
5823 Thu Jun 30 22:01:06 2016

new/./README-ASLR.md

7029 want per-process exploit mitigation features (secflags)

7030 want basic address space layout randomization (aslr)

7031 noexec_user_stack should be a secflag

7032 want a means to forbid mappings around NULL.

1 # Address Space Layout Randomisation

3 Briefly, see: 'psecflags(1)', 'security-flags(5)'

5 ## Administration

7 ASLR is implemented via process security-flags (which we introduce), there are
8 four sets of flags per-process: The effective, inheritable, upper, and lower
9 sets (see 'security-flags(5)'). The effective set is immutable, it can only
10 change when the process calls 'exec(2)', at which point the effective set is
11 replaced by the inheritable set (with one exception). Security flags are
12 inherited upon 'fork(2)' (but the inheritable set is not promoted until
13 'exec(2)', as mentioned). The upper and lower sets bound the permitted values
14 of the inheritable set.

16 This is such that a given execution of an executable has a constant set of
17 security-flags, which simplifies things for everyone.

19 This unfortunately means that to enable ASLR fully system-wide, requires a
20 reboot or at least restart of a majority of services.

22 The system-wide ASLR flag is an SMF property on the new service
23 'svc:/system/process-security', which contains 'default', 'upper', and 'lower'
24 property groups, with one boolean property per implemented flag (see, again,
25 'security-flags(5)'). These will only affect services (and their children)
26 started via SMF after the values have been changed.

28 Per-process setting (and inspecting) of security-flags is done via
29 'psecflags(1)'.
30

31 Per-service setting of security-flags is achievable by the 'security_flags'
32 property on the service 'method_context'. A 'default' pseudo-flag specifies
33 the flags from 'svc:/system/process-security', and flags can be
34 added/subtracted from there much the same as with 'privileges(5)'.
35

36 ## Privilege

38 A process may change the security-flags of any process to which it could send
39 a signal with 'kill(2)', as long as the process also has the
40 'PRIV_PROC_SECFLAGS' privilege. This privilege is granted by default, but is
41 not a 'basic' privilege. If you have configured custom privileges for certain
42 users or services, they will not automatically gain the new
43 'PRIV_PROC_SECFLAGS' (unfortunately).

45 ## Executable tagging

47 There is a somewhat compatible property of dynamic executables, 'DT_SUNW_ASLR'
48 which controls the ASLR behaviour of a given executable. If this dynamic tag
49 has a value of 1, ASLR is always enabled for an execution of this process. If
50 the tag has a value of 0, ASLR is never enabled for an execution of this
51 process. The default is to inherit the ASLR flag as normal.

53 This allows a process for which ASLR is known to be problematic to explicitly
54 forbid it, and for processes of special sensitivity to mandate it.

56 This is controlled via the '-z aslr' flag to 'ld(1)'.
57

58

59 ## Per-zone configuration

61 The default security flags for a zone, and the upper and lower limits, may be
62 specified with the security-flags resource in zonecfg(1M). These are applied
63 to every process in NGZs (unlike the GZ, where there are a small number of
64 processes we must miss)

67 ## Missing bits/Problems/Worries

69 ### The stack skewing may skew too much of the stack

71 At present, we skew the absolute base of the stack, which means the gap
72 between a user stack frame and the process environment is constant. I have
73 this vague memory that that's sub-par, and that we want to skew the stack
74 after the environment, etc. I could be entirely wrong about that though.

76 ### We skew each mapping separately

78 Some systems calculate a random skew for the various parts of a process at
79 execution time, and apply that same skew to each mapping. That obviously
80 minimises the performance impact of this significantly for processes that
81 perform many mappings.

83 Due to some unfortunate aspects of how we manage the user address space and
84 mappings, we don't do that right now. We calculate a separate random skew for
85 each mapping we attempt.

87 ### The way we skew mappings is problematic

89 The user address space is currently managed somewhat unfortunately (from our
90 point of view, at least). When attempting a mapping we first determine the
91 highest gap into which the requested mapping can fit, and then we place the
92 mapping at the highest address in that gap. This is how we manage user
93 fragmentation.

95 The current ASLR implementation works in basically the same way. We still
96 find the highest gap into which the given mapping may fit, and choose the
97 highest address at which it may fit. The only difference is that we then slew
98 it backward by a random (but co-aligned), amount.

100 This is obviously not as random as it could be, but is the easiest way I've
101 found in the current code base for introducing uncertainty while also
102 preserving any attempt at preventing unbounded user fragmentation.

104 It is not impossible to imagine a long-lived process that makes many mappings
105 being in a position where mappings later in its life are 100% predictable
106 given this implementation, however. In fact, I think it is fairly likely.

108 I think the most at risk would be a process which makes many mappings, and
109 uses 'dlopen(3c)' at unpredictable times (rather than, as is most common, during
110 setup). Dynamic objects tend to have strict (and high) alignment requirements
111 which in such a process are likely to only be fulfillable at a single location
112 in the address space gap we choose, and thus be entirely predictable.

114 ### Randomisation of executable base addresses requires PIE

116 To randomise the executable base address, we need position independent
117 executables, which appears like it would turn into a whole separate project
118 (though perhaps not too large a project). That code isn't here, so the
119 executable base is fixed.

121 This means that rather than return-to-libc one could return-to-executable
122 trivially and successfully, I think. (That would include using the
123 executable's PLT to vector yourself to libc. Should the executable have a PLT
124 entry for something useful to you).

new/./README-ASLR.md

3

```
125 #endif /* ! codereview */
```

new/exception_lists/check_rtime

1

```

*****
10033 Thu Jun 30 22:01:06 2016
new/exception_lists/check_rtime
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
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 #
23 # Copyright (c) 2009, 2010, Oracle and/or its affiliates. All rights reserved.
24 # Copyright 2015, OmniTI Computer Consulting, Inc. All rights reserved.
25 #
26 #
27 # This file provides exceptions to the usual rules applied to ELF objects by
28 # check_rtime. All strings are Perl regular expressions that are compared to
29 # file paths. In addition to the standard Perl syntax, there is one extension:
30 #
31 #     MACH(dir)
32 #
33 # is expanded into a regular expression that matches the given
34 # directory, or a 64-bit subdirectory of the directory with the
35 # name of a 64-bit architecture. For example, MACH(lib) will match
36 # any of the following:
37 #
38 #     lib
39 #     lib/amd64
40 #     lib/sparcv9
41 #
42 #
43 # Directory hierarchies to skip completely
44 SKIP      ^usr/lib/libc/          # optimized libc
45 SKIP      ^usr/lib/rcm/          # 4426119
46 SKIP      ^usr/perl5/          # alan's taking care of these :-
47 SKIP      ^usr/src/            # no objects in source code
48 #
49 # Individual files that we don't examine
50 SKIP      ^boot/grub/bin/grub$
51 # USIII specific extns. cause ldd noise on USII bld. m/c
52 SKIP      ^usr/lib/fps/sun4u/UltraSPARC.*/*fptest$
53 SKIP      ^usr/MACH(lib)/lddstub$ # lddstub has no dependencies
54 SKIP      ^usr/MACH(lib)/libssagent\.so\.1$ # 4328854
55 SKIP      ^usr/lib/MACH(iconv)/geniconvtbl.so$ # 4384329
56 #
57 # picl file exclusions (4385799)
58 SKIP      ^usr/platform/.*/*libpsvcplugin_psr\.so\.1

