump_to_call_math(c) == c + 1)
605                 goto cast;

607         rl = filter_by_comparison_call(c, call, &c, rl);

609 cast:
610         rl = cast_rl(type, rl);
611         dinfo->value_ranges = rl;
612 }

614 static int rl_is_sane(struct range_list *rl)
615 {
616         struct data_range *tmp;
617         struct symbol *type;

619         type = rl_type(rl);
620         FOR_EACH_PTR(rl, tmp) {
621                 if (!sval_fits(type, tmp->min))
622                         return 0;
623                 if (!sval_fits(type, tmp->max))
624                         return 0;
625                 if (sval_cmp(tmp->min, tmp->max) > 0)
626                         return 0;
627         } END_FOR_EACH_PTR(tmp);

629         return 1;
630 }

632 void str_to_rl(struct symbol *type, char *value, struct range_list **rl)
633 {
634         struct data_info dinfo = {};

636         str_to_dinfo(NULL, type, value, &dinfo);
637         if (!rl_is_sane(dinfo.value_ranges))
638                 dinfo.value_ranges = alloc_whole_rl(type);
639         *rl = dinfo.value_ranges;
640 }

642 void call_results_to_rl(struct expression *expr, struct symbol *type, const char
544 void call_results_to_rl(struct expression *expr, struct symbol *type, char *valu
643 {
644         struct data_info dinfo = {};

646         str_to_dinfo(strip_expr(expr), type, value, &dinfo);
647         *rl = dinfo.value_ranges;
648 }
```

*unchanged_portion_omitted_*

```
662 int is_unknown_ptr(struct range_list *rl)
663 {
664         struct data_range *drange;
665         int cnt = 0;

667         if (is_whole_rl(rl))
668                 return 1;

670         FOR_EACH_PTR(rl, drange) {
```



```
671                 if (++cnt >= 3)
672                         return 0;
673                 if (sval_cmp(drange->min, valid_ptr_min_sval) == 0 &&
674                     sval_cmp(drange->max, valid_ptr_max_sval) == 0)
675                         return 1;
676         } END_FOR_EACH_PTR(drange);

678         return 0;
679 }

681 int is_whole_rl_non_zero(struct range_list *rl)
682 {
683         struct data_range *drange;

685         if (ptr_list_empty(rl))
686                 return 0;
687         drange = first_ptr_list((struct ptr_list *)rl);
688         if (sval_unsigned(drange->min) &&
689             drange->min.value == 1 &&
690             sval_is_max(drange->max))
691                 return 1;
692         if (!sval_is_min(drange->min) || drange->max.value != -1)
693                 return 0;
694         drange = last_ptr_list((struct ptr_list *)rl);
695         if (drange->min.value != 1 || !sval_is_max(drange->max))
696                 return 0;
697         return 1;
698 }
```

*unchanged_portion_omitted_*

```
792 static bool collapse_pointer_rl(struct range_list **rl, sval_t min, sval_t max)
793 {
794         struct range_list *new_rl = NULL;
795         struct data_range *tmp;
796         static bool recurse;
797         bool ret = false;
798         int cnt = 0;

800         /*
801          * With the mtag work, then we end up getting huge lists of mtags.
802          * That seems cool, but the problem is that we can only store about
803          * 8-10 mtags in the DB before we truncate the list.  Also the mtags
804          * aren't really used at all so it's a waste of resources for now...
805          * In the future, we maybe will revisit this code.
806          *
807          */

809         if (recurse)
810                 return false;
811         recurse = true;
812         if (!type_is_ptr(min.type))
813                 goto out;

815         if (ptr_list_size((struct ptr_list *)*rl) < 8)
816                 goto out;
817         FOR_EACH_PTR(*rl, tmp) {
818                 if (!is_err_ptr(tmp->min))
819                         cnt++;
820         } END_FOR_EACH_PTR(tmp);
821         if (cnt < 8)
822                 goto out;

824         FOR_EACH_PTR(*rl, tmp) {
825                 if (sval_cmp(tmp->min, valid_ptr_min_sval) >= 0 &&
826                     sval_cmp(tmp->max, valid_ptr_max_sval) <= 0)
827                         add_range(&new_rl, valid_ptr_min_sval, valid_ptr_max_sva
```

```
828                 else
829                         add_range(&new_rl, tmp->min, tmp->max);
830         } END_FOR_EACH_PTR(tmp);
831
832         add_range(&new_rl, min, max);
833
834         *rl = new_rl;
835         ret = true;
836 out:
837         recurse = false;
838         return ret;
839 }
840
841 extern int rl_ptrlist_hack;
842 void add_range(struct range_list **list, sval_t min, sval_t max)
843 {
844         struct data_range *tmp;
845         struct data_range *new = NULL;
846         int check_next = 0;
847
848         /*
849          * There is at least on valid reason why the types might be confusing
850          * and that's when you have a void pointer and on some paths you treat
851          * it as a u8 pointer and on other paths you treat it as a u16 pointer.
852          * This case is hard to deal with.
853          *
854          * There are other cases where we probably should be more specific about
855          * the types than we are.  For example, we end up merging a lot of ulong
856          * with pointers and I have not figured out why we do that.
857          *
858          * But this hack works for both cases, I think.  We cast it to pointers
859          * or we use the bigger size.
860          *
861          */
862         if (*list && rl_type(*list) != min.type) {
863                 if (rl_type(*list)->type == SYM_PTR) {
864                         min = sval_cast(rl_type(*list), min);
865                         max = sval_cast(rl_type(*list), max);
866                 } else if (min.type->type == SYM_PTR) {
867                         *list = cast_rl(min.type, *list);
868                 } else if (type_bits(rl_type(*list)) >= type_bits(min.type)) {
869                         min = sval_cast(rl_type(*list), min);
870                         max = sval_cast(rl_type(*list), max);
871                 } else {
872                         *list = cast_rl(min.type, *list);
873                 }
874         }
875
876         if (sval_cmp(min, max) > 0) {
877                 min = sval_type_min(min.type);
878                 max = sval_type_max(min.type);
879         }
880
881         if (collapse_pointer_rl(list, min, max))
882                 return;
883
884         /*
885          * FIXME:  This has a problem merging a range_list like: min-0,3-max
886          * with a range like 1-2.  You end up with min-2,3-max instead of
887          * just min-max.
888          */
889         FOR_EACH_PTR(*list, tmp) {
890                 if (check_next) {
891                         /* Sometimes we overlap with more than one range
892                            so we have to delete or modify the next range. */
893                         if (!sval_is_max(max) && max.value + 1 == tmp->min.value
```



```
894                                 /* join 2 ranges here */
895                                 new->max = tmp->max;
896                                 DELETE_CURRENT_PTR(tmp);
897                                 return;
898                         }
899
900                         /* Doesn't overlap with the next one. */
901                         if (sval_cmp(max, tmp->min) < 0)
902                                 return;
903
904                         if (sval_cmp(max, tmp->max) <= 0) {
905                                 /* Partially overlaps the next one. */
906                                 new->max = tmp->max;
907                                 DELETE_CURRENT_PTR(tmp);
908                                 return;
909                         } else {
910                                 /* Completely overlaps the next one. */
911                                 DELETE_CURRENT_PTR(tmp);
912                                 /* there could be more ranges to delete */
913                                 continue;
914                         }
915                 }
916                 if (!sval_is_max(max) && max.value + 1 == tmp->min.value) {
917                         /* join 2 ranges into a big range */
918                         new = alloc_range(min, tmp->max);
919                         REPLACE_CURRENT_PTR(tmp, new);
920                         return;
921                 }
922                 if (sval_cmp(max, tmp->min) < 0) { /* new range entirely below *
923                         new = alloc_range(min, max);
924                         INSERT_CURRENT(new, tmp);
925                         return;
926                 }
927                 if (sval_cmp(min, tmp->min) < 0) { /* new range partially below
928                         if (sval_cmp(max, tmp->max) < 0)
929                                 max = tmp->max;
930                         else
931                                 check_next = 1;
932                         new = alloc_range(min, max);
933                         REPLACE_CURRENT_PTR(tmp, new);
934                         if (!check_next)
935                                 return;
936                         continue;
937                 }
938                 if (sval_cmp(max, tmp->max) <= 0) /* new range already included
939                         return;
940                 if (sval_cmp(min, tmp->max) <= 0) { /* new range partially above
941                         min = tmp->min;
942                         new = alloc_range(min, max);
943                         REPLACE_CURRENT_PTR(tmp, new);
944                         check_next = 1;
945                         continue;
946                 }
947                 if (!sval_is_min(min) && min.value - 1 == tmp->max.value) {
948                         /* join 2 ranges into a big range */
949                         new = alloc_range(tmp->min, max);
950                         REPLACE_CURRENT_PTR(tmp, new);
951                         check_next = 1;
952                         continue;
953                 }
954                 /* the new range is entirely above the existing ranges */
955         } END_FOR_EACH_PTR(tmp);
956         if (check_next)
957                 return;
958         new = alloc_range(min, max);
```

```
 960         rl_ptrlist_hack = 1;
 961         add_ptr_list(list, new);
 962         rl_ptrlist_hack = 0;
 963 }
______unchanged_portion_omitted_

1391 int rl_fits_in_type(struct range_list *rl, struct symbol *type)
1222 static int rl_is_sane(struct range_list *rl)
1392 {
1393         if (type_bits(rl_type(rl)) <= type_bits(type))
1394                 return 1;
1395         if (sval_cmp(rl_max(rl), sval_type_max(type)) > 0)
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))
1396                 return 0;
1397         if (sval_is_negative(rl_min(rl)) &&
1398             sval_cmp(rl_min(rl), sval_type_min(type)) < 0)
1231                 if (!sval_fits(type, tmp->max))
1399                 return 0;
1233                 if (sval_cmp(tmp->min, tmp->max) > 0)
1234                         return 0;
1235         } END_FOR_EACH_PTR(tmp);

1400         return 1;
1401 }
______unchanged_portion_omitted_

1476 struct range_list *rl_filter(struct range_list *rl, struct range_list *filter)
1313 struct range_list *rl_invert(struct range_list *orig)
1477 {
1315         struct range_list *ret = NULL;
1478         struct data_range *tmp;
1317         sval_t gap_min, abs_max, sval;

1480         FOR_EACH_PTR(filter, tmp) {
1481                 rl = remove_range(rl, tmp->min, tmp->max);
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);
1482         } END_FOR_EACH_PTR(tmp);

1484         return rl;
1337         if (sval_cmp(gap_min, abs_max) <= 0)
1338                 add_range(&ret, gap_min, abs_max);

1340         return ret;
1485 }

1487 struct range_list *do_intersection(struct range_list *one_rl, struct range_list
```



```
1343 struct range_list *rl_filter(struct range_list *rl, struct range_list *filter)
1488 {
1489         struct data_range *one, *two;
1490         struct range_list *ret = NULL;
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);

1493         PREPARE_PTR_LIST(one_rl, one);
1494         PREPARE_PTR_LIST(two_rl, two);

1496         while (true) {
1497                 if (!one || !two)
1498                         break;
1499                 if (sval_cmp(one->max, two->min) < 0) {
1500                         NEXT_PTR_LIST(one);
1501                         continue;
1502                 }
1503                 if (sval_cmp(one->min, two->min) < 0 && sval_cmp(one->max, two->
1504                         add_range(&ret, two->min, one->max);
1505                         NEXT_PTR_LIST(one);
1506                         continue;
1507                 }
1508                 if (sval_cmp(one->min, two->min) >= 0 && sval_cmp(one->max, two-
1509                         add_range(&ret, one->min, one->max);
1510                         NEXT_PTR_LIST(one);
1511                         continue;
1512                 }
1513                 if (sval_cmp(one->min, two->min) < 0 && sval_cmp(one->max, two->
1514                         add_range(&ret, two->min, two->max);
1515                         NEXT_PTR_LIST(two);
1516                         continue;
1517                 }
1518                 if (sval_cmp(one->min, two->max) <= 0 && sval_cmp(one->max, two-
1519                         add_range(&ret, one->min, two->max);
1520                         NEXT_PTR_LIST(two);
1521                         continue;
1522                 }
1523                 if (sval_cmp(one->min, two->max) <= 0) {
1524                         sm_fatal("error calculating intersection of '%s' and '%s
1525                         return NULL;
1526                 }
1527                 NEXT_PTR_LIST(two);
1528         }

1530         FINISH_PTR_LIST(two);
1531         FINISH_PTR_LIST(one);

1533         return ret;
1351         return rl;
1534 }

1536 struct range_list *rl_intersection(struct range_list *one, struct range_list *tw
1537 {
1356         struct range_list *one_orig;
1357         struct range_list *two_orig;
1538         struct range_list *ret;
1539         struct symbol *ret_type;
1540         struct symbol *small_type;
1541         struct symbol *large_type;

1543         if (!one || !two)
1363         if (!two)
1544                 return NULL;
```

```
1365         if (!one)
1366                 return NULL;

1368         one_orig = one;
1369         two_orig = two;

1546         ret_type = rl_type(one);
1547         small_type = rl_type(one);
1548         large_type = rl_type(two);

1550         if (type_bits(rl_type(two)) < type_bits(small_type)) {
1551                 small_type = rl_type(two);
1552                 large_type = rl_type(one);
1553         }

1555         one = cast_rl(large_type, one);
1556         two = cast_rl(large_type, two);

1558         ret = do_intersection(one, 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);

1559         return cast_rl(ret_type, ret);
1560 }
```

_____*unchanged_portion_omitted_*

```
1682 static struct range_list *ptr_add_mult(struct range_list *left, int op, struct r
1683 {
1684         struct range_list *ret;
1685         sval_t l_sval, r_sval, res;

1687         /*
1688          * This function is sort of the wrong API because it takes two pointer
1689          * and adds them together.  The caller is expected to figure out
1690          * alignment.  Neither of those are the correct things to do.
1691          *
1692          * Really this function is quite bogus...
1693          */

1695         if (rl_to_sval(left, &l_sval) && rl_to_sval(right, &r_sval)) {
1696                 res = sval_binop(l_sval, op, r_sval);
1697                 return alloc_rl(res, res);
1698         }

1700         if (rl_min(left).value != 0 || rl_max(right).value != 0) {
1701                 ret = alloc_rl(valid_ptr_min_sval, valid_ptr_max_sval);
1702                 return cast_rl(rl_type(left), ret);
1703         }

1705         return alloc_whole_rl(rl_type(left));
1706 }
```



```
1708 static struct range_list *handle_add_mult_rl(struct range_list *left, int op, st
1709 {
1710         sval_t min, max;

1712         if (type_is_ptr(rl_type(left)) || type_is_ptr(rl_type(right)))
1713                 return ptr_add_mult(left, op, right);

1715         if (sval_binop_overflows(rl_min(left), op, rl_min(right)))
1716                 return NULL;
1717         min = sval_binop(rl_min(left), op, rl_min(right));

1719         if (sval_binop_overflows(rl_max(left), op, rl_max(right)))
1720                 return NULL;
1721         max = sval_binop(rl_max(left), op, rl_max(right));

1723         return alloc_rl(min, max);
1724 }
```

_____*unchanged_portion_omitted_*

```
1836 static sval_t sval_lowest_set_bit(sval_t sval)
1837 {
1838         sval_t ret = { .type = sval.type };
1839         int i;

1841         for (i = 0; i < 64; i++) {
1842                 if (sval.uvalue & 1ULL << i) {
1843                         ret.uvalue = (1ULL << i);
1844                         return ret;
1845                 }
1846         }
1847         return ret;
1848 }

1850 static struct range_list *handle_AND_rl_sval(struct range_list *rl, sval_t sval)
1851 {
1852         struct range_list *known_rl;
1853         sval_t zero = { 0 };
1854         sval_t min;

1856         zero.type = sval.type;
1857         zero.value = 0;