```

new/exception_lists/check_rtime

2

```

59 SKIP      ^usr/platform/.*/*libpsvcpolicy_psr\.so\.1
60 SKIP      ^usr/platform/.*/*libpsvcpolicy\.so\.1
61 SKIP      ^usr/lib/syseven/modules/picl_slm.so$
62 #
63 # Objects that are allowed to have executable data segments
64 EXEC_DATA ^MACH(lib)/ld\.so\.1$
65 EXEC_DATA ^lib/libc\.so\.1$ # 6524709, 32-bit, needed for x86 only
66 EXEC_DATA ^lib/amd64/libumem\.so\.1$ # ptcumem
67 EXEC_DATA ^lib/libumem\.so\.1$ # ptcumem
68 EXEC_DATA ^opt/SUNWdtrt/tst/.*/*ustack/tst\.helper\.exe$
69 EXEC_DATA ^platform/.*/*MACH(kernel)/unix$
70 EXEC_DATA ^platform/.*/*multiboot$
71 #
72 # Objects that are allowed to have an executable stack
73 EXEC_STACK ^platform/.*/*MACH(kernel)/unix$
74 EXEC_STACK ^platform/.*/*multiboot$
75 EXEC_STACK ^opt/os-tests/tests/secflags/stacky$
76 #endif /* ! codereview */
77 #
78 # Objects for which we allow relocations to the text segment
79 TEXTREL    ^platform/.*/*MACH(kernel)/unix$
80 #
81 # Directories and files that are allowed to have no direct bound symbols
82 NODIRECT   ^platform/.*/*MACH(kernel)/unix$
83 NODIRECT   ^usr/ucb
84 NODIRECT   ^usr/4lib/sbcp$
85 #
86 # Identify any files that should be skipped when building a crle(1)
87 # configuration file. As the hwcap libraries can be loop-back mounted onto
88 # libc, these can confuse crle(1) because of their identical dev/inode.
89 NOCLEALT   ^usr/lib/libc/libc_hwcap[1-3].so.1$
90 #
91 # Files that should contain debugging information.
92 STAB       ^platform/.*/*MACH(kernel)/unix$
93 #
94 # Files that are allowed undefined references
95 UNDEF_REF  ^usr/lib/libnisdb\.so\.2$
96 UNDEF_REF  ^usr/snadm/lib/libsvm\.so\.1$
97 #
98 # Objects allowed to have unused dependencies
99 UNUSED_DEPS ^usr/lib/picl/plugins/ # require devtree dependencies
100 #
101 # libm.so.2 dependency
102 UNUSED_OBJ unused object=.*MACH(libm)/libm_hwcap1\.so\.2
103 #
104 # libnetsnmphelpers.so is empty in some net-snmp versions
105 UNUSED_OBJ unused object=.*/*libnetsnmphelpers\.so\.1$
106 UNREF_OBJ  unreferenced object=.*/*libnetsnmphelpers\.so\.1$
107 #
108 # Unused runpaths due to dlopen() use
109 UNUSED_RPATH /usr/lib/fs/autofs.*\ from\ .\automountd
110 UNUSED_RPATH /etc/ppp/plugins.*\ from\ .\pppd
111 UNUSED_RPATH /usr/lib/inet/ppp.*\ from\ .\pppd
112 UNUSED_RPATH /usr/sfw/lib.*\ from\ .*libipsecutil\.so\.1
113 UNUSED_RPATH /usr/platform/.*rsmlib.*\ from\ .*librsm\.so\.2
114 UNUSED_RPATH $ORIGIN.*\ from\ .*fcode.so
115 UNUSED_RPATH /opt/VRTSvxvm/lib.*\ from\ .*libdiskmgt\.so\.1
116 #
117 # Unused runpaths in picl code
118 UNUSED_RPATH /usr/platform/.*\ from\ .*/*usr/platform
119 UNUSED_RPATH /usr/lib/picl.*\ from\ .*/*usr/platform
120 UNUSED_RPATH /usr/platform/.*\ from\ .*/*usr/lib/picl
121 #
122 # Unused runpaths in non-OSNET objects we can't change
123 UNUSED_RPATH /usr/lib/mps.*\ from\ .*libnss3\.so
124 UNUSED_RPATH /usr/lib/mps.*\ from\ .*libnssutil3\.so

```

```

125 UNUSED_RPATH /usr/lib/mps.*\ from\ .*libsime3\so
126 UNUSED_RPATH /usr/lib/mps.*\ from\ .*libssl3\so
127 UNUSED_RPATH /usr/sfw/lib.*\ from\ .*libdbus-1\so.3
128 UNUSED_RPATH /usr/sfw/lib.*\ from\ .*libdbus-glib-1\so.2
129 UNUSED_RPATH /usr/sfw/lib.*\ from\ .*libglib-2\0\so.0
130 UNUSED_RPATH /usr/X11/lib.*\ from\ .*libglib-2\0\so.0
131 UNUSED_RPATH /usr/sfw/lib.*\ from\ .*libgobject-2\0\so.0
132 UNUSED_RPATH /usr/X11/lib.*\ from\ .*libgobject-2\0\so.0
133 UNUSED_RPATH /usr/sfw/lib.*\ from\ .*libgthread-2\0\so.0
134 UNUSED_RPATH /usr/X11/lib.*\ from\ .*libgthread-2\0\so.0
135 UNUSED_RPATH /usr/sfw/lib.*\ from\ .*libcrypto\so.0\9\8
136 UNUSED_RPATH /usr/sfw/lib.*\ from\ .*libnetsnmp\so...
137 UNUSED_RPATH /usr/sfw/lib.*\ from\ .*libgcc_s\so.1
138 UNUSED_RPATH /usr/ccs/lib.*\ from\ .*libgcc_s\so.1
139 UNUSED_RPATH /usr/lib.*\ from\ .*libgcc_s\so.1
140 UNUSED_RPATH /usr/postgres/8.3/lib.*\ from\ .*libpq\so.5
141 UNUSED_RPATH /usr/sfw/lib.*\ from\ .*libpq\so.5
142 UNUSED_RPATH /usr/lib.*\ from\ ./usr/lib/mps
143 UNUSED_RPATH /usr/ccs/lib.*\ from\ ./usr/lib/mps
144 UNUSED_RPATH /usr/gnu/lib.*\ from\ ./usr/lib/libpython2\6
145 UNUSED_RPATH /usr/gnu/lib.*\ from\ ./usr/lib/64/libpython2\6
146 UNUSED_RPATH /usr/snadm/lib.*\ from\ ./usr/snadm/lib/libspmicommon\so.1

149 # Unused runpaths for reasons not captured above
150 UNUSED_RPATH /usr/lib/smbstr.*\ from\ .*libsmb\so.1 # future needs
151 UNUSED_RPATH /usr.*\ from\ .*tst\gcc.exe # gcc built

154 # Unreferenced objects of non-OSnet objects we can't change
155 UNREF_OBJ /lib.*\ of\ .*libcimapi\so
156 UNREF_OBJ /lib.*\ of\ .*libdbus-1\so.3
157 UNREF_OBJ /lib.*\ of\ .*libdbus-glib-1\so.2
158 UNREF_OBJ /lib.*\ of\ .*libgio-2.0\so.0
159 UNREF_OBJ /lib.*\ of\ .*libglib-2.0\so.0
160 UNREF_OBJ /lib.*\ of\ .*libgobject-2.0\so.0
161 UNREF_OBJ /lib.*\ of\ .*libgthread-2\0\so.0
162 UNREF_OBJ /lib.*\ of\ .*libjvm\so
163 UNREF_OBJ /lib.*\ of\ .*libnetsnmp\so...
164 UNREF_OBJ /lib.*\ of\ .*libnetsnmpagent\so...
165 UNREF_OBJ /lib.*\ of\ .*libnetsnmpmibs\so...
166 UNREF_OBJ /lib.*\ of\ .*libnetsnmphelpers\so...
167 UNREF_OBJ /lib.*\ of\ .*libnspr4\so
168 UNREF_OBJ /lib.*\ of\ .*libpq\so.5
169 UNREF_OBJ /lib.*\ of\ .*libsoftokn3\so
170 UNREF_OBJ /lib.*\ of\ .*libspmicommon\so.1
171 UNREF_OBJ /lib.*\ of\ .*libspmicommon\so.1
172 UNREF_OBJ /lib.*\ of\ .*libssl3\so
173 UNREF_OBJ /lib.*\ of\ .*libtspi\so.1
174 UNREF_OBJ /lib.*\ of\ .*libxml2\so.2
175 UNREF_OBJ /lib.*\ of\ .*libxslt\so.1
176 UNREF_OBJ /lib.*\ of\ .*libpq\so.4
177 UNREF_OBJ /lib.*\ of\ .*libpython2\4\so.1\0
178 UNREF_OBJ /lib.*\ of\ .*libpython2\6\so.1\0
179 UNREF_OBJ /libgcc_s.*\ of\ .*libstdc\+\+\so.6
180 UNREF_OBJ /libgcc_s.*\ of\ .*libgmodule-2\0\so.0

182 # Unreferenced object of objects we can't change for other reasons
183 UNREF_OBJ /libmapallocc\so.1;\ unused\ dependency\ of # interposer
184 UNREF_OBJ /libstdc\+\+\so.6;\ unused\ dependency\ of # gcc build
185 UNREF_OBJ /libgcc_s\so.1.*\ of\ .*libstdc\+\+\so.6 # omnios gcc mix
186 UNREF_OBJ /libm\so.2.*\ of\ .*libstdc\+\+\so.6 # gcc build
187 UNREF_OBJ /lib.*\ of\ .*lib/picl/plugins/ # picl
188 UNREF_OBJ /lib.*\ of\ .*kcfid # interposer
189 UNREF_OBJ /libpkcs11\so.1; .*\ of\ .*libkmf\so.1 # interposed
190 # Referenced by the Studio build, not the GCC build. GCC eliminates the unused