1859         if (sm_fls64(rl_max(rl).uvalue) < find_first_zero_bit(sval.uvalue) &&
1860             sm_fls64(rl_min(rl).uvalue) < find_first_zero_bit(sval.uvalue))
1861                 return rl;

1863         min = sval_lowest_set_bit(sval);

1865         if (min.value != 0) {
1866                 sval_t max, mod;

1868                 max = rl_max(rl);
1869                 mod = sval_binop(max, '%', min);
1870                 if (mod.value) {
1871                         max = sval_binop(max, '-', mod);
1872                         max.value++;
1873                         if (max.value > 0 && sval_cmp(max, rl_max(rl)) < 0)
1874                                 rl = remove_range(rl, max, rl_max(rl));
1875                 }
1876         }

1878         known_rl = alloc_rl(min, sval);

1880         rl = rl_intersection(rl, known_rl);
1881         zero = rl_min(rl);
1882         zero.value = 0;
```

```
1883         add_range(&rl, zero, zero);

1885         return rl;
1886 }

1888 static struct range_list *fudge_AND_rl(struct range_list *rl)
1889 {
1890         struct range_list *ret;
1891         sval_t min;

1893         min = sval_lowest_set_bit(rl_min(rl));
1894         ret = clone_rl(rl);
1895         add_range(&ret, min, rl_min(rl));

1897         return ret;
1898 }

1900 static struct range_list *handle_AND_rl(struct range_list *left, struct range_li
1901 {
1902         sval_t sval, zero;
1903         struct range_list *rl;
1650         unsigned long long left_set, left_maybe;
1651         unsigned long long right_set, right_maybe;
1652         sval_t zero, max;

1905         if (rl_to_sval(left, &sval))
1906                 return handle_AND_rl_sval(right, sval);
1907         if (rl_to_sval(right, &sval))
1908                 return handle_AND_rl_sval(left, sval);

1910         left = fudge_AND_rl(left);
1911         right = fudge_AND_rl(right);

1913         rl = rl_intersection(left, right);
1914         zero = rl_min(rl);
1915         zero.value = 0;
1916         add_range(&rl, zero, zero);

1918         return rl;
1919 }

1921 static struct range_list *handle_lshift(struct range_list *left_orig, struct ran
1922 {
1923         struct range_list *left;
1924         struct data_range *tmp;
1925         struct range_list *ret = NULL;
1926         sval_t zero = { .type = rl_type(left_orig), };
1927         sval_t shift, min, max;
1928         bool add_zero = false;

1930         if (!rl_to_sval(right_orig, &shift) || sval_is_negative(shift))
1931                 return NULL;
1932         if (shift.value == 0)
1933                 return left_orig;

1935         /* Cast to unsigned for easier left shift math */
1936         if (type_positive_bits(rl_type(left_orig)) < 32)
1937                 left = cast_rl(&uint_ctype, left_orig);
1938         else if(type_positive_bits(rl_type(left_orig)) == 63)
1939                 left = cast_rl(&ullong_ctype, left_orig);
1940         else
1941                 left = left_orig;
1656         left_set = rl_bits_always_set(left);
1657         left_maybe = rl_bits_maybe_set(left);

1943         FOR_EACH_PTR(left, tmp) {
```



```
1944                 min = tmp->min;
1945                 max = tmp->max;
1659         right_set = rl_bits_always_set(right);
1660         right_maybe = rl_bits_maybe_set(right);

1947                 if (min.value == 0 || max.value > sval_type_max(max.type).uvalue
1948                         add_zero = true;
1949                 if (min.value == 0 && max.value == 0)
1950                         continue;
1951                 if (min.value == 0)
1952                         min.value = 1;
1953                 min = sval_binop(min, SPECIAL_LEFTSHIFT, shift);
1954                 max = sval_binop(max, SPECIAL_LEFTSHIFT, shift);
1955                 add_range(&ret, min, max);
1956         } END_FOR_EACH_PTR(tmp);
1662         zero = max = rl_min(left);
1663         zero.uvalue = 0;
1664         max.uvalue = fls_mask((left_maybe | right_maybe) ^ (left_set & right_set

1958         if (!rl_fits_in_type(ret, rl_type(left_orig)))
1959                 add_zero = true;
1960         ret = cast_rl(rl_type(left_orig), ret);
1961         if (add_zero)
1962                 add_range(&ret, zero, zero);

1964         return ret;
1666         return cast_rl(rl_type(left), alloc_rl(zero, max));
1965 }

1967 static struct range_list *handle_rshift(struct range_list *left_orig, struct ran
1968 {
1969         struct data_range *tmp;
1970         struct range_list *ret = NULL;
1971         sval_t shift, min, max;

1973         if (!rl_to_sval(right_orig, &shift) || sval_is_negative(shift))
1974                 return NULL;
1975         if (shift.value == 0)
1976                 return left_orig;

1978         FOR_EACH_PTR(left_orig, tmp) {
1979                 min = sval_binop(tmp->min, SPECIAL_RIGHTSHIFT, shift);
1980                 max = sval_binop(tmp->max, SPECIAL_RIGHTSHIFT, shift);
1981                 add_range(&ret, min, max);
1982         } END_FOR_EACH_PTR(tmp);

1984         return ret;
1985 }

1987 struct range_list *rl_binop(struct range_list *left, int op, struct range_list *
1988 {
1989         struct symbol *cast_type;
1990         sval_t left_sval, right_sval;
1991         struct range_list *ret = NULL;

1993         cast_type = rl_type(left);
1994         if (sval_type_max(rl_type(left)).uvalue < sval_type_max(rl_type(right)).
1995                 cast_type = rl_type(right);
1996         if (sval_type_max(cast_type).uvalue < INT_MAX)
1997                 cast_type = &int_ctype;

1999         left = cast_rl(cast_type, left);
2000         right = cast_rl(cast_type, right);

2002         if (!left && !right)
2003                 return NULL;
```

```
2005         if (rl_to_sval(left, &left_sval) && rl_to_sval(right, &right_sval)) {
2006                 sval_t val = sval_binop(left_sval, op, right_sval);
2007                 return alloc_rl(val, val);
2008         }

2010         switch (op) {
2011         case '%':
2012                 ret = handle_mod_rl(left, right);
2013                 break;
2014         case '/':
2015                 ret = handle_divide_rl(left, right);
2016                 break;
2017         case '*':
2018         case '+':
2019                 ret = handle_add_mult_rl(left, op, right);
2020                 break;
2021         case '|':
2022                 ret = handle_OR_rl(left, right);
2023                 break;
2024         case '^':
2025                 ret = handle_XOR_rl(left, right);
2026                 break;
2027         case '&':
2028                 ret = handle_AND_rl(left, right);
2029                 break;
2030         case '-':
2031                 ret = handle_sub_rl(left, right);
2032                 break;
1715         /* FIXME:  Do the rest as well */
2033         case SPECIAL_RIGHTSHIFT:
2034                 return handle_rshift(left, right);
2035         case SPECIAL_LEFTSHIFT:
2036                 return handle_lshift(left, right);
1718                 break;
2037         }

2039         return ret;
2040 }
```

_____*unchanged_portion_omitted_*

****
3750 Mon Aug 5 08:38:43 2019
new/usr/src/tools/smatch/src/smatch_real_absolute.c
11506 smatch resync
****

_____unchanged_portion_omitted_

```
 88 static void match_assign(struct expression *expr)
 89 {
 90         struct range_list *rl;
 91         struct symbol *type;
 92         sval_t sval;

 94         if (expr->op != '=')
 95                 return;
 96         if (is_fake_call(expr->right))
 97                 return;
 98         if (in_iterator_pre_statement())
 99                 return;

101         get_real_absolute_rl(expr->right, &rl);

103         type = get_type(expr->left);
104         if (!type)
105                 return;
106         if (type->type != SYM_PTR && type->type != SYM_BASETYPE &&
107             type->type != SYM_ENUM)
108                 return;

110         rl = cast_rl(type, rl);
111         if (is_whole_rl(rl) && !get_state_expr(my_id, expr->left))
112                 return;
113         /* These are handled by smatch_extra.c */
114         if (rl_to_sval(rl, &sval) && !get_state_expr(my_id, expr->left))
115                 return;

117         set_state_expr(my_id, expr->left, alloc_estate_rl(clone_rl(rl)));
118 }
```

_____unchanged_portion_omitted_

```
130 void register_real_absolute(int id)
131 {
132         my_id = id;

134         set_dynamic_states(my_id);
135         add_pre_merge_hook(my_id, &pre_merge_hook);
136         add_unmatched_state_hook(my_id, &empty_state);
137         add_merge_hook(my_id, &merge_estates);
138         add_modification_hook(my_id, &reset);

140         add_hook(&match_assign, ASSIGNMENT_HOOK);
141 }
```

_____unchanged_portion_omitted_

```
**********************************************************
    5971 Mon Aug  5 08:38:43 2019
new/usr/src/tools/smatch/src/smatch_return_to_param.c
11506 smatch resync
**********************************************************
```

_____unchanged_portion_omitted_

```
  52 char *map_call_to_other_name_sym(const char *name, struct symbol *sym, struct sy
  53 {
  54         struct smatch_state *state;
  55         int skip;
  56         char buf[256];

  58         /* skip 'foo->'.  This was checked in the caller. */
  59         skip = sym->ident->len + 2;
  59         skip = strlen(sym->ident->name) + 2;

  61         state = get_state(my_id, sym->ident->name, sym);
  62         if (!state || !state->data)
  63                 return NULL;

  65         snprintf(buf, sizeof(buf), "%s->%s", state->name, name + skip);
  66         *new_sym = state->data;
  67         return alloc_string(buf);
  68 }
```

_____unchanged_portion_omitted_

```
  90 static char *map_assignment_long_to_short(struct sm_state *sm, const char *name,
  91 {
  92         struct expression *orig_expr;
  93         struct symbol *orig_sym;
  94         int len;
  95         char buf[256];

  97         orig_expr = sm->state->data;
  98         if (!orig_expr)
  99                 return NULL;

 101         /*
 102          * Say we have an assignment like:
 103          *     foo->bar->my_ptr = my_ptr;
 104          * We still expect the function to carry on using "my_ptr" as the
 105          * shorter name.  That's not a long to short mapping.
 106          *
 107          */
 108         if (orig_expr->type == EXPR_SYMBOL)
 109                 return NULL;

 111         orig_sym = expr_to_sym(orig_expr);
 112         if (!orig_sym)
 113                 return NULL;
 114         if (sym != orig_sym)
 115                 return NULL;

 117         len = strlen(sm->state->name);
 118         if (strncmp(name, sm->state->name, len) != 0)
 119                 return NULL;

 121         if (name[len] == '.')
 122                 return NULL;
 123         if (!stack && name[len] != '-')
 124                 return NULL;
 125         snprintf(buf, sizeof(buf), "%s%s", sm->name, name + len);
 126         *new_sym = sm->sym;
 127         return alloc_string(buf);
 128 }
```



```
  90 /*
  91  * Normally, we expect people to consistently refer to variables by the shortest
  92  * name.  So they use "b->a" instead of "foo->bar.a" when both point to the
  93  * same memory location.  However, when we're dealing across function boundaries
  94  * then sometimes we pass frob(foo) which sets foo->bar.a.  In that case, we
  95  * translate it to the shorter name.  Smatch extra updates the shorter name,
  96  * which in turn updates the longer name.
  97  *
  98  */
  99 char *map_long_to_short_name_sym(const char *name, struct symbol *sym, struct sy
 139 static char *map_long_to_short_name_sym_helper(const char *name, struct symbol *
 100 {
 101         char *ret;
 102         struct sm_state *sm;

 104         *new_sym = NULL;

 106         FOR_EACH_SM(__get_cur_stree(), sm) {
 107                 if (sm->owner == my_id) {
 108                         ret = map_my_state_long_to_short(sm, name, sym, new_sym,
 109                         if (ret) {
 110                                 if (local_debug)
 111                                         sm_msg("%s: my_state: name = '%s' sm = '
 112                                                 __func__, name, show_sm(sm));
 148                         ret = map_my_state_long_to_short(sm, name, sym, new_sym,
 149                         if (ret)
 113                                 return ret;
 151                         continue;
 114                         }
 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;
 115                         continue;
 116                 }
 117         } END_FOR_EACH_SM(sm);

 119         return NULL;
 120 }

 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 }

 122 char *map_call_to_param_name_sym(struct expression *expr, struct symbol **sym)
 123 {
 124         char *name;
 125         struct symbol *start_sym;
 126         struct smatch_state *state;

 128         *sym = NULL;

 130         name = expr_to_str_sym(expr, &start_sym);
 131         if (!name)
 132                 return NULL;
 133         if (expr->type == EXPR_CALL)
 134                 start_sym = expr_to_sym(expr->fn);

 136         state = get_state(my_id, name, start_sym);
```

```
 137         free_string(name);
 138         if (!state || !state->data)
 139                 return NULL;

 141         *sym = state->data;
 142         return alloc_string(state->name);
 143 }
______unchanged_portion_omitted_

 228 void register_return_to_param(int id)
 229 {
 230         my_id = id;
 231         set_dynamic_states(my_id);
 232         add_modification_hook(my_id, &undef);
 233 }
______unchanged_portion_omitted_
```

```
**********************************************************
    3755 Mon Aug  5 08:38:44 2019
new/usr/src/tools/smatch/src/smatch_returns.c
11506 smatch resync
**********************************************************
______unchanged_portion_omitted_

 129 void register_returns_early(int id)
 130 {
 131         RETURN_ID = id;

 133         set_dynamic_states(RETURN_ID);
 134         add_split_return_callback(match_return);
 135 }
______unchanged_portion_omitted_
```

```
**********************************************************
    1310 Mon Aug  5 08:38:44 2019
new/usr/src/tools/smatch/src/smatch_scripts/build_kernel_data.sh
11506 smatch resync
**********************************************************
______unchanged_portion_omitted_

  13 if [ "$1" = "-h" ] || [ "$1" = "--help" ] ; then
  14         usage;
  15 fi

  17 SCRIPT_DIR=$(dirname $0)
  18 if [ -e $SCRIPT_DIR/../smatch -a -d kernel -a -d fs ] ; then
  19     CMD=$SCRIPT_DIR/../smatch
  20     DATA_DIR=$SCRIPT_DIR/../smatch_data
  21 else
  22     echo "This script should be located in the smatch_scripts/ subdirectory of t
  23     echo "It should be run from the root of a kernel source tree."
  24     exit 1
  25 fi

  27 # If someone is building the database for the first time then make sure all the
  28 # required packages are installed
  29 if [ ! -e smatch_db.sqlite ] ; then
  30     [ -e smatch_warns.txt ] || touch smatch_warns.txt
  31     if ! $DATA_DIR/db/create_db.sh -p=kernel smatch_warns.txt ; then
  32         echo "Hm... Not working.  Make sure you have all the sqlite3 packages"
  33         echo "And the sqlite3 libraries for Perl and Python"
  34         exit 1
  35     fi
  36 fi

  38 BUILD_STATUS=0
  39 $SCRIPT_DIR/test_kernel.sh --call-tree --info --param-mapper --spammy --data=$DA
  38 $SCRIPT_DIR/test_kernel.sh --call-tree --info --param-mapper --spammy --data=$DA

  41 for i in $SCRIPT_DIR/gen_* ; do
  42         $i smatch_warns.txt -p=kernel
  43 done

  45 mv ${PROJECT}.* $DATA_DIR

  47 $DATA_DIR/db/create_db.sh -p=kernel smatch_warns.txt

  49 exit $BUILD_STATUS
```

```
**********************************************************
    2634 Mon Aug  5 08:38:44 2019
new/usr/src/tools/smatch/src/smatch_scripts/kpatch.sh
11506 smatch resync
**********************************************************
   1 #!/bin/bash -e

   3 TMP_DIR=/tmp

   5 help()
   6 {
   7     echo "Usage: $0 [--no-compile|--amend] <filename>"
   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 }
_____unchanged_portion_omitted_

  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" == "--amend" ]] ; then
  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 MSG_FILE=$TMP_DIR/${filename}.msg
  57 MAIL_FILE=$TMP_DIR/${filename}.mail
  56 MAIL_FILE=$TMP_DIR/${filename}.msg

  59 # heat up the disk cache
  60 #git log --oneline $fullname | head -n 10 > /dev/null &

  62 echo "QC checklist"
  63 qc "Have you handled all the errors properly?"
  64 if git diff $fullname | grep ^+ | grep -qi alloc ; then
  65     qc "Have you freed all your mallocs?"
  66 fi
  67 if git diff $fullname | grep ^+ | grep -qi alloc ; then
  68     qc "Have you check all your mallocs for NULL returns?"
  69 fi

  71 if [ "$NO_COMPILE" != "true" ] ; then
  72     kchecker --spammy $fullname
  73     kchecker --sparse --endian $fullname
  74 #    rm $oname
  75 #    make C=1 CHECK="scripts/coccicheck" $oname
  76 fi

  78 for file in $(grep -l $fullname ~/var/mail/sent-*) ; do
```



```
  79     grepmail $fullname $file | grep -i ^subject || echo -n ""
  80 done
  74 grepmail $fullname ~/var/mail/sent* | grep -i ^subject || echo -n ""
  81 qc "Looks OK?"