```

```

191 # statics which have the dependence.
192 UNREF_OBJ /libc\so.1.*\ of\ .*kldap\so.1

195 # Objects that used to contain system functionality that has since
196 # migrated to libc. We preserve these libraries as pure filters for
197 # backward compatibility but nothing needs to link to them.
198 OLDDEP libaio\so.1 # onnv build 44
199 OLDDEP libdl\so.1 # on10 build 49
200 OLDDEP libdoor\so.1 # onnv build 12
201 OLDDEP libintl\so.1 # on297 build 7
202 OLDDEP libpthread\so.1 # on10 build 53
203 OLDDEP librt\so.1 # onnv build 44
204 OLDDEP libsched\so.1 # on10 build 36
205 OLDDEP libthread\so.1 # on10 build 53
206 OLDDEP libw\so.1 # on297 build 7

208 # Files for which we skip checking of duplicate addresses in the
209 # symbol sort sections. Such exceptions should be rare --- most code will
210 # not have duplicate addresses, since it takes assembler or a "#pragma weak"
211 # to do such aliasing in C. C++ is different: The compiler generates aliases
212 # for implementation reasons, and the mangled names used to encode argument
213 # and return value types are difficult to handle well in mapfiles.
214 # Furthermore, the Sun compiler and gcc use different and incompatible
215 # name mangling conventions. Since ON must be buildable by either, we
216 # would have to maintain two sets of mapfiles for each such object.
217 # C++ use is rare in ON, so this is not worth pursuing.
218 #
219 NOSYMSORT opt/SUNWdtrt/tst/common/pid/tst.weak2.exe # DTrace test
220 NOSYMSORT lib/amd64/libns1\so.1 # C++
221 NOSYMSORT lib/sparcv9/libns1\so.1 # C++
222 NOSYMSORT lib/sparcv9/libfru\so.1 # C++
223 NOSYMSORT usr/lib/lms # C++
224 NOSYMSORT ld\so.1 # libc_pic.a use
225 NOSYMSORT lib/libsun_fc\so.1 # C++
226 NOSYMSORT lib/amd64/libsun_fc\so.1 # C++
227 NOSYMSORT lib/sparcv9/libsun_fc\so.1 # C++
228 NOSYMSORT usr/lib/amd64/libfru\so.1 # C++

231 # The libprtdiag_psr.so.1 objects built under usr/src/lib/libprtdiag_psr
232 # are a family, all built using the same makefile, targeted at different
233 # sparc hardware variants. There are a small number of cases where this
234 # one size fits all approach causes an object to be linked against an
235 # unneeded library.
236 UNREF_OBJ lib/(libdevinfo|libcfgadm)\so.1; .*\ of\ .*SUNW,Netra-CP2300/1

```

new/exception_lists/manlint

1

```
*****  
183 Thu Jun 30 22:01:07 2016  
new/exception_lists/manlint  
7029 want per-process exploit mitigation features (secflags)  
7030 want basic address space layout randomization (aslr)  
7031 noexec_user_stack should be a secflag  
7032 want a means to forbid mappings around NULL.  
*****  
1 # Not actually a manual page  
2 usr/src/cmd/svc/dtd/service_bundle.dtd.1  
3 usr/src/lib/libbsm/adt_record.dtd.1  
4 usr/src/lib/libbsm/adt_record.xsl.1  
5 usr/src/lib/libzonecfg/dtd/zonecfg.dtd.1  
6 #endif /* ! codereview */
```

```

*****
37484 Thu Jun 30 22:01:08 2016
new/usr/src/cmd/auditreduce/token.c
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
_____unchanged_portion_omitted_____

1943 /*
1944 * Format of security flags token:
1945 *   security flag set      string      string
1946 *   security flags        string
1947 */

1949 int
1950 secflags_token(adr_t *adr)
1951 {
1952     skip_string(adr);      /* set name */
1953     skip_string(adr);      /* security flags */
1954     return (-1);
1955 }

1957 /*
1958 #endif /* ! codereview */
1959 * Format of label token:
1960 *   label ID                1 byte
1961 *   compartment length     1 byte
1962 *   classification         2 bytes
1963 *   compartment words     <compartment length> * 4 bytes
1964 */
1965 int
1966 label_token(adr_t *adr)
1967 {
1968     static m_label_t *label = NULL;
1969     static size32_t l_size;
1970     int len;

1972     if (label == NULL) {
1973         label = m_label_alloc(MAC_LABEL);
1974         l_size = blabel_size() - 4;
1975     }

1977     if (label == NULL) {
1978         /* out of memory, should never happen; skip label */
1979         char l;          /* length */

1981         adr->adr_now += sizeof (char);
1982         adrm_char(adr, (char *)&l, 1);
1983         adr->adr_now += sizeof (short) + (4 * 1);
1984         return (-1);
1985     }

1987     adrm_char(adr, (char *)label, 4);
1988     len = (int)(((char *)label)[1] * 4);
1989     if (len > l_size) {
1990         return (-1);
1991     }
1992     adrm_char(adr, &((char *)label)[4], len);

1994     if (flags & M_LABEL) {
1995         if (blinrange(label, m_label))
1996             checkflags = checkflags | M_LABEL;
1997     }