  77 git log --oneline $fullname | head -n 10
  78 echo "Copy and paste one of these subjects?"
  79 read unused

  83 git add $fullname
  82 git commit --signoff $AMEND

  85 cat /dev/null > $MSG_FILE
  86 if [ "$AMEND" != "" ] ; then
  87     git format-patch HEAD^ --stdout >> $MSG_FILE
  88 else
  89     echo "" >> $MSG_FILE
  90     echo "Signed-off-by: Dan Carpenter <dan.carpenter@oracle.com>" >> $MSG_FILE
  91     echo "" >> $MSG_FILE
  92     echo "# $sm_err" >> $MSG_FILE
  93 fi
  94 git log -10 --oneline $fullname | sed -e 's/^/# /' >> $MSG_FILE
  95 vim $MSG_FILE

  97 grep -v '^#' $MSG_FILE > $MSG_FILE.1
  98 mv $MSG_FILE.1 $MSG_FILE

 100 git commit $AMEND -F $MSG_FILE

 102 to_addr=$(./scripts/get_maintainer.pl -f --noroles --norolestats $fullname | hea
 103 cc_addr=$(./scripts/get_maintainer.pl -f --noroles --norolestats $fullname | tai
 104     perl -ne 's/\n$/, /; print')
 105 cc_addr="$cc_addr, kernel-janitors@vger.kernel.org"

 107 echo -n "To:  "  > $MAIL_FILE
 108 echo "$to_addr" >> $MAIL_FILE
 109 echo -n "CC:  " >> $MAIL_FILE
 110 echo "$cc_addr" >> $MAIL_FILE
 111 echo "X-Mailer: git-send-email haha only kidding" >> $MAIL_FILE

 113 git format-patch HEAD^ --stdout >> $MAIL_FILE

 115 ./scripts/checkpatch.pl $MAIL_FILE || continue_yn

 117 echo "Press ENTER to continue"
 118 read unused

 120 mutt -H $MAIL_FILE
```

****
1996 Mon Aug 5 08:38:45 2019
new/usr/src/tools/smatch/src/smatch_scripts/test_kernel.sh
11506 smatch resync
****
_____unchanged_portion_omitted_

```
 21 while true ; do
 22     if [[ "$1" == "--endian" ]] ; then
 23         ENDIAN="CF=-D__CHECK_ENDIAN__"
 24         shift
 25     elif [[ "$1" == "--target" ]] ; then
 26         shift
 27         TARGET="$1"
 28         shift
 29     elif [[ "$1" == "--log" ]] ; then
 30         shift
 31         LOG="$1"
 32         shift
 33     elif [[ "$1" == "--wlog" ]] ; then
 34         shift
 35         WLOG="$1"
 36         shift
 37     elif [[ "$1" == "--help" ]] ; then
 38         usage
 39     else
 40             break
 41     fi
 42 done

 44 # receive parameters from environment, which override
 45 [ -z "${SMATCH_ENV_TARGET:-}" ] || TARGET="$SMATCH_ENV_TARGET"
 46 [ -z "${SMATCH_ENV_BUILD_PARAM:-}" ] || BUILD_PARAM="$SMATCH_ENV_BUILD_PARAM"

 48 SCRIPT_DIR=$(dirname $0)
 49 if [ -e $SCRIPT_DIR/../smatch ] ; then
 50     cp $SCRIPT_DIR/../smatch $SCRIPT_DIR/../bak.smatch
 51     CMD=$SCRIPT_DIR/../bak.smatch
 52 elif which smatch | grep smatch > /dev/null ; then
 53     CMD=smatch
 54 else
 55     echo "Smatch binary not found."
 56     exit 1
 57 fi

 59 make clean
 60 find -name \*.c.smatch -exec rm \{\} \;
 61 make -j${NR_CPU} $ENDIAN -k CHECK="$CMD -p=kernel --file-output --succeed $*" \
 62         C=1 $BUILD_PARAM $TARGET 2>&1 | tee $LOG
 63 BUILD_STATUS=${PIPESTATUS[0]}
 58         C=1 $TARGET 2>&1 | tee $LOG
 64 find -name \*.c.smatch -exec cat \{\} \; -exec rm \{\} \; > $WLOG
 65 find -name \*.c.smatch.sql -exec cat \{\} \; -exec rm \{\} \; > $WLOG.sql
 66 find -name \*.c.smatch.caller_info -exec cat \{\} \; -exec rm \{\} \; > $WLOG.ca

 68 echo "Done. Build with status $BUILD_STATUS. The warnings are saved to $WLOG"
 69 exit $BUILD_STATUS
 63 echo "Done.  The warnings are saved to $WLOG"
```

```
**********************************************************
   24609 Mon Aug  5 08:38:45 2019
new/usr/src/tools/smatch/src/smatch_slist.c
11506 smatch resync
**********************************************************
_____unchanged_portion_omitted_

/* 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;
        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;
}

int *dynamic_states;
void allocate_dynamic_states_array(int num_checks)
static int cmp_sm_states(const struct sm_state *a, const struct sm_state *b, int
{
        dynamic_states = calloc(num_checks + 1, sizeof(int));
}

void set_dynamic_states(unsigned short owner)
{
        dynamic_states[owner] = true;
}

bool has_dynamic_states(unsigned short owner)
{
        if (owner >= num_checks)
                return false;
        return dynamic_states[owner];
}

static int cmp_possible_sm(const struct sm_state *a, const struct sm_state *b, i
{
        int ret;
```



```
        if (a == b)
                return 0;
        ret = cmp_tracker(a, b);
        if (ret)
                return ret;

        if (!has_dynamic_states(a->owner)) {
        /* todo:  add hook for smatch_extra.c */
                if (a->state > b->state)
                        return -1;
                if (a->state < b->state)
                        return 1;
                return 0;
        }

        if (a->owner == SMATCH_EXTRA) {
                /*
                 * In Smatch extra you can have borrowed implications.
        /* This is obviously a massive disgusting hack but we need to preserve
         * the unmerged states for smatch extra because we use them in
         * smatch_db.c.  Meanwhile if we preserve all the other unmerged states
         * then it uses a lot of memory and we don't use it.  Hence this hack.
                 *
                 * FIXME: review how borrowed implications work and if they
                 * are the best way.  See also smatch_implied.c.
                 *
         * Also sometimes even just preserving every possible SMATCH_EXTRA state
         * takes too much resources so we have to cap that.  Capping is probably
         * not often a problem in real life.
                 */
                ret = cmp_tracker(a, b);
                if (ret)
                        return ret;

                /*
                 * We want to preserve leaf states.  They're use to split
                 * returns in smatch_db.c.
                 *
                 */
                if (preserve) {
                        if (a->merged && !b->merged)
        if (a->owner == SMATCH_EXTRA && preserve) {
                if (a == b)
                        return 0;
                if (a->merged == 1 && b->merged == 0)
                                return -1;
                        if (!a->merged)
                if (a->merged == 0)
                                return 1;
                }
        }
        if (!a->state->name || !b->state->name)
                return 0;

        return strcmp(a->state->name, b->state->name);
        return 0;
}

struct sm_state *alloc_sm_state(int owner, const char *name,
                                struct symbol *sym, struct smatch_state *state)
{
        struct sm_state *sm_state = __alloc_sm_state(0);

        sm_state_counter++;
```

```
192         sm_state->name = alloc_sname(name);
193         sm_state->owner = owner;
194         sm_state->sym = sym;
195         sm_state->state = state;
196         sm_state->line = get_lineno();
197         sm_state->merged = 0;
198         sm_state->pool = NULL;
199         sm_state->left = NULL;
200         sm_state->right = NULL;
172         sm_state->nr_children = 1;
201         sm_state->possible = NULL;
202         add_ptr_list(&sm_state->possible, sm_state);
203         return sm_state;
204 }
```

*unchanged_portion_omitted_*

```
224 void add_possible_sm(struct sm_state *to, struct sm_state *new)
225 {
226         struct sm_state *tmp;
227         int preserve = 1;
228         int cmp;

230         if (too_many_possible(to))
231                 preserve = 0;

233         FOR_EACH_PTR(to->possible, tmp) {
234                 cmp = cmp_possible_sm(tmp, new, preserve);
235                 if (cmp < 0)
205                 if (cmp_sm_states(tmp, new, preserve) < 0)
236                         continue;
237                 else if (cmp == 0) {
207                 else if (cmp_sm_states(tmp, new, preserve) == 0) {
238                         return;
239                 } else {
240                         INSERT_CURRENT(new, tmp);
241                         return;
242                 }
243         } END_FOR_EACH_PTR(tmp);
244         add_ptr_list(&to->possible, new);
245 }

247 static void copy_possibles(struct sm_state *to, struct sm_state *one, struct sm_
217 static void copy_possibles(struct sm_state *to, struct sm_state *from)
248 {
249         struct sm_state *large = one;
250         struct sm_state *small = two;
251         struct sm_state *tmp;

253         /*
254          * We spend a lot of time copying the possible lists.  I've tried to
255          * optimize the process a bit.
256          *
257          */

259         if (ptr_list_size((struct ptr_list *)two->possible) >
260             ptr_list_size((struct ptr_list *)one->possible)) {
261                 large = two;
262                 small = one;
263         }

265         to->possible = clone_slist(large->possible);
266         add_possible_sm(to, to);
267         FOR_EACH_PTR(small->possible, tmp) {
221         FOR_EACH_PTR(from->possible, tmp) {
268                 add_possible_sm(to, tmp);
269         } END_FOR_EACH_PTR(tmp);
```



```
270 }
```

*unchanged_portion_omitted_*

```
283 static struct symbol *oom_func;
284 static int oom_limit = 3000000;  /* Start with a 3GB limit */
285 int out_of_memory(void)
286 {
287         if (oom_func)
288                 return 1;

290         /*
291          * I decided to use 50M here based on trial and error.
292          * It works out OK for the kernel and so it should work
293          * for most other projects as well.
294          */
295         if (sm_state_counter * sizeof(struct sm_state) >= 100000000)
296                 return 1;

298         /*
299          * We're reading from statm to figure out how much memory we
300          * are using.  The problem is that at the end of the function
301          * we release the memory, so that it can be re-used but it
302          * stays in cache, it's not released to the OS.  So then if
303          * we allocate memory for different purposes we can easily
304          * hit the 3GB limit on the next function, so that's why I give
305          * the next function an extra 100MB to work with.
306          *
307          */
308         if (get_mem_kb() > oom_limit) {
309                 oom_func = cur_func_sym;
310                 final_pass++;
311                 sm_perror("OOM: %luKb sm_state_count = %d", get_mem_kb(), sm_sta
312                 final_pass--;
313                 return 1;
314         }

316         return 0;
317 }
```

*unchanged_portion_omitted_*

```
348 /* At the end of every function we free all the sm_states */
349 void free_every_single_sm_state(void)
350 {
351         struct allocator_struct *desc = &sm_state_allocator;
352         struct allocation_blob *blob = desc->blobs;

354         desc->blobs = NULL;
355         desc->allocations = 0;
356         desc->total_bytes = 0;
357         desc->useful_bytes = 0;
358         desc->freelist = NULL;
359         while (blob) {
360                 struct allocation_blob *next = blob->next;
361                 free_all_sm_states(blob);
362                 blob_free(blob, desc->chunking);
363                 blob = next;
364         }
365         clear_sname_alloc();
366         clear_smatch_state_alloc();

368         free_stack_and_strees(&all_pools);
369         sm_state_counter = 0;
370         if (oom_func) {
371                 oom_limit += 100000;
372                 oom_func = NULL;
373         }
```

```
 374 }
_____unchanged_portion_omitted_

 381 struct sm_state *clone_sm(struct sm_state *s)
 382 {
 383         struct sm_state *ret;

 385         ret = alloc_state_no_name(s->owner, s->name, s->sym, s->state);
 386         ret->merged = s->merged;
 387         ret->line = s->line;
 388         /* clone_sm() doesn't copy the pools.  Each state needs to have
 389            only one pool. */
 390         ret->possible = clone_slist(s->possible);
 391         ret->left = s->left;
 392         ret->right = s->right;
 319         ret->nr_children = s->nr_children;
 393         return ret;
 394 }
_____unchanged_portion_omitted_

 450 struct sm_state *merge_sm_states(struct sm_state *one, struct sm_state *two)
 451 {
 452         struct smatch_state *s;
 453         struct sm_state *result;
 454         static int warned;

 456         if (one == two)
 457                 return one;
 458         if (out_of_memory()) {
 459                 if (!warned)
 460                         sm_warning("Function too hairy.  No more merges.");
 461                 warned = 1;
 462                 return one;
 463         }
 464         warned = 0;
 465         s = merge_states(one->owner, one->name, one->sym, one->state, two->state
 466         result = alloc_state_no_name(one->owner, one->name, one->sym, s);
 467         result->merged = 1;
 468         result->left = one;
 469         result->right = two;
 397         result->nr_children = one->nr_children + two->nr_children;
 398         copy_possibles(result, one);
 399         copy_possibles(result, two);

 471         copy_possibles(result, one, two);

 473         /*
 474          * The ->line information is used by deref_check where we complain about
 475          * checking pointers that have already been dereferenced.  Let's say we
 476          * dereference a pointer on both the true and false paths and then merge
 477          * the states here.  The result state is &derefed, but the ->line number
 478          * is on the line where the pointer is merged not where it was
 479          * dereferenced..
 480          *
 481          * So in that case, let's just pick one dereference and set the ->line
 482          * to point at it.
 483          *
 484          */

 486         if (result->state == one->state)
 487                 result->line = one->line;
 488         if (result->state == two->state)
 489                 result->line = two->line;

 491         if (option_debug ||
 492             strcmp(check_name(one->owner), option_debug_check) == 0) {
```



```
 493                 struct sm_state *tmp;
 494                 int i = 0;

 496                 printf("%s:%d %s() merge [%s] '%s' %s(L %d) + %s(L %d) => %s (",
 497                         get_filename(), get_lineno(), get_function(),
 498                         check_name(one->owner), one->name,
 499                         show_state(one->state), one->line,
 500                         show_state(two->state), two->line,
 501                         show_state(s));

 503                 FOR_EACH_PTR(result->possible, tmp) {
 504                         if (i++)
 505                                 printf(", ");
 506                         printf("%s", show_state(tmp->state));
 507                 } END_FOR_EACH_PTR(tmp);
 508                 printf(")\n");
 509         }

 511         return result;
 512 }
_____unchanged_portion_omitted_

 780 int __stree_id;