```

```

1999         return (-1);
2000     }

2003 /*
2004 * Format of useofpriv token:
2005 *   success/failure      adr_char
2006 *   privilege(s)        adr_string
2007 */
2008 /* ARGSUSED */
2009 int
2010 useofpriv_token(adr_t *adr)
2011 {
2012     char flag;

2014     adrm_char(adr, &flag, 1);
2015     skip_string(adr);
2016     return (-1);
2017 }

```

new/usr/src/cmd/praudit/praudit.xcl

1

```
*****
4255 Thu Jun 30 22:01:09 2016
new/usr/src/cmd/praudit/praudit.xcl
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
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 2010 Sun Microsystems, Inc. All rights reserved.
23 # Use is subject to license terms.
24 #
25 msgid ", "
26 msgstr ""
27 msgid ""
28 msgstr ""
29 msgid "r"
30 msgstr ""
31 msgid "stdin"
32 msgstr ""
33 msgid "%- "
34 msgstr ""
35 msgid "%s"
36 msgstr ""
37 msgid " "
38 msgstr ""
39 msgid "%d"
40 msgstr ""
41 msgid "s"
42 msgstr ""
43 msgid "\n"
44 msgstr ""
45 msgid "%s%s%d%s"
46 msgstr ""
47 msgid "%s\n"
48 msgstr ""
49 msgid "%d %d %s"
50 msgstr ""
51 msgid "%d%s"
52 msgstr ""
53 msgid ":"
54 msgstr ""
55 msgid "%o"
56 msgstr ""
57 msgid "0x%x"
58 msgstr ""
```

new/usr/src/cmd/praudit/praudit.xcl

2

```
59 msgid "%hd"
60 msgstr ""
61 msgid "%"
62 msgid "llo"
63 msgstr ""
64 msgid "% "
65 msgid "lld"
66 msgstr ""
67 msgid "0x%"
68 msgid "llx"
69 msgstr ""
70 msgid "0x"
71 msgstr ""
72 msgid "%02x"
73 msgstr ""
74 msgid "%s%d"
75 msgstr ""
76 msgid "%u"
77 msgstr ""
78 msgid "%"
79 msgid "llu"
80 msgstr ""
81 msgid "%c"
82 msgstr ""
83 msgid "0x%xh"
84 msgstr ""
85 msgid "%ho"
86 msgstr ""
87 msgid " %s=\" "
88 msgstr ""
89 msgid "<%s>"
90 msgstr ""
91 msgid "<%s"
92 msgstr ""
93 msgid "</%s>"
94 msgstr ""
95 msgid "/>"
96 msgstr ""
97 msgstr ""
98 msgid "%C"
99 msgstr ""
100 msgid "%*s"
101 msgid "<?xml version='1.0' encoding='UTF-8' ?>\n"
102 msgstr ""
103 msgid "\n<!DOCTYPE audit PUBLIC "
104 "'-//Sun Microsystems, Inc.//DTD Audit V1//EN' "
105 "'file:///usr/share/lib/xml/dtd/adt_record.dtd.1'>\n\n"
106 msgstr ""
107 msgid "<?xml-stylesheet type='text/xsl' "
108 "href='file:///usr/share/lib/xml/style/adt_record.xsl.1' ?>\n"
109 msgstr ""
110 msgid "<audit>"
111 msgstr ""
112 msgid "%s%s%s%s"
113 msgstr ""
114 msgid "\n</audit>\n"
115 msgstr ""
116 msgid "old_header"
117 msgstr ""
118 msgid "record"
119 msgstr ""
120 msgid "ip_address"
121 msgstr ""
122 msgid "ip_port"
123 msgstr ""
124 msgid "old_socket"
```

```
125 msgstr
126 msgid "IPC_perm"
127 msgstr
128 msgid "cmw_label"
129 msgstr
130 msgid "information_label"
131 msgstr
132 msgid "sensitivity_label"
133 msgstr
134 msgid "use_of_privilege"
135 msgstr
136 msgid "use_of_authorization"
137 msgstr
138 msgid "X_atom"
139 msgstr
140 msgid "X_object"
141 msgstr
142 msgid "X_protocol"
143 msgstr
144 msgid "X_selection"
145 msgstr
146 msgid "X_color_map"
147 msgstr
148 msgid "X_cursor"
149 msgstr
150 msgid "X_font"
151 msgstr
152 msgid "X_graphic_context"
153 msgstr
154 msgid "X_pixmap"
155 msgstr
156 msgid "X_property"
157 msgstr
158 msgid "X_window"
159 msgstr
160 msgid "X_client"
161 msgstr
162 msgid "audit-uid"
163 msgstr
164 msgid "uid"
165 msgstr
166 msgid "gid"
167 msgstr
168 msgid "ruid"
169 msgstr
170 msgid "rgid"
171 msgstr
172 msgid "pid"
173 msgstr
174 msgid "sid"
175 msgstr
176 msgid "tid"
177 msgstr
178 msgid "event-modifier"
179 msgstr
180 msgid "modifier"
181 msgstr
182 msgid "event-type"
183 msgstr
184 msgid "event"
185 msgstr
186 msgid "token-version"
187 msgstr
188 msgid "version"
189 msgstr
190 msgid "time"
```

```
191 msgstr
192 msgid "msec"
193 msgstr
194 msgid "errval"
195 msgstr
196 msgid "retval"
197 msgstr
198 msgid "set-type"
199 msgstr
200 msgid "xid"
201 msgstr
202 msgid "xcreator-uid"
203 msgstr
204 msgid "x_sel_text"
205 msgstr
206 msgid "x_sel_type"
207 msgstr
208 msgid "x_sel_data"
209 msgstr
210 msgid "arg-num"
211 msgstr
212 msgid "value"
213 msgstr
214 msgid "desc"
215 msgstr
216 msgid "mode"
217 msgstr
218 msgid "fsid"
219 msgstr
220 msgid "nodeid"
221 msgstr
222 msgid "device"
223 msgstr
224 msgid "seq-num"
225 msgstr
226 msgid "argv"
227 msgstr
228 msgid "arge"
229 msgstr
230 msgid "arg"
231 msgstr
232 msgid "env"
233 msgstr
234 msgid "result"
235 msgstr
236 msgid "creator-uid"
237 msgstr
238 msgid "creator-gid"
239 msgstr
240 msgid "seq"
241 msgstr
242 msgid "key"
243 msgstr
244 msgid "service_type"
245 msgstr
246 msgid "len"
247 msgstr
248 msgid "id"
249 msgstr
250 msgid "offset"
251 msgstr
252 msgid "time_to_live"
253 msgstr
254 msgid "protocol"
255 msgstr
256 msgid "cksum"
```



```
257 msgstr
258 msgid "src_addr"
259 msgstr
260 msgid "dest_addr"
261 msgstr
262 msgid "type"
263 msgstr
264 msgid "port"
265 msgstr
266 msgid "addr"
267 msgstr
268 msgid "sock_domain"
269 msgstr
270 msgid "sock_type"
271 msgstr
272 msgid "lport"
273 msgstr
274 msgid "laddr"
275 msgstr
276 msgid "fport"
277 msgstr
278 msgid "faddr"
279 msgstr
280 msgid "ipc-type"
281 msgstr
282 msgid "ipc-id"
283 msgstr
284 msgid "print"
285 msgstr
286 msgid "count"
287 msgstr
288 msgid "fmri"
289 msgstr
290 msgid "user"
291 msgstr
292 msgid "secflags"
293 msgstr
294 #endif /* ! codereview */
```

new/usr/src/cmd/praudit/token.c

1

```
*****
64769 Thu Jun 30 22:01:09 2016
new/usr/src/cmd/praudit/token.c
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
_____unchanged_portion_omitted_____
```

```
2350 /*
2351 * -----
2352 * secflags_token()      : Process privilege token and display contents
2353 * return codes         : -1 - error
2354 *                      : 0 - successful
2355 * NOTE: At the time of call, the secflags token id has been retrieved
2356 *
2357 * Format of secflags token:
2358 *   secflags token id      adr_char
2359 *   secflag set name      adr_string
2360 *   secflags               adr_string
2361 * -----
2362 */
2363 int
2364 secflags_token(pr_context_t *context)
2365 {
2366     int    returnstat;
2367
2368     /* Set name */
2369     returnstat = process_tag(context, TAG_SETTYPE, 0, 0);
2370
2371     /* Done with attributes; force end of token open */
2372     if (returnstat == 0)
2373         returnstat = finish_open_tag(context);
2374
2375     /* set */
2376     return (pa_adr_string(context, returnstat, 1));
2377 }
2378 #endif /* ! codereview */
```

```

*****
12427 Thu Jun 30 22:01:10 2016
new/usr/src/cmd/praudit/toktable.c
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
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 2010 Sun Microsystems, Inc. All rights reserved.
23 * Use is subject to license terms.
24 */

26 /*
27  * Solaris Audit Token Table.
28  */

30 #include <locale.h>

32 #include <stdio.h>
33 #include <stdlib.h>
34 #include <string.h>
35 #include <bsm/audit.h>
36 #include <bsm/audit_record.h>
37 #include <bsm/libbsm.h>

39 #include "praudit.h"
40 #include "toktable.h"

42 token_desc_t tokentable[MAXTAG + 1];

44 #define table_init(i, n, f, t) \
45     tokentable[(int)(i)].t_name = (n); \
46     tokentable[(int)(i)].t_tagname = (n); \
47     tokentable[(int)(i)].func = (f); \
48     tokentable[(int)(i)].t_type = (t);

50 /* table_initx is for entries which need name different from tagname */
51 #define table_initx(i, n, tn, f, t) \
52     tokentable[(int)(i)].t_name = (n); \
53     tokentable[(int)(i)].t_tagname = (tn); \
54     tokentable[(int)(i)].func = (f); \
55     tokentable[(int)(i)].t_type = (t);

57 /*
58  * Initialize the table of tokens & other tags.

```

```

59 */
60 void
61 init_tokens(void)
62 {
63     /*
64      * TRANSLATION_NOTE
65      * These names refer to different type of audit tokens.
66      * To gain a better understanding of each token, read
67      * System Administration Guide: Security Services >> Solaris Auditing
68      * at http://docs.sun.com.
69      */

71     (void) gettext("file"); /* to force out the translation note */

73     /*
74      * Control token types
75      */

77     table_init(AUT_INVALID, (char *)0, NOFUNC, T_UNKNOWN);
78     table_init(AUT_OTHER_FILE32, "file", file_token, T_EXTENDED);
79     table_init(AUT_OHEADER, "old_header", NOFUNC, T_EXTENDED);
80     table_init(AUT_TRAILER, "trailer", trailer_token, T_UNKNOWN);
81     table_initx(AUT_HEADER32, "header", "record",
82                header_token, T_EXTENDED);
83     table_initx(AUT_HEADER32_EX, "header", "record",
84                header32_ex_token, T_EXTENDED);

86     /*
87      * Data token types
88      */

90     table_init(AUT_DATA, "arbitrary", arbitrary_data_token, T_EXTENDED);
91     table_init(AUT_FMRI, "fmri", fmri_token, T_ELEMENT);
92     table_init(AUT_IPC, "IPC", s5_IPC_token, T_ENCLOSED);
93     table_init(AUT_PATH, "path", path_token, T_ELEMENT);
94     table_init(AUT_XATPATH, "path_attr", path_attr_token, T_ELEMENT);
95     table_init(AUT_SUBJECT32, "subject", subject32_token, T_ENCLOSED);
96     table_init(AUT_PROCESS32, "process", process32_token, T_ENCLOSED);
97     table_init(AUT_RETURN32, "return", return_value32_token, T_ENCLOSED);
98     table_init(AUT_TEXT, "text", text_token, T_ELEMENT);
99     table_init(AUT_OPAQUE, "opaque", opaque_token, T_ELEMENT);
100    table_initx(AUT_IN_ADDR, "ip address", "ip_address",
101               ip_addr_token, T_ELEMENT);
102    table_init(AUT_IP, "ip", ip_token, T_ENCLOSED);
103    table_initx(AUT_IPORT, "ip port", "ip_port",
104               iport_token, T_ELEMENT);
105    table_init(AUT_ARG32, "argument", argument32_token, T_ENCLOSED);
106    table_initx(AUT_SOCKET, "socket", "old_socket",
107               socket_token, T_ENCLOSED);
108    table_init(AUT_SEQ, "sequence", sequence_token, T_ENCLOSED);

110    /*
111     * Modifier token types
112     */

114    table_init(AUT_ACL, "acl", acl_token, T_ENCLOSED);
115    table_init(AUT_ACE, "acl", ace_token, T_ENCLOSED);
116    table_init(AUT_ATTR, "attribute", attribute_token, T_ENCLOSED);
117    table_init(AUT_IPC_PERM, "IPC_perm", s5_IPC_perm_token, T_ENCLOSED);
118    table_init(AUT_GROUPS, "group", group_token, T_ELEMENT);
119    table_initx(AUT_LABEL, "sensitivity label", "sensitivity_label",
120               label_token, T_ELEMENT);
121    table_init(AUT_PRIV, "privilege", privilege_token, T_EXTENDED);
122    table_init(AUT_SECFLAGS, "secflags", secflags_token, T_EXTENDED);
123 #endif /* ! codereview */
124    table_initx(AUT_UPRIV, "use of privilege", "use_of_privilege",