 782 /*
 783  * merge_slist() is called whenever paths merge, such as after
 784  * an if statement.  It takes the two slists and creates one.
 785  */
 786 static void __merge_stree(struct stree **to, struct stree *stree, int add_pool)
 787 {
 788         struct stree *results = NULL;
 789         struct stree *implied_one = NULL;
 790         struct stree *implied_two = NULL;
 791         AvlIter one_iter;
 792         AvlIter two_iter;
 793         struct sm_state *one, *two, *res;
 721         struct sm_state *tmp_sm;

 795         if (out_of_memory())
 796                 return;

 798         /* merging a null and nonnull path gives you only the nonnull path */
 799         if (!stree)
 800                 return;
 801         if (*to == stree)
 802                 return;

 804         if (!*to) {
 805                 *to = clone_stree(stree);
 806                 return;
 807         }

 809         implied_one = clone_stree(*to);
 810         implied_two = clone_stree(stree);

 812         match_states_stree(&implied_one, &implied_two);
 813         call_pre_merge_hooks(&implied_one, &implied_two);

 815         if (add_pool) {
 816                 clone_pool_havers_stree(&implied_one);
 817                 clone_pool_havers_stree(&implied_two);

 819                 set_stree_id(&implied_one, ++__stree_id);
 820                 set_stree_id(&implied_two, ++__stree_id);
 821                 if (implied_one->base_stree)
 822                         set_stree_id(&implied_one->base_stree, ++__stree_id);
```

```
823                 if (implied_two->base_stree)
824                         set_stree_id(&implied_two->base_stree, ++__stree_id);
825         }

827         push_stree(&all_pools, implied_one);
828         push_stree(&all_pools, implied_two);

830         avl_iter_begin(&one_iter, implied_one, FORWARD);
831         avl_iter_begin(&two_iter, implied_two, FORWARD);

833         for (;;) {
834                 if (!one_iter.sm || !two_iter.sm)
835                         break;

837                 one = one_iter.sm;
838                 two = two_iter.sm;

840                 if (one == two) {
841                         avl_insert(&results, one);
842                         goto next;
843                 }

845                 if (add_pool) {
846                         one->pool = implied_one;
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;
847                         if (implied_one->base_stree)
848                                 one->pool = implied_one->base_stree;
849                         two->pool = implied_two;
771                                         one_iter.sm->pool = implied_one->base_st
772                                 two_iter.sm->pool = implied_two;
850                         if (implied_two->base_stree)
851                                 two->pool = implied_two->base_stree;
774                                         two_iter.sm->pool = implied_two->base_st
852                 }
853                 res = merge_sm_states(one, two);
854                 add_possible_sm(res, one);
855                 add_possible_sm(res, two);
856                 avl_insert(&results, res);
857 next:
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);
858                 avl_iter_next(&one_iter);
859                 avl_iter_next(&two_iter);
782                 } else {
783                         sm_perror(" in %s", __func__);
784                         avl_iter_next(&two_iter);
860         }
786         }

862         free_stree(to);
863         *to = results;
864 }
```

_____***unchanged_portion_omitted_***

```
*******************************************************
    3758 Mon Aug  5 08:38:46 2019
new/usr/src/tools/smatch/src/smatch_slist.h
11506 smatch resync
*******************************************************
_____unchanged_portion_omitted_
```

```
13 DECLARE_ALLOCATOR(named_stree);
14 DECLARE_PTR_LIST(named_stree_stack, struct named_stree);


17 extern struct state_list_stack *implied_pools;
18 extern int __stree_id;
19 extern int sm_state_counter;

21 const char *show_sm(struct sm_state *sm);
22 void __print_stree(struct stree *stree);
23 void add_history(struct sm_state *sm);
24 int cmp_tracker(const struct sm_state *a, const struct sm_state *b);
25 char *alloc_sname(const char *str);
26 struct sm_state *alloc_sm_state(int owner, const char *name,
27                                 struct symbol *sym, struct smatch_state *state);

29 void free_every_single_sm_state(void);
30 struct sm_state *clone_sm(struct sm_state *s);
31 int is_merged(struct sm_state *sm);
32 int is_leaf(struct sm_state *sm);
33 struct state_list *clone_slist(struct state_list *from_slist);

35 int slist_has_state(struct state_list *slist, struct smatch_state *state);

37 int too_many_possible(struct sm_state *sm);
38 void add_possible_sm(struct sm_state *to, struct sm_state *new);
39 struct sm_state *merge_sm_states(struct sm_state *one, struct sm_state *two);
40 struct smatch_state *get_state_stree(struct stree *stree, int owner, const char
41                      struct symbol *sym);

43 struct sm_state *get_sm_state_stree(struct stree *stree, int owner, const char *
44                      struct symbol *sym);

46 void overwrite_sm_state_stree(struct stree **stree, struct sm_state *sm);
47 void overwrite_sm_state_stree_stack(struct stree_stack **stack, struct sm_state
48 struct sm_state *set_state_stree(struct stree **stree, int owner, const char *na
49                       struct symbol *sym, struct smatch_state *state);
50 void set_state_stree_perm(struct stree **stree, int owner, const char *name,
51                      struct symbol *sym, struct smatch_state *state);
52 void delete_state_stree(struct stree **stree, int owner, const char *name,
53                         struct symbol *sym);

55 void delete_state_stree_stack(struct stree_stack **stack, int owner, const char
56                         struct symbol *sym);

58 void push_stree(struct stree_stack **list_stack, struct stree *stree);
59 struct stree *pop_stree(struct stree_stack **list_stack);
60 struct stree *top_stree(struct stree_stack *stack);

62 void free_slist(struct state_list **slist);
63 void free_stree_stack(struct stree_stack **stack);
64 void free_stack_and_strees(struct stree_stack **stree_stack);
65 unsigned long get_pool_count(void);

67 struct sm_state *set_state_stree_stack(struct stree_stack **stack, int owner, co
68                                 struct symbol *sym, struct smatch_state *state);

70 struct sm_state *get_sm_state_stree_stack(struct stree_stack *stack,
71                                 int owner, const char *name,
72                                 struct symbol *sym);
```



```
73 struct smatch_state *get_state_stree_stack(struct stree_stack *stack, int owner,
74                                 const char *name, struct symbol *sym);

76 int out_of_memory(void);
77 int low_on_memory(void);
78 void merge_stree(struct stree **to, struct stree *stree);
79 void merge_stree_no_pools(struct stree **to, struct stree *stree);
80 void merge_stree(struct stree **to, struct stree *right);
81 void merge_fake_stree(struct stree **to, struct stree *stree);
82 void filter_stree(struct stree **stree, struct stree *filter);
83 void and_stree_stack(struct stree_stack **stree_stack);

85 void or_stree_stack(struct stree_stack **pre_conds,
86                     struct stree *cur_stree,
87                     struct stree_stack **stack);

89 struct stree **get_named_stree(struct named_stree_stack *stack,
90                                const char *name,
91                                struct symbol *sym);

93 void overwrite_stree(struct stree *from, struct stree **to);

95 /* add stuff smatch_returns.c here */

97 void all_return_states_hook(void (*callback)(void));

99 void allocate_dynamic_states_array(int num_checks);
```

****
2133 Mon Aug 5 08:38:46 2019
new/usr/src/tools/smatch/src/smatch_statement_count.c
11506 smatch resync
****
_____***unchanged_portion_omitted_***

```
 78 void register_statement_count(int id)
 79 {
 80         my_id = id;

 82         set_dynamic_states(my_id);
 83         add_hook(match_statement, STMT_HOOK);
 84         add_merge_hook(my_id, &merge_states);

 86         add_split_return_callback(&insert_return_info);
 87         select_return_states_hook(STMT_CNT, &select_return_info);
 88 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   25946 Mon Aug  5 08:38:46 2019
new/usr/src/tools/smatch/src/smatch_states.c
11506 smatch resync
**********************************************************
______unchanged_portion_omitted_

 83 struct sm_state *set_state(int owner, const char *name, struct symbol *sym, stru
 84 {
 85         struct sm_state *ret;

 87         if (!name || !state)
 87                 if (!name)
 88                         return NULL;

 90         if (read_only)
 91                 sm_perror("cur_stree is read only.");

 93         if (option_debug || strcmp(check_name(owner), option_debug_check) == 0)
 94                 struct smatch_state *s;

 96                 s = __get_state(owner, name, sym);
 96                 s = get_state(owner, name, sym);
 97                 if (!s)
 98                         sm_msg("%s new [%s] '%s' %s", __func__,
 99                                check_name(owner), name, show_state(state));
100                 else
101                         sm_msg("%s change [%s] '%s' %s => %s",
102                                __func__, check_name(owner), name, show_state(s)
103                                show_state(state));
104         }

106         if (owner != -1 && unreachable())
107                 return NULL;

109         if (fake_cur_stree_stack)
110                 set_state_stree_stack(&fake_cur_stree_stack, owner, name, sym, s

112         ret = set_state_stree(&cur_stree, owner, name, sym, state);

114         return ret;
115 }
______unchanged_portion_omitted_

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);
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);
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);
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 }
______unchanged_portion_omitted_

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);
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 }
______unchanged_portion_omitted_

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);
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 }
______unchanged_portion_omitted_

788 void __negate_cond_stacks(void)
789 {
790         struct stree *old_false, *old_true;

792         __use_cond_stack(&cond_false_stack);
792         old_false = pop_stree(&cond_false_stack);
793         old_true = pop_stree(&cond_true_stack);
794         push_stree(&cond_false_stack, old_true);
```



```
795         push_stree(&cond_true_stack, old_false);
796 }
______unchanged_portion_omitted_
```

```
**********************************************************
    7241 Mon Aug  5 08:38:47 2019
new/usr/src/tools/smatch/src/smatch_stored_conditions.c
11506 smatch resync
**********************************************************
______unchanged_portion_omitted_

 238 void register_stored_conditions(int id)
 239 {
 240         my_id = id;
 241         set_dynamic_states(my_id);
 242 }

 244 void register_stored_conditions_links(int id)
 245 {
 246         link_id = id;
 247         db_ignore_states(link_id);
 248         set_dynamic_states(link_id);
 249         add_merge_hook(link_id, &merge_links);
 250         add_modification_hook(link_id, &match_link_modify);
 251 }
______unchanged_portion_omitted_
```

```
**********************************************************
    1863 Mon Aug  5 08:38:47 2019
new/usr/src/tools/smatch/src/smatch_string_list.c
11506 smatch resync
**********************************************************
```

```
 1 /*
 2  * Copyright (C) 2013 Oracle.
 3  *
 4  * This program is free software; you can redistribute it and/or
 5  * modify it under the terms of the GNU General Public License
 6  * as published by the Free Software Foundation; either version 2
 7  * of the License, or (at your option) any later version.
 8  *
 9  * This program is distributed in the hope that it will be useful,
10  * but WITHOUT ANY WARRANTY; without even the implied warranty of
11  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
12  * GNU General Public License for more details.
13  *
14  * You should have received a copy of the GNU General Public License
15  * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
16  */

18 #include "smatch.h"

20 int list_has_string(struct string_list *str_list, const char *str)
21 {
22         char *tmp;
23         int cmp;

25         if (!str)
26                 return 0;

28         FOR_EACH_PTR(str_list, tmp) {
29                 cmp = strcmp(tmp, str);
30                 if (cmp < 0)
28                 if (strcmp(tmp, str) < 0)
31                         continue;
32                 if (cmp == 0)
30                 if (strcmp(tmp, str) == 0)
33                         return 1;
34                 return 0;
35         } END_FOR_EACH_PTR(tmp);
36         return 0;
37 }

39 int insert_string(struct string_list **str_list, const char *_new)
37 void insert_string(struct string_list **str_list, const char *_new)
40 {
41         char *new = (char *)_new;
42         char *tmp;
43         int cmp;

45         FOR_EACH_PTR(*str_list, tmp) {
46                 cmp = strcmp(tmp, new);
47                 if (cmp < 0)
43                 if (strcmp(tmp, new) < 0)
48                         continue;
49                 else if (cmp == 0) {
50                         return 0;
45                 else if (strcmp(tmp, new) == 0) {
46                         return;
51                 } else {
52                         INSERT_CURRENT(alloc_string(new), tmp);
53                         return 1;
49                         return;
54                 }
```



```
55         } END_FOR_EACH_PTR(tmp);
56         new = alloc_string(new);
57         add_ptr_list(str_list, new);
58         return 1;
59 }
______unchanged_portion_omitted_
```

****
8938 Mon Aug 5 08:38:47 2019
new/usr/src/tools/smatch/src/smatch_strlen.c
11506 smatch resync
****
_____***unchanged_portion_omitted_***

```
 332 void register_strlen(int id)
 333 {
 334         my_strlen_id = id;

 336         set_dynamic_states(my_strlen_id);

 338         add_unmatched_state_hook(my_strlen_id, &unmatched_strlen_state);

 340         select_caller_info_hook(set_param_strlen, STR_LEN);
 341         add_hook(&match_string_assignment, ASSIGNMENT_HOOK);

 343         add_modification_hook(my_strlen_id, &set_strlen_undefined);
 344         add_merge_hook(my_strlen_id, &merge_estates);
 345         add_hook(&match_call, FUNCTION_CALL_HOOK);
 346         add_member_info_callback(my_strlen_id, struct_member_callback);
 347         add_hook(&match_strlen_condition, CONDITION_HOOK);

 349         add_function_hook("snprintf", &match_snprintf, NULL);

 351         add_function_hook("strlcpy", &match_strlcpycat, NULL);
 352         add_function_hook("strlcat", &match_strlcpycat, NULL);
 353         add_function_hook("strcpy", &match_strcpy, NULL);
 354 }

 356 void register_strlen_equiv(int id)
 357 {
 358         my_equiv_id = id;
 359         set_dynamic_states(my_equiv_id);
 360         add_function_assign_hook("strlen", &match_strlen, NULL);
 361         add_modification_hook(my_equiv_id, &set_strlen_equiv_undefined);
 362 }
```

_____***unchanged_portion_omitted_***

****
13960 Mon Aug 5 08:38:48 2019
new/usr/src/tools/smatch/src/smatch_struct_assignment.c
11506 smatch resync
****
_____unchanged_portion_omitted_

```
448 static void match_memdup(const char *fn, struct expression *call_expr,
449                           struct expression *expr, void *_unused)
450 {
451         struct expression *left, *right, *arg;
452
453         if (!expr || expr->type != EXPR_ASSIGNMENT)
454                 return;
455
456         left = strip_expr(expr->left);
457         right = strip_expr(expr->right);
458
459         if (right->type != EXPR_CALL)
460                 return;
461         arg = get_argument_from_call_expr(right->args, 0);
462         __struct_members_copy(COPY_MEMCPY, expr, left, arg);
463 }
464
465 static void match_memcpy_unknown(const char *fn, struct expression *expr, void *
466 {
467         struct expression *dest;
468
469         dest = get_argument_from_call_expr(expr->args, 0);
470         __struct_members_copy(COPY_MEMCPY, expr, remove_addr(dest), NULL);
471 }
```

_____unchanged_portion_omitted_

```
558 void register_struct_assignment(int id)
559 {
560         add_function_hook("memset", &match_memset, NULL);
561         add_function_hook("__memset", &match_memset, NULL);
562
563         add_function_hook("memcpy", &match_memcpy, INT_PTR(0));
564         add_function_hook("memmove", &match_memcpy, INT_PTR(0));
565         add_function_hook("__memcpy", &match_memcpy, INT_PTR(0));
566         add_function_hook("__memmove", &match_memcpy, INT_PTR(0));
567
568         if (option_project == PROJ_KERNEL)
569                 return_implies_state_sval("kmemdup", valid_ptr_min_sval, valid_p
570
571         add_function_hook("sscanf", &match_sscanf, NULL);
572
573         add_hook(&unop_expr, OP_HOOK);
574         register_clears_param();
575         select_return_states_hook(PARAM_CLEARED, &db_param_cleared);
576
577         select_return_states_hook(CONTAINER, &returns_container_of);
578 }
```

_____unchanged_portion_omitted_

```
**********************************************************
   14551 Mon Aug  5 08:38:48 2019
new/usr/src/tools/smatch/src/smatch_sval.c
11506 smatch resync
**********************************************************
______unchanged_portion_omitted_

  65 sval_t sval_type_val(struct symbol *type, long long val)
  66 {
  67         sval_t ret;

  69         if (!type)
  70                 type = &llong_ctype;
  70                 type = &int_ctype;

  72         ret.type = type;
  73         ret.value = val;
  74         return ret;
  75 }
______unchanged_portion_omitted_

  95 int sval_unsigned(sval_t sval)
  96 {
  97         if (is_ptr_type(sval.type))
  98                 return true;
  99         return type_unsigned(sval.type);
 100 }
______unchanged_portion_omitted_

 431 static sval_t ptr_binop(struct symbol *type, sval_t left, int op, sval_t right)
 432 {
 433         sval_t ret;
 434         int align;

 436         if (op != '+' && op != '-')
 437                 return sval_binop_unsigned(type, left, op, right);

 439         ret.type = type;
 440         if (type->type == SYM_PTR)
 441                 type = get_real_base_type(type);
 442         align = type->ctype.alignment;
 443         if (align <= 0)
 444                 align = 1;

 446         if (op == '+') {
 447                 if (type_is_ptr(left.type))
 448                         ret.value = left.value + right.value * align;
 449                 else
 450                         ret.value = left.value * align + right.value;
 451         } else {
 452                 if (!type_is_ptr(left.type)) {
 453                         left.value = -left.value;
 454                         ret = ptr_binop(type, left, '+', right);
 455                 } else if (!type_is_ptr(right.type)) {
 456                         right.value = -right.value;
 457                         ret = ptr_binop(type, left, '+', right);
 458                 } else {
 459                         ret.value = (left.value - right.value) / align;
 460                 }
 461         }

 463         if (op == '-')
 464                 ret.type = ssize_t_ctype;
 465         return ret;
 466 }
______unchanged_portion_omitted_
```



```
 590 int find_first_zero_bit(unsigned long long uvalue)
 586 unsigned long long fls_mask(unsigned long long uvalue)
 591 {
 592         int i;
 588         unsigned long long high_bit = 0;

 594         for (i = 0; i < 64; i++) {
 595                 if (!(uvalue & (1ULL << i)))
 596                         return i;
 597         }
 598         return i;
 599 }

 601 int sm_fls64(unsigned long long uvalue)
 602 {
 603         int high_bit = 0;

 605         while (uvalue) {
 606                 uvalue >>= 1;
 607                 high_bit++;
 608         }

 610         return high_bit;
 611 }

 613 unsigned long long fls_mask(unsigned long long uvalue)
 614 {
 615         int high_bit = 0;

 617         high_bit = sm_fls64(uvalue);
 618         if (high_bit == 0)
 619                 return 0;

 621         return ((unsigned long long)-1) >> (64 - high_bit);
 622 }
______unchanged_portion_omitted_

 629 const char *sval_to_str(sval_t sval)
 630 {
 631         char buf[30];

 633         if (sval_is_ptr(sval) && sval.value == valid_ptr_max)
 634                 return "ptr_max";
 635         if (sval_unsigned(sval) && sval.value == ULLONG_MAX)
 636                 return "u64max";
 637         if (sval_unsigned(sval) && sval.value == UINT_MAX)
 638                 return "u32max";
 639         if (sval.value == USHRT_MAX)
 640                 return "u16max";

 642         if (sval_signed(sval) && sval.value == LLONG_MAX)
 643                 return "s64max";
 644         if (sval.value == INT_MAX)
 645                 return "s32max";
 646         if (sval.value == SHRT_MAX)
 647                 return "s16max";

 649         if (sval_signed(sval) && sval.value == SHRT_MIN)
 650                 return "s16min";
 651         if (sval_signed(sval) && sval.value == INT_MIN)
 652                 return "s32min";
 653         if (sval_signed(sval) && sval.value == LLONG_MIN)
 654                 return "s64min";

 656         if (sval_unsigned(sval))
```

```
657                 snprintf(buf, sizeof(buf), "%llu", sval.value);
658         else if (sval.value < 0)
659                 snprintf(buf, sizeof(buf), "(%lld)", sval.value);
660         else
661                 snprintf(buf, sizeof(buf), "%lld", sval.value);

663         return alloc_sname(buf);
664 }

666 const char *sval_to_str_or_err_ptr(sval_t sval)
667 {
668         char buf[12];

670         if (option_project != PROJ_KERNEL ||
671             !is_ptr_type(sval.type))
672                 return sval_to_str(sval);

674         if (sval.uvalue >= -4905ULL) {
675                 snprintf(buf, sizeof(buf), "(%lld)", sval.value);
676                 return alloc_sname(buf);
677         }

679         return sval_to_str(sval);
680 }

682 const char *sval_to_numstr(sval_t sval)
683 {
684         char buf[30];

686         if (sval_unsigned(sval))
687                 snprintf(buf, sizeof(buf), "%llu", sval.value);
688         else if (sval.value < 0)
689                 snprintf(buf, sizeof(buf), "(%lld)", sval.value);
690         else
691                 snprintf(buf, sizeof(buf), "%lld", sval.value);

693         return alloc_sname(buf);
694 }
```

*unchanged_portion_omitted_*

```
**********************************************************
   16744 Mon Aug  5 08:38:49 2019
new/usr/src/tools/smatch/src/smatch_type.c
11506 smatch resync
**********************************************************
```

```
   1 /*
   2  * Copyright (C) 2009 Dan Carpenter.
   3  *
   4  * This program is free software; you can redistribute it and/or
   5  * modify it under the terms of the GNU General Public License
   6  * as published by the Free Software Foundation; either version 2
   7  * of the License, or (at your option) any later version.
   8  *
   9  * This program is distributed in the hope that it will be useful,
  10  * but WITHOUT ANY WARRANTY; without even the implied warranty of
  11  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  12  * GNU General Public License for more details.
  13  *
  14  * You should have received a copy of the GNU General Public License
  15  * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
  16  */

  18 /*
  19  * The idea here is that you have an expression and you
  20  * want to know what the type is for that.
  21  */

  23 #include "smatch.h"
  24 #include "smatch_slist.h"

  26 struct symbol *get_real_base_type(struct symbol *sym)
  27 {
  28         struct symbol *ret;

  30         if (!sym)
  31                 return NULL;
  32         if (sym->type == SYM_BASETYPE)
  33                 return sym;
  34         ret = get_base_type(sym);
  35         if (!ret)
  36                 return NULL;
  37         if (ret->type == SYM_RESTRICT || ret->type == SYM_NODE)
  38                 return get_real_base_type(ret);
  39         return ret;
  40 }
```

*unchanged_portion_omitted_*

```
  62 static struct symbol *get_binop_type(struct expression *expr)
  63 {
  64         struct symbol *left, *right;

  66         left = get_type(expr->left);
  67         if (!left)
  68                 return NULL;

  70         if (expr->op == SPECIAL_LEFTSHIFT ||
  71             expr->op == SPECIAL_RIGHTSHIFT) {
  72                 if (type_positive_bits(left) < 31)
  73                         return &int_ctype;
  74                 return left;
  75         }
  76         right = get_type(expr->right);
  77         if (!right)
  78                 return NULL;

  80         if (expr->op == '-' &&
```



```
  81             (is_ptr_type(left) && is_ptr_type(right)))
  82                 return ssize_t_ctype;

  84         if (left->type == SYM_PTR || left->type == SYM_ARRAY)
  85                 return left;
  86         if (right->type == SYM_PTR || right->type == SYM_ARRAY)
  87                 return right;

  89         if (type_positive_bits(left) < 31 && type_positive_bits(right) < 31)
  90                 return &int_ctype;

  92         if (type_positive_bits(left) > type_positive_bits(right))
  93                 return left;
  94         return right;
  95 }
```

*unchanged_portion_omitted_*

```
 220 static struct symbol *get_type_helper(struct expression *expr)
 221 {
 222         struct symbol *ret;

 224         expr = strip_parens(expr);
 225         if (!expr)
 226                 return NULL;

 228         if (expr->ctype)
 229                 return expr->ctype;

 231         switch (expr->type) {
 232         case EXPR_STRING:
 233                 ret = &string_ctype;
 234                 break;
 235         case EXPR_SYMBOL:
 236                 ret = get_type_symbol(expr);
 237                 break;
 238         case EXPR_DEREF:
 239                 ret = get_symbol_from_deref(expr);
 240                 break;
 241         case EXPR_PREOP:
 242         case EXPR_POSTOP:
 243                 if (expr->op == '&')
 244                         ret = fake_pointer_sym(expr);
 245                 else if (expr->op == '*')
 246                         ret = get_pointer_type(expr->unop);
 247                 else
 248                         ret = get_type(expr->unop);
 249                 break;
 250         case EXPR_ASSIGNMENT:
 251                 ret = get_type(expr->left);
 252                 break;
 253         case EXPR_CAST:
 254         case EXPR_FORCE_CAST:
 255         case EXPR_IMPLIED_CAST:
 256                 ret = get_real_base_type(expr->cast_type);
 257                 break;
 258         case EXPR_COMPARE:
 259         case EXPR_BINOP:
 260                 ret = get_binop_type(expr);
 261                 break;
 262         case EXPR_CALL:
 263                 ret = get_return_type(expr);
 264                 break;
 265         case EXPR_STATEMENT:
 266                 ret = get_expr_stmt_type(expr->statement);
 267                 break;
 268         case EXPR_CONDITIONAL:
```

```
        case EXPR_SELECT:
                ret = get_select_type(expr);
                break;
        case EXPR_SIZEOF:
                ret = &ulong_ctype;
                break;
        case EXPR_LOGICAL:
                ret = &int_ctype;
                break;
        case EXPR_OFFSETOF:
                ret = &ulong_ctype;
                break;
        default:
                return NULL;
        }

        if (ret && ret->type == SYM_TYPEOF)
                ret = get_type(ret->initializer);

        expr->ctype = ret;
        return ret;
}

static struct symbol *get_final_type_helper(struct expression *expr)
{
        /*
         * The problem is that I wrote a bunch of Smatch to think that
         * you could do get_type() on an expression and it would give
         * you what the comparison was type promoted to.  This is wrong
         * but fixing it is a big of work...  Hence this horrible hack.
         * I'm not totally positive I understand types...
         *
         * So, when you're doing pointer math, and you do a subtraction, then
         * the sval_binop() and whatever need to know the type of the pointer
         * so they can figure out the alignment.  But the result is going to be
         * and ssize_t.  So get_operation_type() gives you the pointer type
         * and get_type() gives you ssize_t.
         *
         * Most of the time the operation type and the final type are the same
         * but this just handles the few places where they are different.
         *
         */

        expr = strip_parens(expr);
        if (!expr)
                return NULL;

        if (expr->type == EXPR_COMPARE)
        switch (expr->type) {
        case EXPR_COMPARE:
                return &int_ctype;
        case EXPR_BINOP: {
                struct symbol *left, *right;

                if (expr->op != '-')
        return NULL;

                left = get_type(expr->left);
                right = get_type(expr->right);
                if (type_is_ptr(left) || type_is_ptr(right))
                        return ssize_t_ctype;
                }
        }

        return NULL;
}
```

_____*unchanged_portion_omitted_*


```
int is_pointer(struct expression *expr)
{
        return type_is_ptr(get_type(expr));
        struct symbol *sym;

        sym = get_type(expr);
        if (!sym)
                return 0;
        if (sym == &string_ctype)
                return 0;
        if (sym->type == SYM_PTR)
                return 1;
        return 0;
}
```

_____*unchanged_portion_omitted_*

```
sval_t sval_type_min(struct symbol *base_type)
{
        sval_t ret;

        if (!base_type || !type_bits(base_type))
                base_type = &llong_ctype;
        ret.type = base_type;

        if (type_unsigned(base_type) || is_ptr_type(base_type)) {
        if (type_unsigned(base_type)) {
                ret.value = 0;
                return ret;
        }

        ret.value = (~0ULL) << type_positive_bits(base_type);

        return ret;
}
```

_____*unchanged_portion_omitted_*

```
static struct symbol *get_member_from_string(struct symbol_list *symbol_list, co
{
        struct symbol *tmp, *sub;
        int chunk_len;

        if (strncmp(name, ".", 1) == 0)
                name += 1;
        else if (strncmp(name, "->", 2) == 0)
        if (strncmp(name, "->", 2) == 0)
                name += 2;

        FOR_EACH_PTR(symbol_list, tmp) {
                if (!tmp->ident) {
                        sub = get_real_base_type(tmp);
                        sub = get_member_from_string(sub->symbol_list, name);
                        if (sub)
                                return sub;
                        continue;
                }

                if (strcmp(tmp->ident->name, name) == 0)
                        return tmp;

                chunk_len = tmp->ident->len;
                chunk_len = strlen(tmp->ident->name);
                if (strncmp(tmp->ident->name, name, chunk_len) == 0 &&
                    (name[chunk_len] == '.' || name[chunk_len] == '-')) {
                        sub = get_real_base_type(tmp);
                        if (sub->type == SYM_PTR)
```

```
608                                 sub = get_real_base_type(sub);
609                         return get_member_from_string(sub->symbol_list, name + c
610                 }

612         } END_FOR_EACH_PTR(tmp);

614         return NULL;
615 }
```

*unchanged_portion_omitted_*

```
719 static int type_str_helper(char *buf, int size, struct symbol *type)
720 {
721         int n;

723         if (!type)
724                 return snprintf(buf, size, "<unknown>");

726         if (type->type == SYM_BASETYPE) {
727                 return snprintf(buf, size, "%s", base_type_str(type));
744                 return snprintf(buf, size, base_type_str(type));
728         } else if (type->type == SYM_PTR) {
729                 type = get_real_base_type(type);
730                 n = type_str_helper(buf, size, type);
731                 if (n > size)
732                         return n;
733                 return n + snprintf(buf + n, size - n, "*");
734         } else if (type->type == SYM_ARRAY) {
735                 type = get_real_base_type(type);
736                 n = type_str_helper(buf, size, type);
737                 if (n > size)
738                         return n;
739                 return n + snprintf(buf + n, size - n, "[]");
740         } else if (type->type == SYM_STRUCT) {
741                 return snprintf(buf, size, "struct %s", type->ident ? type->iden
742         } else if (type->type == SYM_UNION) {
743                 if (type->ident)
744                         return snprintf(buf, size, "union %s", type->ident->name
745                 else
746                         return snprintf(buf, size, "anonymous union");
747         } else if (type->type == SYM_FN) {
748                 struct symbol *arg, *return_type, *arg_type;
749                 int i;

751                 return_type = get_real_base_type(type);
752                 n = type_str_helper(buf, size, return_type);
753                 if (n > size)
754                         return n;
755                 n += snprintf(buf + n, size - n, "(*)(");
756                 if (n > size)
757                         return n;

759                 i = 0;
760                 FOR_EACH_PTR(type->arguments, arg) {
761                         if (i++)
762                                 n += snprintf(buf + n, size - n, ", ");
763                         if (n > size)
764                                 return n;
765                         arg_type = get_real_base_type(arg);
766                         n += type_str_helper(buf + n, size - n, arg_type);
767                         if (n > size)
768                                 return n;
769                 } END_FOR_EACH_PTR(arg);

771                 return n + snprintf(buf + n, size - n, ")");
772         } else if (type->type == SYM_NODE) {
773                 n = snprintf(buf, size, "node {");
```