```

```

125     useofpriv_token, T_EXTENDED);
126     table_init(AUT_LIAISON, "liaison", liaison_token, T_ELEMENT);
127     table_init(AUT_NEWGROUPS, "group", newgroup_token, T_ELEMENT);
128     table_init(AUT_EXEC_ARGS, "exec_args", exec_args_token, T_ELEMENT);
129     table_init(AUT_EXEC_ENV, "exec_env", exec_env_token, T_ELEMENT);
130     table_init(AUT_ATTR32, "attribute", attribute32_token, T_ENCLOSED);
131     table_initx(AUT_UAUTH, "use of authorization",
132     "use_of_authorization", useofauth_token, T_ELEMENT);
133     table_init(AUT_USER, "user", user_token, T_ENCLOSED);
134     table_init(AUT_ZONENAME, "zone", zonename_token, T_ENCLOSED);

136     /*
137     * X windows token types
138     */
139     table_initx(AUT_XATOM, "X atom", "X_atom", xatom_token, T_ELEMENT);
140     table_initx(AUT_XOBJ, "X object", "X_object", NOFUNC, T_UNKNOWN);
141     table_initx(AUT_XPROTO, "X protocol", "X_protocol", NOFUNC, T_UNKNOWN);
142     table_initx(AUT_XSELECT, "X selection", "X_selection",
143     xselect_token, T_ELEMENT);
144     table_initx(AUT_XCOLORMAP, "X color map", "X_color_map",
145     xcolormap_token, T_ENCLOSED);
146     table_initx(AUT_XCURSOR, "X cursor", "X_cursor",
147     xcursor_token, T_ENCLOSED);
148     table_initx(AUT_XFONT, "X font", "X_font", xfont_token, T_ENCLOSED);
149     table_initx(AUT_XGIC, "X graphic context", "X_graphic_context",
150     xgc_token, T_ENCLOSED);
151     table_initx(AUT_XPIXMAP, "X pixmap", "X_pixmap",
152     xpixmap_token, T_ENCLOSED);
153     table_initx(AUT_XPROPERTY, "X property", "X_property",
154     xproperty_token, T_EXTENDED);
155     table_initx(AUT_XWINDOW, "X window", "X_window",
156     xwindow_token, T_ENCLOSED);
157     table_initx(AUT_XCLIENT, "X client", "X_client",
158     xclient_token, T_ELEMENT);

160     /*
161     * Command token types
162     */

164     table_init(AUT_CMD, "cmd", cmd_token, T_ELEMENT);
165     table_init(AUT_EXIT, "exit", exit_token, T_ENCLOSED);

167     /*
168     * Miscellaneous token types
169     */

171     table_init(AUT_HOST, "host", host_token, T_ELEMENT);

173     /*
174     * Solaris64 token types
175     */

177     table_init(AUT_ARG64, "argument", argument64_token, T_ENCLOSED);
178     table_init(AUT_RETURN64, "return", return_value64_token, T_ENCLOSED);
179     table_init(AUT_ATTR64, "attribute", attribute64_token, T_ENCLOSED);
180     table_initx(AUT_HEADER64, "header", "record",
181     header64_token, T_EXTENDED);
182     table_init(AUT_SUBJECT64, "subject", subject64_token, T_ENCLOSED);
183     table_init(AUT_PROCESS64, "process", process64_token, T_ENCLOSED);
184     table_init(AUT_OTHER_FILE64, "file", file64_token, T_EXTENDED);

186     /*
187     * Extended network address token types
188     */

190     table_initx(AUT_HEADER64_EX, "header", "record",

```

```

191     header64_ex_token, T_EXTENDED);
192     table_init(AUT_SUBJECT32_EX, "subject", subject32_ex_token, T_ENCLOSED);
193     table_init(AUT_PROCESS32_EX, "process", process32_ex_token, T_ENCLOSED);
194     table_init(AUT_SUBJECT64_EX, "subject", subject64_ex_token, T_ENCLOSED);
195     table_init(AUT_PROCESS64_EX, "process", process64_ex_token, T_ENCLOSED);
196     table_initx(AUT_IN_ADDR_EX, "ip address", "ip_address",
197     ip_addr_ex_token, T_ELEMENT);
198     table_init(AUT_SOCKET_EX, "socket", socket_ex_token, T_ENCLOSED);
199     table_init(AUT_TID, "tid", tid_token, T_EXTENDED);

201 #ifdef _PRAUDIT
202     /*
203     * Done with tokens above here. Now do remaining tags.
204     */
205     table_init(TAG_AUID, "audit-uid", pa_pw_uid, T_ATTRIBUTE);
206     table_init(TAG_UID, "uid", pa_pw_uid, T_ATTRIBUTE);
207     table_init(TAG_GID, "gid", pa_gr_uid, T_ATTRIBUTE);
208     table_init(TAG_RUID, "ruid", pa_pw_uid, T_ATTRIBUTE);
209     table_init(TAG_RGID, "rgid", pa_gr_uid, T_ATTRIBUTE);

211     table_init(TAG_PID, "pid", pa_adr_u_int32, T_ATTRIBUTE);
212     table_init(TAG_SID, "sid", pa_adr_u_int32, T_ATTRIBUTE);

214     table_init(TAG_TID32, "tid", pa_tid32, T_ATTRIBUTE);
215     table_init(TAG_TID64, "tid", pa_tid64, T_ATTRIBUTE);
216     table_init(TAG_TID32_EX, "tid", pa_tid32_ex, T_ATTRIBUTE);
217     table_init(TAG_TID64_EX, "tid", pa_tid64_ex, T_ATTRIBUTE);
218     table_init(TAG_TID_TYPE, "type", NOFUNC, T_ATTRIBUTE);
219     table_init(TAG_IP, "ipadr", NOFUNC, T_ENCLOSED);
220     table_init(TAG_IP_LOCAL, "local-port", pa_adr_u_short, T_ATTRIBUTE);
221     table_init(TAG_IP_REMOTE, "remote-port", pa_adr_u_short, T_ATTRIBUTE);
222     table_init(TAG_IP_ADR, "host", pa_ip_addr, T_ATTRIBUTE);

224     table_initx(TAG_EVMOD, "event-modifier", "modifier",
225     pa_event_modifier, T_ATTRIBUTE);
226     table_initx(TAG_EVTTYPE, "event-type", "event",
227     pa_event_type, T_ATTRIBUTE);
228     table_initx(TAG_TOKVERS, "token-version", "version",
229     pa_adr_byte, T_ATTRIBUTE);

231     table_init(TAG_ISO, "iso8601", NOFUNC, T_ATTRIBUTE);

233     table_init(TAG_ERRVAL, "errval", NOFUNC, T_ATTRIBUTE);
234     table_init(TAG_RETVAL, "retval", pa_adr_int32, T_ATTRIBUTE);

236     table_init(TAG_SETTYPE, "set-type", pa_adr_string, T_ATTRIBUTE);
237     /* Sub-element of groups & newgroups token: */
238     table_init(TAG_GROUPID, "gid", pa_gr_uid, T_ELEMENT);

240     table_init(TAG_XID, "xid", pa_xid, T_ATTRIBUTE);
241     table_init(TAG_XCUID, "xcreator-uid", pa_pw_uid, T_ATTRIBUTE);

243     table_init(TAG_XSELTEXT, "x_sel_text", pa_adr_string, T_ELEMENT);
244     table_init(TAG_XSELTTYPE, "x_sel_type", pa_adr_string, T_ELEMENT);
245     table_init(TAG_XSELDATA, "x_sel_data", pa_adr_string, T_ELEMENT);

247     table_init(TAG_ARGNUM, "arg-num", pa_adr_byte, T_ATTRIBUTE);
248     table_init(TAG_ARGVAL32, "value", pa_adr_int32hex, T_ATTRIBUTE);
249     table_init(TAG_ARGVAL64, "value", pa_adr_int64hex, T_ATTRIBUTE);
250     table_init(TAG_ARGDESC, "desc", pa_adr_string, T_ATTRIBUTE);

252     table_init(TAG_MODE, "mode", pa_mode, T_ATTRIBUTE);
253     table_init(TAG_FSID, "fsid", pa_adr_int32, T_ATTRIBUTE);
254     table_init(TAG_NODEID32, "nodeid", pa_adr_int32, T_ATTRIBUTE);
255     table_init(TAG_NODEID64, "nodeid", pa_adr_int64, T_ATTRIBUTE);
256     table_init(TAG_DEVICE32, "device", pa_adr_u_int32, T_ATTRIBUTE);

```

```
257     table_init(TAG_DEVICE64, "device", pa_adr_u_int64, T_ATTRIBUTE);

259     table_init(TAG_SEQNUM, "seq-num", pa_adr_u_int32, T_ATTRIBUTE);
260     table_init(TAG_ZONENAME, "name", pa_adr_string, T_ATTRIBUTE);
261     table_init(TAG_ARGV, "argv", pa_cmd, T_ELEMENT);
262     table_init(TAG_ARGE, "arge", pa_cmd, T_ELEMENT);
263     table_init(TAG_ARG, "arg", pa_string, T_ELEMENT);
264     table_init(TAG_ENV, "env", pa_string, T_ELEMENT);
265     table_init(TAG_XAT, "xattr", pa_string, T_ELEMENT);

267     table_init(TAG_RESULT, "result", NOFUNC, T_ATTRIBUTE);
268     table_init(TAG_CUID, "creator-uid", pa_pw_uid, T_ATTRIBUTE);
269     table_init(TAG_CGID, "creator-gid", pa_gr_uid, T_ATTRIBUTE);
270     table_init(TAG_SEQ, "seq", pa_adr_u_int32, T_ATTRIBUTE);
271     table_init(TAG_KEY, "key", pa_adr_int32hex, T_ATTRIBUTE);

273     table_init(TAG_IPVERS, "version", pa_adr_charhex, T_ATTRIBUTE);
274     table_init(TAG_IPSERV, "service_type", pa_adr_charhex, T_ATTRIBUTE);
275     table_init(TAG_IPLEN, "len", pa_adr_short, T_ATTRIBUTE);
276     table_init(TAG_IPID, "id", pa_adr_u_short, T_ATTRIBUTE);
277     table_init(TAG_IPOFFS, "offset", pa_adr_u_short, T_ATTRIBUTE);
278     table_init(TAG_IPTTL, "time_to_live", pa_adr_charhex, T_ATTRIBUTE);
279     table_init(TAG_IPPROTO, "protocol", pa_adr_charhex, T_ATTRIBUTE);
280     table_init(TAG_IPCKSUM, "cksum", pa_adr_u_short, T_ATTRIBUTE);
281     table_init(TAG_IPSRC, "src_addr", pa_adr_int32hex, T_ATTRIBUTE);
282     table_init(TAG_IPDEST, "dest_addr", pa_adr_int32hex, T_ATTRIBUTE);

284     table_init(TAG_ACLTYPE, "type", NOFUNC, T_ATTRIBUTE);
285     table_init(TAG_ACLVAL, "value", NOFUNC, T_ATTRIBUTE);
286     table_init(TAG_ACEMASK, "access_mask", NOFUNC, T_ATTRIBUTE);
287     table_init(TAG_ACEFLAGS, "flags", NOFUNC, T_ATTRIBUTE);
288     table_init(TAG_ACETYPE, "type", NOFUNC, T_ATTRIBUTE);
289     table_init(TAG_ACEID, "id", NOFUNC, T_ATTRIBUTE);
290     table_init(TAG_SOCKETYPE, "type", pa_adr_shorthex, T_ATTRIBUTE);
291     table_init(TAG_SOCKETPORT, "port", pa_adr_shorthex, T_ATTRIBUTE);
292     table_init(TAG_SOCKETADDR, "addr", NOFUNC, T_ATTRIBUTE);

294     table_init(TAG_SOCKETEXDOM, "sock_domain", pa_adr_shorthex, T_ATTRIBUTE);
295     table_init(TAG_SOCKETEXTYPE, "sock_type", pa_adr_shorthex, T_ATTRIBUTE);
296     table_init(TAG_SOCKETEXLPORT, "lport", NOFUNC, T_ATTRIBUTE);
297     table_init(TAG_SOCKETEXLADDR, "laddr", NOFUNC, T_ATTRIBUTE);
298     table_init(TAG_SOCKETEXFPORT, "fport", NOFUNC, T_ATTRIBUTE);
299     table_init(TAG_SOCKETEXFADDR, "faddr", NOFUNC, T_ATTRIBUTE);

301     table_init(TAG_IPCTYPE, "ipc-type", NOFUNC, T_ATTRIBUTE);
302     table_init(TAG_IPCID, "ipc-id", pa_adr_int32, T_ATTRIBUTE);

304     table_init(TAG_ARBPRINT, "print", NOFUNC, T_ATTRIBUTE);
305     table_init(TAG_ARBTYPE, "type", NOFUNC, T_ATTRIBUTE);
306     table_init(TAG_ARBCOUNT, "count", NOFUNC, T_ATTRIBUTE);

308     table_init(TAG_HOSTID, "host", NOFUNC, T_ATTRIBUTE);
309     table_init(TAG_USERNAME, "username", pa_adr_string, T_ATTRIBUTE);
310 #endif /* _PRAUDIT */
311 }
```

new/usr/src/cmd/praudit/toktable.h

1

```
*****
6547 Thu Jun 30 22:01:11 2016
new/usr/src/cmd/praudit/toktable.h
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
_____unchanged_portion_omitted_____

163 /*
164 * These tokens are the same for all versions of Solaris
165 */

167 /*
168 * Control tokens
169 */

171 extern int    file_token();
172 extern int    trailer_token();
173 extern int    header_token();
174 extern int    header32_ex_token();

176 /*
177 * Data tokens
178 */

180 extern int    arbitrary_data_token();
181 extern int    fmri_token();
182 extern int    s5_IPC_token();
183 extern int    path_token();
184 extern int    path_attr_token();
185 extern int    subject32_token();
186 extern int    process32_token();
187 extern int    return_value32_token();
188 extern int    text_token();
189 extern int    opaque_token();
190 extern int    ip_addr_token();
191 extern int    ip_token();
192 extern int    ipport_token();
193 extern int    argument32_token();
194 extern int    socket_token();
195 extern int    sequence_token();

197 /*
198 * Modifier tokens
199 */

201 extern int    acl_token();
202 extern int    ace_token();
203 extern int    attribute_token();
204 extern int    s5_IPC_perm_token();
205 extern int    group_token();
206 extern int    label_token();
207 extern int    privilege_token();
208 extern int    useofpriv_token();
209 extern int    liaison_token();
210 extern int    newgroup_token();
211 extern int    exec_args_token();
212 extern int    exec_env_token();
213 extern int    attribute32_token();
214 extern int    useofauth_token();
215 extern int    user_token();
216 extern int    zonename_token();
217 extern int    secflags_token();
```

new/usr/src/cmd/praudit/toktable.h

2

```
218 #endif /* ! codereview */

220 /*
221 * X windows tokens
222 */

224 extern int    xatom_token();
225 extern int    xselect_token();
226 extern int    xcolormap_token();
227 extern int    xcursor_token();
228 extern int    xfont_token();
229 extern int    xgc_token();
230 extern int    xpixmap_token();
231 extern int    xproperty_token();
232 extern int    xwindow_token();
233 extern int    xclient_token();

235 /*
236 * Command tokens
237 */

239 extern int    cmd_token();
240 extern int    exit_token();

242 /*
243 * Miscellaneous tokens
244 */

246 extern int    host_token();

248 /*
249 * Solaris64 tokens
250 */

252 extern int    argument64_token();
253 extern int    return_value64_token();
254 extern int    attribute64_token();
255 extern int    header64_token();
256 extern int    subject64_token();
257 extern int    process64_token();
258 extern int    file64_token();

260 /*
261 * Extended network address tokens
262 */

264 extern int    header64_ex_token();
265 extern int    subject32_ex_token();
266 extern int    process32_ex_token();
267 extern int    subject64_ex_token();
268 extern int    process64_ex_token();
269 extern int    ip_addr_ex_token();
270 extern int    socket_ex_token();
271 extern int    tid_token();

273 #ifdef __cplusplus
274 }
275 #endif

277 #endif /* _TOKTABLE_H */
```