```
774                 if (n > size)
775                         return n;
776                 type = get_real_base_type(type);
777                 n += type_str_helper(buf + n, size - n, type);
778                 if (n > size)
779                         return n;
780                 return n + snprintf(buf + n, size - n, "}");
781         } else if (type->type == SYM_ENUM) {
782                 return snprintf(buf, size, "enum %s", type->ident ? type->ident-
783         } else {
784                 return snprintf(buf, size, "<type %d>", type->type);
785         }
786 }
```

*unchanged_portion_omitted_*

****
14152 Mon Aug 5 08:38:49 2019
new/usr/src/tools/smatch/src/smatch_type_val.c
11506 smatch resync
****
_____***unchanged_portion_omitted_***

```
 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;

 399         member = get_member_name(expr->left);
 400         if (!member)
 401                 return;

 403         /* if we're saying foo->mtu = bar->mtu then that doesn't add information
 404         right_member = get_member_name(expr->right);
 405         if (right_member && strcmp(right_member, member) == 0)
 406                 goto free;

 408         if (is_fake_call(expr->right)) {
 409                 if (is_ignored_macro())
 410                         goto free;
 411                 if (is_ignored_function())
 412                         goto free;
 413                 if (is_uncasted_pointer_assign())
 414                         goto free;
 415                 if (is_uncasted_fn_param_from_db())
 416                         goto free;
 417                 if (is_container_of())
 418                         goto free;
 419                 add_fake_type_val(member, alloc_whole_rl(get_type(expr->left)),
 420                 goto free;
 421         }

 423         if (expr->op == '=') {
 424                 get_absolute_rl(expr->right, &rl);
 425                 rl = cast_rl(type, rl);
 426         } else {
 427                 /*
 428                  * This is a bit cheating.  We order it so this will already be
 429                  * by smatch_extra.c and we just look up the value.
 430                  */
 431                 get_absolute_rl(expr->left, &rl);
 432         }
 433         add_type_val(member, rl);
 434 free:
 435         free_string(right_member);
 436         free_string(member);
 437 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    7570 Mon Aug  5 08:38:49 2019
new/usr/src/tools/smatch/src/smatch_untracked_param.c
11506 smatch resync
**********************************************************
```

```
/*
 * 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);
STATE(lost);

typedef void (untracked_hook)(struct expression *call, int param);
DECLARE_PTR_LIST(untracked_hook_list, untracked_hook *);
static struct untracked_hook_list *untracked_hooks;
static struct untracked_hook_list *lost_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);
}
```

*unchanged_portion_omitted_*

```
void add_lost_param_hook(void (func)(struct expression *call, int param))
{
        untracked_hook **p = malloc(sizeof(untracked_hook *));
```



```
        *p = func;
        add_ptr_list(&lost_hooks, p);
}

static void call_lost_callbacks(struct expression *expr, int param)
{
        untracked_hook **fn;

        FOR_EACH_PTR(lost_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;
}

static char *get_array_from_key(struct expression *expr, int param, const char *
void mark_untracked(struct expression *expr, int param, const char *key, const c
{
        struct expression *arg;

        arg = get_argument_from_call_expr(expr->args, param);
        if (!arg)
                return NULL;
        if (arg->type != EXPR_PREOP || arg->op != '&')
                return NULL;
        arg = arg->unop;
        if (!is_array(arg))
                return NULL;
        arg = get_array_base(arg);

        return expr_to_var_sym(arg, sym);
}

static void mark_untracked_lost(struct expression *expr, int param, const char *
{
        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) {
                name = get_array_from_key(expr, param, key, &sym);
                if (!name || !sym)
                        goto free;
        }

        if (type == LOST_PARAM)
                call_lost_callbacks(expr, param);
        call_untracked_callbacks(expr, param);
        set_state(my_id, name, sym, &untracked);
free:
        free_string(name);

}

void mark_untracked(struct expression *expr, int param, const char *key, const c
{
        mark_untracked_lost(expr, param, key, UNTRACKED_PARAM);
}
```

```
136 void mark_lost(struct expression *expr, int param, const char *key, const char *
137 {
138         mark_untracked_lost(expr, param, key, LOST_PARAM);
139 }

141 static int lost_in_va_args(struct expression *expr)
142 {
143         struct symbol *fn;
144         char *name;
145         int is_lost;

147         fn = get_type(expr->fn);
148         if (!fn || !fn->variadic)
149                 return 0;

151         is_lost = 1;
152         name = expr_to_var(expr->fn);
153         if (name && strstr(name, "print"))
154                 is_lost = 0;
155         free_string(name);

157         return is_lost;
158 }
```

*unchanged_portion_omitted_*

```
188 static void mark_all_params(int return_id, char *return_ranges, int type)
136 void mark_all_params_untracked(int return_id, char *return_ranges, struct expres
189 {
190         struct symbol *arg;
191         int param;

193         param = -1;
194         FOR_EACH_PTR(cur_func_sym->ctype.base_type->arguments, arg) {
195                 param++;

197                 if (!arg->ident)
198                         continue;
199                 sql_insert_return_states(return_id, return_ranges,
200                                          type, param, "$", "");
148                                          UNTRACKED_PARAM, param, "$", "");
201         } END_FOR_EACH_PTR(arg);
202 }

205 void mark_all_params_untracked(int return_id, char *return_ranges, struct expres
206 {
207         mark_all_params(return_id, return_ranges, UNTRACKED_PARAM);
208 }

210 void mark_all_params_lost(int return_id, char *return_ranges, struct expression
211 {
212         mark_all_params(return_id, return_ranges, LOST_PARAM);
213 }

215 static void print_untracked_params(int return_id, char *return_ranges, struct ex
216 {
217         struct sm_state *sm;
218         struct symbol *arg;
219         int param;
220         int type;

222         param = -1;
223         FOR_EACH_PTR(cur_func_sym->ctype.base_type->arguments, arg) {
224                 param++;
```



```
226                 if (!arg->ident)
227                         continue;

229                 if (__bail_on_rest_of_function) {
230                         /* hairy functions are lost */
231                         type = LOST_PARAM;
232                 } else if ((sm = get_sm_state(my_id, arg->ident->name, arg))) {
233                         if (slist_has_state(sm->possible, &lost))
234                                 type = LOST_PARAM;
235                         else
236                                 type = UNTRACKED_PARAM;
237                 } else {
163                 if (!get_state(my_id, arg->ident->name, arg) &&
164                     !__bail_on_rest_of_function)  /* hairy functions are untrack
238                         continue;
239                 }

241                 sql_insert_return_states(return_id, return_ranges,
242                                          type, param, "$", "");
168                                          UNTRACKED_PARAM, param, "$", "");
243         } END_FOR_EACH_PTR(arg);
244 }
```

*unchanged_portion_omitted_*

```
308 void register_untracked_param(int id)
309 {
310         my_id = id;

312         select_return_states_hook(INTERNAL, &assume_tracked);
313         select_return_states_hook(UNTRACKED_PARAM, &mark_untracked);
314         select_return_states_hook(LOST_PARAM, &mark_lost);
315         add_hook(&match_after_call, FUNCTION_CALL_HOOK_AFTER_DB);

317         add_split_return_callback(&print_untracked_params);

319         add_hook(&match_param_assign, ASSIGNMENT_HOOK);
320         add_hook(&match_param_assign_in_asm, ASM_HOOK);

322         add_hook(&match_inline_start, INLINE_FN_START);
323         add_hook(&match_inline_end, INLINE_FN_END);
324 }
```

*unchanged_portion_omitted_*

```
**********************************************************
     533 Mon Aug  5 08:38:50 2019
new/usr/src/tools/smatch/src/validation/sm_bitwise1.c
11506 smatch resync
**********************************************************
______unchanged_portion_omitted_

 12 /*
 13  * check-name: smatch bitwise #1
 14  * check-command: smatch -I.. sm_bitwise1.c
 15  *
 16  * check-output-start
 17 sm_bitwise1.c:6 test() implied: x & 1 = '0-1'
 18 sm_bitwise1.c:7 test() implied: x & 2 = '0,2'
 19 sm_bitwise1.c:8 test() implied: x & ~(255) = '0,256-4294967040'
 20 sm_bitwise1.c:9 test() implied: x & ~(255) = '0,256-4294967040'
 20 sm_bitwise1.c:9 test() implied: x & ~(255) = '0-4294967040'
 21  * check-output-end
 22  */
```

```
**********************************************************
     777 Mon Aug  5 08:38:50 2019
new/usr/src/tools/smatch/src/validation/sm_equiv1.c
11506 smatch resync
**********************************************************
_____unchanged_portion_omitted_
  26 /*
  27  * check-name: smatch equivalent variables #1
  28  * check-command: smatch -I.. -m64 sm_equiv1.c
  29  *
  30  * check-output-start
  31 sm_equiv1.c:13 func() one = 1
  32 sm_equiv1.c:14 func() two = 1
  33 sm_equiv1.c:16 func() one = 0-u64max
  34 sm_equiv1.c:17 func() two = 0-u64max
  33 sm_equiv1.c:16 func() one = s64min-s64max
  34 sm_equiv1.c:17 func() two = s64min-s64max
  35 sm_equiv1.c:19 func() one = 2
  36 sm_equiv1.c:20 func() two = 2
  37 sm_equiv1.c:22 func() one = 0-u64max
  38 sm_equiv1.c:23 func() two = 0-u64max
  37 sm_equiv1.c:22 func() one = s64min-s64max
  38 sm_equiv1.c:23 func() two = s64min-s64max
  39  * check-output-end
  40  */
```

```
**********************************************************
     487 Mon Aug  5 08:38:50 2019
new/usr/src/tools/smatch/src/validation/sm_implied.c
11506 smatch resync
**********************************************************
_____unchanged_portion_omitted_
  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 sm_implied.c:20 func() error: potentially dereferencing uninitialized 'aa'.
  30  * check-output-end
  31  */
```

****
672 Mon Aug 5 08:38:51 2019
new/usr/src/tools/smatch/src/validation/sm_implied10.c
11506 smatch resync
****

```
 1 #include "check_debug.h"

 3 void frob(void){}

 5 int x[10];
 6 int offset;
 7 void func(int *y)
 8 {
 9         if (({int test2 = !!(!y || !*y); frob(); frob(); frob(); test2;}))
10                 __smatch_value("y");
11         else
12                 __smatch_value("y");

14         if (({int test2 = !!(offset >= 10u || x[offset] == 1); frob(); frob(); f
14         if (({int test2 = !!(offset >= 10 || x[offset] == 1); frob(); frob(); fr
15                 __smatch_value("offset");
16         else
17                 __smatch_value("offset");

19 }
20 /*
21  * check-name: smatch implied #10
22  * check-command: smatch -I.. -m64 sm_implied10.c
23  *
24  * check-output-start
25 sm_implied10.c:10 func() y = 0,4096-ptr_max
26 sm_implied10.c:12 func() y = 4096-ptr_max
27 sm_implied10.c:15 func() offset = s32min-s32max
25 sm_implied10.c:10 func() y = 0,4096-2117777777777777777
26 sm_implied10.c:12 func() y = 4096-2117777777777777777
27 sm_implied10.c:15 func() offset = 0-s32max
28 sm_implied10.c:17 func() offset = 0-9
29  * check-output-end
30  */
```

```
**********************************************************
     456 Mon Aug  5 08:38:51 2019
new/usr/src/tools/smatch/src/validation/sm_implied11.c
11506 smatch resync
**********************************************************
_____unchanged_portion_omitted_
  27 /*
  28  * check-name: smatch implied #11
  29  * check-command: smatch -I.. -m64 sm_implied11.c
  30  *
  31  * check-output-start
  32 sm_implied11.c:25 ad_agg_selection_logic() implied: foo = '0,4096-ptr_max'
  32 sm_implied11.c:25 ad_agg_selection_logic() implied: foo = '0,4096-21177777777777
  33  * check-output-end
  34  */
```

```
**********************************************************
     485 Mon Aug  5 08:38:51 2019
new/usr/src/tools/smatch/src/validation/sm_implied12.c
11506 smatch resync
**********************************************************
_____unchanged_portion_omitted_
```

```
  31 /*
  32  * check-name: smatch implied #12
  33  * check-command: smatch -I.. -m64 sm_implied12.c
  34  *
  35  * check-output-start
  36 sm_implied12.c:28 ad_agg_selection_logic() implied: foo = '0,4096-ptr_max'
  36 sm_implied12.c:28 ad_agg_selection_logic() implied: foo = '0,4096-21177777777777
  37  * check-output-end
  38  */
```

****
641 Mon Aug 5 08:38:52 2019
new/usr/src/tools/smatch/src/validation/sm_implied2.c
11506 smatch resync
****
_____unchanged_portion_omitted_

```
 34 /*
 35  * check-name: Smatch implied #2
 36  * check-command: smatch --spammy sm_implied2.c
 37  *
 38  * check-output-start
 39 sm_implied2.c:28 func() error: potentially dereferencing uninitialized 'aa'.
 40 sm_implied2.c:28 func() error: potentially dereferencing uninitialized 'aa'.
 41  * check-output-end
 42  */
```

```
**********************************************************
     478 Mon Aug  5 08:38:52 2019
new/usr/src/tools/smatch/src/validation/sm_implied5.c
11506 smatch resync
**********************************************************
_____unchanged_portion_omitted_
  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 sm_implied5.c:18 func() error: potentially dereferencing uninitialized 'aa'.
  27  * check-output-end
  28  */
```

```
**********************************************************
     466 Mon Aug  5 08:38:53 2019
new/usr/src/tools/smatch/src/validation/sm_memory.c
11506 smatch resync
**********************************************************
_____unchanged_portion_omitted_
  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'
  33  * check-output-end
  34  */
```

```
**********************************************************
     904 Mon Aug  5 08:38:53 2019
new/usr/src/tools/smatch/src/validation/sm_null_deref.c
11506 smatch resync
**********************************************************
_____unchanged_portion_omitted_
  41 /*
  42  * check-name: Null Dereferences
  43  * check-command: smatch --spammy -I.. sm_null_deref.c
  44  *
  45  * check-output-start
  46 sm_null_deref.c:18 func() error: potentially dereferencing uninitialized 'aa'.
  47 sm_null_deref.c:18 func() error: potentially dereferencing uninitialized 'aa'.
  48 sm_null_deref.c:23 func() error: we previously assumed 'a' could be null (see li
  49 sm_null_deref.c:25 func() warn: variable dereferenced before check 'a' (see line
  50 sm_null_deref.c:30 func() error: we previously assumed 'b' could be null (see li
  51  * check-output-end
  52  */
```

****
519 Mon Aug 5 08:38:54 2019
new/usr/src/tools/smatch/src/validation/sm_select5.c
11506 smatch resync
****
_____***unchanged_portion_omitted_***

```
 22 /*
 23  * check-name: smatch select #5
 24  * check-command: smatch -I.. sm_select5.c
 25  *
 26  * check-output-start
 27 sm_select5.c:15 test() implied: ret = '(-12)'
 28 sm_select5.c:16 test() implied: a = 's32min-(-1),4-s32max'
 28 sm_select5.c:16 test() implied: a = 's32min-s32max'
 29 sm_select5.c:18 test() implied: a = '0-3'
 30  * check-output-end
 31  */
```

```
**********************************************************
    5450 Mon Aug  5 08:38:54 2019
new/usr/src/uts/i86pc/unix/Makefile
11506 smatch resync
**********************************************************
#
# 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 2008 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#
# Copyright 2019 Joyent, Inc.
# Copyright (c) 2018, Joyent, Inc.
# Copyright 2019 OmniOS Community Edition (OmniOSce) Association.

#
#       Path to the base of the uts directory tree (usually /usr/src/uts).
#
UTSBASE = ../..

#
#       Define the module and object file sets.
#
UNIX            = unix
DBOOT           = dboot

OBJECTS         = $(SPECIAL_OBJS:%=$(OBJS_DIR)/%) \
                  $(CORE_OBJS:%=$(OBJS_DIR)/%) \
                  $(KRTLD_OBJS:%=$(OBJS_DIR)/%) \
                  $(MACH_NOT_YET_KMODS:%=$(OBJS_DIR)/%)

ROOTMODULE      = $(ROOT_PSM_KERN_DIR)/$(UNIX)

UNIX_BIN        = $(OBJS_DIR)/$(UNIX)

LIBS            = $(GENLIB)

GENUNIX         = genunix
GENUNIX_DIR     = ../../intel/$(GENUNIX)

LIBOPTS         = -L $(GENUNIX_DIR)/$(OBJS_DIR) -l $(GENUNIX)

COMMP_CTF_SRC   = $(OBJS_DIR)/comm_page_ctf.c

CTFEXTRAOBJS    = $(OBJS_DIR)/vers.o $(OBJS_DIR)/comm_page_ctf.o

DBOOT_OBJS_DIR  = dboot/$(OBJS_DIR)
DBOOT_OBJECTS   = $(DBOOT_OBJS:%=$(DBOOT_OBJS_DIR)/%)
DBOOT_BIN       = $(DBOOT_OBJS_DIR)/$(DBOOT)
```



```
DBOOT_O         = $(OBJS_DIR)/$(DBOOT).o
DBOOT_S         = $(DBOOT_O:%.o=%.s)

#
#       Include common rules.
#
include $(UTSBASE)/i86pc/Makefile.i86pc

#
#       Define targets
#
ALL_TARGET      = $(UNIX_BIN)
INSTALL_TARGET  = $(UNIX_BIN) $(ROOTMODULE)

#
#       This is UNIX_DIR. Use a short path.
#
UNIX_DIR        = .

#
#       Overrides
#
CLEANFILES      +=              \
        $(UNIX_O) $(MODSTUBS_O) \
        $(OBJS_DIR)/vers.c \
        $(OBJS_DIR)/dtracestubs.s \
        $(DTRACESTUBS_O) $(DTRACESTUBS) \
        $(CTFEXTRAOBJS) \
        $(COMMP_CTF_SRC)

CLEANFILES      +=              \
        $(DBOOT_O) $(DBOOT_S)   \
        $(DBOOT_OBJECTS)        \
        $(OBJS_DIR)/bios_call_src.o \
        $(OBJS_DIR)/bios_call_src \
        $(OBJS_DIR)/bios_call.s \
        $(DBOOT_BIN)

CLEANFILES      +=              \
        $(DBOOT_OBJS_DIR)/$(FONT).c \
        $(OBJS_DIR)/$(FONT).c

CLEANFILES      +=              \
        $(OBJS_DIR)/fb_swtch_src.o \
        $(OBJS_DIR)/fb_swtch_src \
        $(OBJS_DIR)/fb_swtch.s

CLEANFILES      += \
        $(ZLIB_OBJS:%.o=$(OBJS_DIR)/%.o) \
        $(ZLIB_OBJS:%.o=$(OBJS_DIR)/%.ln)

CLOBBERFILES    = $(CLEANFILES) $(UNIX_BIN)

# instr_size needs a special header
$(OBJS_DIR)/instr_size.o :=     EXTRA_OPTIONS   = -I$(SRC)/common/dis/i386
$(OBJS_DIR)/instr_size.ln :=    EXTRA_OPTIONS   = -I$(SRC)/common/dis/i386

#
# When performing shadow studio builds, the pre-processed comm page
# output from gcc can end up confusing studio.
#
$(OBJS_DIR)/comm_page_ctf.o :=  CERRWARN += -_cc=-erroff=E_TKNS_IGNORED_AT_END_O

CFLAGS += -DDIS_MEM

#
```

```
127 # For now, disable these checks; maintainers should endeavor
128 # to investigate and remove these for maximum coverage.
129 # Please do not carry these forward to new Makefiles.
130 #
131 CERRWARN        += -_gcc=-Wno-parentheses
132 CERRWARN        += -_gcc=-Wno-uninitialized
133 CERRWARN        += -_gcc=-Wno-char-subscripts
134 CERRWARN        += -_gcc=-Wno-unused-variable
135 CERRWARN        += -_gcc=-Wno-unused-function
136 CERRWARN        += -_gcc=-Wno-unused-label
137 CERRWARN        += -_gcc=-Wno-type-limits
138 CERRWARN        += -_gcc=-Wno-clobbered
139 CERRWARN        += -_gcc=-Wno-empty-body
140 CERRWARN        += -_gcc=-Wno-unused-value

142 # false positives
143 SMOFF += index_overflow

145 # needs work
146 SMOFF += all_func_returns,deref_check,signed

148 $(OBJS_DIR)/fmsmb.o := SMOFF += indenting
149 $(OBJS_DIR)/zutil.o := SMOFF += indenting
150 $(OBJS_DIR)/bootrd_cpio.o := SMOFF += allocating_enough_data

152 # too hairy
153 $(OBJS_DIR)/inflate.o := SMATCH=off

155 #
156 #       Default build targets.
157 #
158 .KEEP_STATE:

160 def: $(DEF_DEPS)

162 all: $(ALL_DEPS)

164 clean: $(CLEAN_DEPS)

166 clobber: $(CLOBBER_DEPS)

168 install: $(INSTALL_DEPS)

170 MAPFILE_32 = $(MAPFILE)
171 MAPFILE_64 = $(MAPFILE).amd64

173 MAPFILE_NAME = $(MAPFILE_$(CLASS))

175 $(UNIX_BIN):    $(UNIX_O) $(MODSTUBS_O) $(MAPFILE_NAME) \
176                 $(GENLIB) $(DTRACESTUBS) $(DBOOT_O)
177         $(LD) -dy -b -o $@ -e dboot_image -znointerp -M $(MAPFILE_NAME) \
178             $(UNIX_O) $(DBOOT_O) $(MODSTUBS_O) $(LIBOPTS) \
179             $(DTRACESTUBS)
180         $(MBH_PATCH) $(UNIX_BIN)
181         $(CTFMERGE_UNIQUIFY_AGAINST_GENUNIX)
182         $(POST_PROCESS)

184 $(UNIX_O):      $(OBJECTS) $(CTFEXTRAOBJS)
185         $(LD) -r -o $@ $(OBJECTS) $(OBJS_DIR)/vers.o

187 $(DBOOT_BIN):   $(DBOOT_OBJS_DIR) $(DBOOT_OBJECTS) dboot/Mapfile.dboot
188         $(LD) -dn -e _start -M dboot/Mapfile.dboot \
189                 -o $(DBOOT_BIN) $(DBOOT_OBJECTS)

191 $(DBOOT_O):     $(DBOOT_BIN)
192         @echo " .data"                                  > $(DBOOT_S)
```



```
193         @echo " .globl  dboot_image"                    >> $(DBOOT_S)
194         @echo "dboot_image:"                           >> $(DBOOT_S)
195         $(ELFEXTRACT) $(DBOOT_BIN)                      >> $(DBOOT_S)
196         $(COMPILE.s) -o $(DBOOT_O) $(DBOOT_S)

198 $(DBOOT_OBJS_DIR):
199         -@mkdir -p $@ 2> /dev/null

201 $(COMMP_CTF_SRC):       $(UTSBASE)/i86pc/ml/comm_page.s
202         $(COMPILE.cpp) -D_GENCTF -o $@ $(UTSBASE)/i86pc/ml/comm_page.s

204 $(OBJS_DIR)/comm_page_ctf.o:    $(COMMP_CTF_SRC)
205         $(COMPILE.c) -o $@ $<
206         $(CTFCONVERT_O)

208 #
209 #       Special rules for generating assym.h for inclusion in assembly files.
210 #
211 $(DSF_DIR)/$(OBJS_DIR)/assym.h $(DSF_DIR)/$(OBJS_DIR)/kdi_assym.h:      FRC
212         @cd $(DSF_DIR); $(MAKE) all.targ

214 #
215 #       Include common targets.
216 #
217 include $(UTSBASE)/i86pc/Makefile.targ
```

```
**********************************************************
    2983 Mon Aug  5 08:38:54 2019
new/usr/src/uts/intel/emlxs/Makefile
11506 smatch resync
**********************************************************
#
# 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) 2011 Bayard G. Bell. All rights reserved.
# Copyright 2019 Joyent, Inc.
# Copyright (c) 2018, Joyent, Inc.
#
# This makefile drives the production of the emlxs driver kernel module.
#
# Path to the base of the uts directory tree (usually /usr/src/uts).
#
UTSBASE         = ../../
COMMON_BASE     = ../../../common

#
#       Define the module and object file sets.
#
MODULE          = emlxs
OBJECTS         = $(EMLXS_OBJS:%=$(OBJS_DIR)/%)
LINTS           = $(EMLXS_OBJS:%.o=$(LINTS_DIR)/%.ln)
ROOTMODULE      = $(ROOT_DRV_DIR)/$(MODULE)
CONF_SRCDIR     = $(UTSBASE)/common/io/fibre-channel/fca/emlxs

#
#       Include common rules.
#
ARCHDIR         = intel
include ../Makefile.$(ARCHDIR)

#
#       Define targets
#
ALL_TARGET      = $(BINARY) $(SRC_CONFILE)
LINT_TARGET     = $(MODULE).lint
INSTALL_TARGET  = $(BINARY) $(ROOTMODULE) $(ROOT_CONFFILE)

EMLXS_FLAGS             = -DEMLXS_I386
EMLXS_FLAGS             += -DS11
EMLXS_FLAGS             += -DVERSION=\"11\"
EMLXS_FLAGS             += -DMACH=\"$(MACH)\"
EMLXS_CFLAGS            = $(EMLXS_FLAGS)
```



```
EMLXS_LFLAGS            = $(EMLXS_FLAGS)
CFLAGS                  += $(EMLXS_CFLAGS) -DEMLXS_ARCH=\"$(CLASS)\"
LINTTAGS                += $(EMLXS_LFLAGS) -DEMLXS_ARCH=\"$(CLASS)\"


#
#       Overrides and depends_on
#
INC_PATH        += -I$(ROOT)/usr/include
INC_PATH        += -I$(UTSBASE)/common/sys
INC_PATH        += -I$(COMMON_BASE)/bignum
INC_PATH        += -I$(UTSBASE)/common/sys/fibre-channel
INC_PATH        += -I$(UTSBASE)/common/sys/fibre-channel/fca
INC_PATH        += -I$(UTSBASE)/common/sys/fibre-channel/fca/emlxs
INC_PATH        += -I$(UTSBASE)/common/sys/fibre-channel/impl
INC_PATH        += -I$(UTSBASE)/common/sys/fibre-channel/ulp

#
#       misc/fctl required because #ifdef MODSYM_LOAD code
#       triggered by -DS11; uses DDI calls to load FCA symbols
#
LDFLAGS         += -dy -Nmisc/md5 -Nmisc/sha1
LDFLAGS         += -Nmisc/bignum -Nmisc/fctl

#
# For now, disable these lint checks; maintainers should endeavor
# to investigate and remove these for maximum lint coverage.
#
LINTTAGS        += -erroff=E_BAD_PTR_CAST_ALIGN
LINTTAGS        += -erroff=E_STATIC_UNUSED
LINTTAGS        += -erroff=E_ASSIGN_NARROW_CONV
LINTTAGS        += -erroff=E_SUSPICIOUS_COMPARISON
LINTTAGS        += -erroff=E_INCONS_VAL_TYPE_DECL2

CERRWARN        += -_gcc=-Wno-parentheses
CERRWARN        += -_gcc=-Wno-unused-label
CERRWARN        += -_gcc=-Wno-uninitialized

# needs work
SMOFF += indenting,deref_check,all_func_returns,index_overflow
SMOFF += indenting,deref_check,all_func_returns

# seems definitely wrong
$(OBJS_DIR)/emlxs_fcf.o := SMOFF += logical_instead_of_bitwise

#
#       Default build targets.
#
.KEEP_STATE:

def:            $(DEF_DEPS)

all:            $(ALL_DEPS)

clean:          $(CLEAN_DEPS)

clobber:        $(CLOBBER_DEPS)

lint:           $(LINT_DEPS)

modlintlib:     $(MODLINTLIB_DEPS)

clean.lint:     $(CLEAN_LINT_DEPS)

install:        $(INSTALL_DEPS)
```

```
107 #
108 #       Include common targets.
109 #
110 include ../Makefile.targ
```

```
**********************************************************
    4224 Mon Aug  5 08:38:55 2019
new/usr/src/uts/intel/genunix/Makefile
11506 smatch resync
**********************************************************
```

```
#
# 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 2019 Joyent, Inc.
# Copyright (c) 2018, Joyent, Inc.

#
#       This makefile drives the production of the generic
#       unix kernel module.
#
#       x86 implementation architecture dependent
#

#
#       Path to the base of the uts directory tree (usually /usr/src/uts).
#
UTSBASE = ../..

#
#       Define the module and object file sets.
#
MODULE          = genunix
GENUNIX         = $(OBJS_DIR)/$(MODULE)

OBJECTS         = $(GENUNIX_OBJS:%=$(OBJS_DIR)/%) \
                  $(NOT_YET_KMODS:%=$(OBJS_DIR)/%)

LINTS           = $(GENUNIX_OBJS:%.o=$(LINTS_DIR)/%.ln) \
                  $(NOT_YET_KMODS:%.o=$(LINTS_DIR)/%.ln)

ROOTMODULE      = $(ROOT_KERN_DIR)/$(MODULE)

LIBGEN          = $(OBJS_DIR)/libgenunix.so
LIBSTUBS        = $(GENSTUBS_OBJS:%=$(OBJS_DIR)/%)

#
#       Include common rules.
#
include $(UTSBASE)/intel/Makefile.intel
```



```
#
#       Define targets
#
ALL_TARGET      = $(LIBGEN) $(GENUNIX)
LINT_TARGET     = $(MODULE).lint
INSTALL_TARGET  = $(LIBGEN) $(GENUNIX) $(ROOTMODULE)

#
#       Overrides
#
CLOBBERFILES    += $(GENUNIX)
CLEANFILES      += $(LIBSTUBS) $(LIBGEN)
BINARY          =

#
# Non-patch genunix builds merge a version of the ip module called ipctf.  This
# is to ensure that the common network-related types are included in genunix and
# can thus be uniquified out of other modules.  We don't want to do this for
# patch builds, since we can't guarantee that ip and genunix will be in the same
# patch.
#
IPCTF_TARGET    = $(IPCTF)
$(PATCH_BUILD)IPCTF_TARGET =

CPPFLAGS        += -I$(SRC)/common
CPPFLAGS        += -I$(SRC)/uts/common/fs/zfs

CPPFLAGS        += -I$(UTSBASE)/i86pc

#
# For now, disable these lint checks; maintainers should endeavor
# to investigate and remove these for maximum lint coverage.
# Please do not carry these forward to new Makefiles.
#
LINTTAGS        += -erroff=E_SUSPICIOUS_COMPARISON
LINTTAGS        += -erroff=E_BAD_PTR_CAST_ALIGN
LINTTAGS        += -erroff=E_SUPPRESSION_DIRECTIVE_UNUSED
LINTTAGS        += -erroff=E_STATIC_UNUSED
LINTTAGS        += -erroff=E_PTRDIFF_OVERFLOW
LINTTAGS        += -erroff=E_ASSIGN_NARROW_CONV

CERRWARN        += -_gcc=-Wno-unused-label
CERRWARN        += -_gcc=-Wno-unused-variable
CERRWARN        += -_gcc=-Wno-unused-value
CERRWARN        += -_gcc=-Wno-unused-function
CERRWARN        += -_gcc=-Wno-parentheses
CERRWARN        += -_gcc=-Wno-switch
CERRWARN        += -_gcc=-Wno-type-limits
CERRWARN        += -_gcc=-Wno-uninitialized
CERRWARN        += -_gcc=-Wno-clobbered
CERRWARN        += -_gcc=-Wno-empty-body

# very hairy
$(OBJS_DIR)/u8_textprep.o := SMATCH=off

# false positives
SMOFF += index_overflow
$(OBJS_DIR)/seg_vn.o := SMOFF += deref_check
$(OBJS_DIR)/ddi_intr_irm.o := SMOFF += deref_check

# need work still
SMOFF += signed,indenting,all_func_returns
$(OBJS_DIR)/clock_highres.o := SMOFF += signed_integer_overflow_check
$(OBJS_DIR)/evchannels.o := SMOFF += allocating_enough_data
$(OBJS_DIR)/klpd.o := SMOFF += cast_assign
```

```
105 $(OBJS_DIR)/lookup.o := SMOFF += strcpy_overflow
106 $(OBJS_DIR)/process.o := SMOFF += or_vs_and
107 $(OBJS_DIR)/sunpci.o := SMOFF += deref_check
108 $(OBJS_DIR)/timers.o := SMOFF += signed_integer_overflow_check

110 # definitely wrong
111 $(OBJS_DIR)/acl_common.o := SMOFF += or_vs_and

113 #
133 # Ensure that lint sees 'struct cpu' containing a fully declared
134 # embedded 'struct machcpu'
135 #
136 LINTFLAGS       += -D_MACHDEP -I../../i86pc

138 #
114 #       Default build targets.
115 #
116 .KEEP_STATE:

118 def:            $(DEF_DEPS)

120 all:            $(ALL_DEPS)

122 clean:          $(CLEAN_DEPS)

124 clobber:        $(CLOBBER_DEPS)

151 lint:           $(LINT_DEPS)

153 modlintlib:     $(MODLINTLIB_DEPS)

155 clean.lint:     $(CLEAN_LINT_DEPS)

126 install:        $(INSTALL_DEPS)

128 # Due to what seems to be an issue in GCC 4 generated DWARF containing
129 # symbolic relocations against non-allocatable .debug sections, libgenunix.so
130 # must be built from a stripped object, thus we create an intermediary
131 # libgenunix.o we can safely strip.
132 LIBGENUNIX_O = $(OBJS_DIR)/libgenunix.o
133 CLEANFILES += $(LIBGENUNIX_O)

135 $(LIBGENUNIX_O): $(OBJECTS)
136         $(LD) -r -o $(OBJS_DIR)/libgenunix.o $(OBJECTS)
137         $(STRIP) -x $(OBJS_DIR)/libgenunix.o

139 $(LIBGEN):      $(LIBGENUNIX_O) $(LIBSTUBS)
140         $(BUILD.SO) $(LIBGENUNIX_O) $(LIBSTUBS)

142 $(IPCTF_TARGET) ipctf_target: FRC
143         @cd $(IPDRV_DIR); pwd; $(MAKE) ipctf.$(OBJS_DIR)
144         @pwd

146 $(GENUNIX): $(IPCTF_TARGET) $(OBJECTS)
147         $(LD) -r $(LDFLAGS) -o $@ $(OBJECTS)
148         $(CTFMERGE_GENUNIX_MERGE)
149         $(POST_PROCESS)

151 #
152 #       Include common targets.
153 #
154 include $(UTSBASE)/intel/Makefile.targ

156 #
157 #       Software workarounds for hardware "features".
158 #
```



```
159 include $(UTSBASE)/i86pc/Makefile.workarounds

161 ALL_DEFS += $(WORKAROUND_DEFS)

194 #
195 # Override.
196 #
197 $(MODULE).lint := GEN_LINT_LIB =
```

```
**********************************************************
    1829 Mon Aug  5 08:38:55 2019
new/usr/src/uts/intel/mega_sas/Makefile
11506 smatch resync
**********************************************************
#
# 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 2008 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#
# Copyright 2019 Joyent, Inc.
# Copyright (c) 2018, Joyent, Inc.

#
# uts/intel/mega_sas/Makefile
#
#       This makefile drives the production of the mega_sas driver kernel module
#
#       intel implementation architecture dependent
#

#
#       Path to the base of the uts directory tree (usually /usr/src/uts).
#
UTSBASE = ../..

#
#       Define the module and object file sets.
#
MODULE          = mega_sas
OBJECTS         = $(MEGA_SAS_OBJS:%=$(OBJS_DIR)/%)
LINTS           = $(MEGA_SAS_OBJS:%.o=$(LINTS_DIR)/%.ln)
ROOTMODULE      = $(ROOT_DRV_DIR)/$(MODULE)
CONF_SRCDIR     = $(UTSBASE)/common/io/mega_sas

#
#       Include common rules.
#
include $(UTSBASE)/intel/Makefile.intel

#
#       Define targets
#
ALL_TARGET      = $(BINARY) $(CONFMOD)
LINT_TARGET     = $(MODULE).lint
INSTALL_TARGET  = $(BINARY) $(ROOTMODULE) $(ROOT_CONFFILE)

#
```



```
#       Kernel Module Dependencies
#
LDFLAGS         += -dy -Nmisc/scsi

CERRWARN        += -_gcc=-Wno-uninitialized

# needs work
$(OBJS_DIR)/megaraid_sas.o := SMOFF += snprintf_overflow,all_func_returns,index_
$(OBJS_DIR)/megaraid_sas.o := SMOFF += snprintf_overflow,all_func_returns

#
#       Default build targets.
#
.KEEP_STATE:

def:            $(DEF_DEPS)

all:            $(ALL_DEPS)

clean:          $(CLEAN_DEPS)

clobber:        $(CLOBBER_DEPS)

lint:           $(LINT_DEPS)

modlintlib:     $(MODLINTLIB_DEPS)

clean.lint:     $(CLEAN_LINT_DEPS)

install:        $(INSTALL_DEPS)

#
#       Include common targets.
#
include $(UTSBASE)/intel/Makefile.targ
```

```
**********************************************************
    1820 Mon Aug  5 08:38:56 2019
new/usr/src/uts/intel/simnet/Makefile
11506 smatch resync
**********************************************************
```

```
   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 2019 Joyent, Inc.
  26 #

  28 #
  29 # Path to the base of the uts directory tree (usually /usr/src/uts).
  30 #
  31 UTSBASE = ../..

  33 #
  34 # Define the module and object file sets.
  35 #
  36 MODULE          = simnet
  37 OBJECTS         = $(SIMNET_OBJS:%=$(OBJS_DIR)/%)
  35 LINTS           = $(SIMNET_OBJS:%.o=$(LINTS_DIR)/%.ln)
  38 ROOTMODULE      = $(ROOT_DRV_DIR)/$(MODULE)
  39 CONF_SRCDIR     = $(UTSBASE)/common/io/$(MODULE)

  41 #
  42 # Include common rules.
  43 #
  44 include $(UTSBASE)/intel/Makefile.intel

  46 #
  47 # Define targets
  48 #
  49 ALL_TARGET      = $(BINARY) $(SRC_CONFILE)
  48 LINT_TARGET     = $(MODULE).lint
  50 INSTALL_TARGET  = $(BINARY) $(ROOTMODULE) $(ROOT_CONFFILE)

  52 #
  53 # Overrides
  54 #
  55 CFLAGS          += $(CCVERBOSE)
  56 LDFLAGS         += -dy -Ndrv/dld -Nmisc/mac -Nmisc/dls -Ndrv/random

  58 CERRWARN        += -_gcc=-Wno-switch
```



```
  60 # needs work
  61 $(OBJS_DIR)/simnet.o := SMOFF += index_overflow

  63 #
  64 #       Default build targets.
  65 #
  66 .KEEP_STATE:

  68 def:            $(DEF_DEPS)

  70 all:            $(ALL_DEPS)

  72 clean:          $(CLEAN_DEPS)

  74 clobber:        $(CLOBBER_DEPS)

  72 lint:           $(LINT_DEPS)

  74 modlintlib:     $(MODLINTLIB_DEPS)

  76 clean.lint:     $(CLEAN_LINT_DEPS)

  76 install:        $(INSTALL_DEPS)

  78 #
  79 #       Include common targets.
  80 #
  81 include $(UTSBASE)/intel/Makefile.targ
```

```
**********************************************************
    2087 Mon Aug  5 08:38:56 2019
new/usr/src/uts/intel/spppcomp/Makefile
11506 smatch resync
**********************************************************
```

```
#
# 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
#
#
# uts/intel/spppcomp/Makefile
#
# Copyright 2007 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
# Copyright (c) 2011 Bayard G. Bell. All rights reserved.
#
# Copyright 2019 Joyent, Inc.
#
# Copyright (c) 2018, Joyent, Inc.

#
#       Path to the base of the uts directory tree (usually /usr/src/uts).
#
UTSBASE = ../..

#
#       Define the module and object file sets.
#
MODULE          = spppcomp
OBJECTS         = $(SPPPCOMP_OBJS:%=$(OBJS_DIR)/%)
LINTS           = $(SPPPCOMP_OBJS:%.o=$(LINTS_DIR)/%.ln)
ROOTMODULE      = $(USR_STRMOD_DIR)/$(MODULE)

#
#       Include common rules.
#
include $(UTSBASE)/intel/Makefile.intel

#
#       Define targets
#
ALL_TARGET      = $(BINARY)
LINT_TARGET     = $(MODULE).lint
INSTALL_TARGET  = $(BINARY) $(ROOTMODULE)

#
#       Internal build definitions
#
CPPFLAGS        += -DINTERNAL_BUILD -DSOL2 -DMUX_FRAME
```



```
#
#       Additional compiler definitions
#
INC_PATH        += -I$(UTSBASE)/common/io/ppp/common

#
# For now, disable these lint checks; maintainers should endeavor
# to investigate and remove these for maximum lint coverage.
# Please do not carry these forward to new Makefiles.
#
LINTTAGS        += -erroff=E_BAD_PTR_CAST_ALIGN
LINTTAGS        += -erroff=E_PTRDIFF_OVERFLOW
LINTTAGS        += -erroff=E_ASSIGN_NARROW_CONV

CERRWARN        += -_gcc=-Wno-parentheses
CERRWARN        += -_gcc=-Wno-uninitialized

# needs work
SMOFF += indenting,index_overflow
SMOFF += indenting

#
# Depends on sppp
#
LDFLAGS         += -dy -N drv/sppp

#
#       Default build targets.
#
.KEEP_STATE:

def:            $(DEF_DEPS)

all:            $(ALL_DEPS)

clean:          $(CLEAN_DEPS)

clobber:        $(CLOBBER_DEPS)

lint:           $(LINT_DEPS)

modlintlib:     $(MODLINTLIB_DEPS)

clean.lint:     $(CLEAN_LINT_DEPS)

install:        $(INSTALL_DEPS)

$(LINTS_DIR)/%.ln:              $(UTSBASE)/common/io/ppp/spppcomp/%.c
        @($(LHEAD) $(LINT.c) $< $(LTAIL))

$(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/ppp/spppcomp/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

#
#       Include common targets.
#
include $(UTSBASE)/intel/Makefile.targ
```

```
**********************************************************
    3034 Mon Aug  5 08:38:56 2019
new/usr/src/uts/intel/xge/Makefile
11506 smatch resync
**********************************************************
#
# 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 2008 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#
# Copyright 2019 Joyent, Inc.
#
# Copyright (c) 2018, Joyent, Inc.

#
#       Paths to the base of the uts directory trees
#
UTSBASE   = ../..

#
#       Define the module and object file sets.
#
MODULE          = xge
OBJECTS         = $(XGE_HAL_OBJS:%=$(OBJS_DIR)/%) $(XGE_OBJS:%=$(OBJS_DIR)/%)
LINTS           = $(XGE_HAL_OBJS:%.o=$(LINTS_DIR)/%.ln) $(XGE_OBJS:%.o=$(LINTS_D
ROOTMODULE      = $(ROOT_DRV_DIR)/$(MODULE)

#
#       Include common rules.
#
include $(UTSBASE)/intel/Makefile.intel

#
#       Define targets
#
ALL_TARGET      = $(BINARY)
LINT_TARGET     = $(MODULE).lint
INSTALL_TARGET  = $(BINARY) $(ROOTMODULE)

#
#       GENERAL PURPOUSE HAL FLAGS: Tuning HAL for Solaris specific modes
#
HAL_CFLAGS       = -DXGE_HAL_USE_MGMT_AUX

#
#       TRACE SECTION: Possible values for MODULE, TRACE and ERR masks:
```



```
#
# XGE_COMPONENT_HAL_CONFIG              0x1
# XGE_COMPONENT_HAL_FIFO                0x2
# XGE_COMPONENT_HAL_RING                0x4
# XGE_COMPONENT_HAL_CHANNEL             0x8
# XGE_COMPONENT_HAL_DEVICE              0x10
# XGE_COMPONENT_HAL_MM                  0x20
# XGE_COMPONENT_HAL_QUEUE               0x40
# XGE_COMPONENT_HAL_STATS               0x100
# XGE_COMPONENT_OSDEP                   0x1000
# XGE_COMPONENT_LL                      0x2000
# XGE_COMPONENT_TOE                     0x4000
# XGE_COMPONENT_RDMA                    0x8000
# XGE_COMPONENT_ALL                     0xffffffff
#TRACE_CFLAGS = -DXGE_DEBUG_MODULE_MASK=0xffffffff \
#               -DXGE_DEBUG_TRACE_MASK=0xffffffff \
#               -DXGE_DEBUG_ERR_MASK=0xffffffff
TRACE_CFLAGS    = -DXGE_DEBUG_MODULE_MASK=0x00003010 \
                  -DXGE_DEBUG_TRACE_MASK=0x00000000  \
                  -DXGE_DEBUG_ERR_MASK=0x00003010

XGE_CFLAGS      = $(HAL_CFLAGS) $(TRACE_CFLAGS) $(CCVERBOSE) \
                  -I$(UTSBASE)/common/io/xge/hal/include \
                  -I$(UTSBASE)/common/io/xge/hal/xgehal \
                  -I$(UTSBASE)/common/io/xge/drv -DSOLARIS

CFLAGS          += $(XGE_CFLAGS) -xO4 -xcrossfile
CFLAGS64        += $(XGE_CFLAGS) -xO4 -xcrossfile

#
#       Driver depends on MAC & IP
#
LDFLAGS         +=  -dy -N misc/mac -N drv/ip

#
#       Lint flag
#
LINTFLAGS       += $(XGE_CFLAGS) -Xc99=%all

#
# For now, disable these lint checks; maintainers should endeavor
# to investigate and remove these for maximum lint coverage.
# Please do not carry these forward to new Makefiles.
#
LINTTAGS        += -erroff=E_BAD_PTR_CAST_ALIGN
LINTTAGS        += -erroff=E_STATIC_UNUSED
LINTTAGS        += -erroff=E_PTRDIFF_OVERFLOW

CERRWARN        += -_gcc=-Wno-parentheses
CERRWARN        += -_gcc=-Wno-unused-variable
CERRWARN        += -_gcc=-Wno-unused-label
CERRWARN        += -_gcc=-Wno-empty-body
CERRWARN        += -_gcc=-Wno-uninitialized

# needs work
SMOFF += indenting
SMOFF += all_func_returns
SMOFF += no_if_block
SMOFF += allocating_enough_data
SMOFF += indenting,all_func_returns,no_if_block

#
#
#       Default build targets.
#
.KEEP_STATE:
```

```
111 def:            $(DEF_DEPS)

113 all:            $(ALL_DEPS)

115 clean:          $(CLEAN_DEPS)

117 clobber:        $(CLOBBER_DEPS)

130 lint:           $(LINT_DEPS)

132 modlintlib:     $(MODLINTLIB_DEPS)

134 clean.lint:     $(CLEAN_LINT_DEPS)

119 install:        $(INSTALL_DEPS)

121 #
122 #       Include common targets.
123 #
124 include $(UTSBASE)/intel/Makefile.targ
```