new/usr/src/cmd/priocntl/subr.c

1

```
*****
13924 Thu Jun 30 22:01:12 2016
new/usr/src/cmd/priocntl/subr.c
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
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, Version 1.0 only
6  * (the "License").  You may not use this file except in compliance
7  * with the License.
8  *
9  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
10 * or http://www.opensolaris.org/os/licensing.
11 * See the License for the specific language governing permissions
12 * and limitations under the License.
13 *
14 * When distributing Covered Code, include this CDDL HEADER in each
15 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
16 * If applicable, add the following below this CDDL HEADER, with the
17 * fields enclosed by brackets "[]" replaced with your own identifying
18 * information: Portions Copyright [yyyy] [name of copyright owner]
19 *
20 * CDDL HEADER END
21 */
22 /*
23  * Copyright 2004 Sun Microsystems, Inc.  All rights reserved.
24  * Use is subject to license terms.
25 */

27 #pragma ident      "%Z%M% %I%      %E% SMI"

27 /*      Copyright (c) 1984, 1986, 1987, 1988, 1989 AT&T */
28 /*      All Rights Reserved      */

31 #include      <stdio.h>
32 #include      <string.h>
33 #include      <strings.h>
34 #include      <stdlib.h>
35 #include      <unistd.h>
36 #include      <sys/types.h>
37 #include      <limits.h>
38 #include      <dirent.h>
39 #include      <fcntl.h>
40 #include      <sys/time.h>
41 #include      <sys/procset.h>
42 #include      <sys/priocntl.h>
43 #include      <sys/task.h>
44 #include      <proefs.h>
45 #include      <project.h>
46 #include      <errno.h>
47 #include      <zone.h>
48 #include      <libcontract_priv.h>

50 #include "priocntl.h"

52 /*LINTLIBRARY*/

54 /*
55  * Utility functions for priocntl command.
56  */
```

new/usr/src/cmd/priocntl/subr.c

2

```
58 static char      *procdir = "/proc";

60 /*PRINTFLIKE1*/
61 void
62 fatalerr(format, a1, a2, a3, a4, a5)
63 char      *format;
64 int      a1, a2, a3, a4, a5;
65 {
66     (void) fprintf(stderr, format, a1, a2, a3, a4, a5);
67     exit(1);
68 }

71 /*
72  * Structure defining idtypes known to the priocntl command
73  * along with the corresponding names
74  * along with the corresponding names and a liberal guess
75  * of the max number of procs sharing any given ID of that type.
76  * The idtype values themselves are defined in <sys/procset.h>.
77  */
78 static struct idtypes {
79     idtype_t      idtype;
80     char      *idtypnm;
81 } idtypes [] = {
82     unchanged_portion_omitted
83 }
```

new/usr/src/cmd/ptools/Makefile

1

```
*****
2277 Thu Jun 30 22:01:13 2016
new/usr/src/cmd/ptools/Makefile
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
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 #
23 # Copyright 2008 Sun Microsystems, Inc. All rights reserved.
24 # Use is subject to license terms.
25 #

28 include ../Makefile.cmd

30 #
31 # Don't add new ptools here; these are legacy ptools which must be symlinked
32 # into /usr/proc/bin
33 #
34 LEGACY_SUBDIRS = \
35     pcred \
36     pfiles \
37     pflags \
38     pldd \
39     pmap \
40     prun \
41     psig \
42     pstack \
43     pstop \
44     ptime \
45     ptree \
46     pwait \
47     pwdx

49 #
50 # 'new' ptools are not symlinked into /usr/proc/bin
51 #
52 NEW_SUBDIRS = \
53     pargs \
54     plgrp \
55     pmadvise \
56     ppriv \
57     preap \
58     psecflags
```

new/usr/src/cmd/ptools/Makefile

2

```
57     preap

60 SUBDIRS = $(LEGACY_SUBDIRS) $(NEW_SUBDIRS)

62 all      :=      TARGET = all
63 install  :=      TARGET = install
64 clean    :=      TARGET = clean
65 clobber  :=      TARGET = clobber
66 lint     :=      TARGET = lint
67 _msg     :=      TARGET = _msg

70 # pmadvise depends on pmap components
71 PMAP = $(SRC)/cmd/ptools/pmap
72 pmadvise/pmadvise.po := CPPFLAGS += -I$(PMAP)

74 #
75 # Commands with messages support
76 #
77 POFILES = plgrp/plgrp.po pmadvise/pmadvise.po psecflags/psecflags.po
78 POFILES = plgrp/plgrp.po pmadvise/pmadvise.po

80 .KEEP_STATE:

82 .PARALLEL: $(SUBDIRS)

84 all install clean lint: $(SUBDIRS)
85 clobber: $(SUBDIRS) clobber_local
86 clobber_local:
87     $(RM) $(CLOBBERFILES)

89 $(NEW_SUBDIRS): FRC
90     @cd $@; pwd; $(MAKE) PTOOL_TYPE=NEW -f ../Makefile.ptool $(TARGET)

92 $(LEGACY_SUBDIRS): FRC
93     @cd $@; pwd; $(MAKE) PTOOL_TYPE=LEGACY -f ../Makefile.ptool $(TARGET)

95 #
96 # Combine all messages files into a single file and copy it to
97 # MSGDOMAIN directory
98 #
99 _msg: ${POFILES}
100     $(RM) $(POFILE)
101     $(CAT) $(POFILES) > $(POFILE)
102     $(RM) $(MSGDOMAIN)/$(POFILE)
103     $(CP) $(POFILE) $(MSGDOMAIN)

105 FRC:
```



```

*****
3201 Thu Jun 30 22:01:14 2016
new/usr/src/cmd/ptools/Makefile.bld
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
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 #
23 # Copyright 2009 Sun Microsystems, Inc. All rights reserved.
24 # Use is subject to license terms.
25 #
26 #
27 PROG:sh = basename `cd ../ pwd`
28 #
29 OBJS = $(PROG).o
30 #
31 SRCS = ../$(PROG).c
32 #
33 FILEMODE = 0555
34 #
35 # libproc is added individually as pwait doesn't need it.
36 # These are defined this way so lint can use them
37 LDLIBS_pargs = -lproc
38 LDLIBS_pcred = -lproc
39 LDLIBS_pfiles = -lproc -lnsl
40 LDLIBS_pflags = -lproc
41 LDLIBS_pldd = -lproc
42 LDLIBS_plgrp = -lproc -llgrp
43 LDLIBS_pmap = -lproc
44 LDLIBS_pmadvise = -lproc
45 LDLIBS_ppriv = -lproc
46 LDLIBS_preap = -lproc
47 LDLIBS_prun = -lproc
48 LDLIBS_psecflags = -lproc -lproject
49 #endif /* ! codereview */
50 LDLIBS_psig = -lproc
51 LDLIBS_pstack = -lproc -lc_db
52 LDLIBS_pstop = -lproc
53 LDLIBS_ptime = -lproc
54 LDLIBS_ptree = -lproc -lcontract
55 LDLIBS_pwdx = -lproc
56 #
57 LDLIBS += $(LDLIBS_$(PROG))

```

```

59 CERRWARN_plgrp += _gcc=-Wno-parentheses
60 #
61 CERRWARN_ppriv += _gcc=-Wno-parentheses
62 CERRWARN_ppriv += _gcc=-Wno-uninitialized
63 #
64 CERRWARN_ptree += _gcc=-Wno-parentheses
65 #
66 CERRWARN_pstack += _gcc=-Wno-uninitialized
67 CERRWARN_pstack += _gcc=-Wno-clobbered
68 #
69 CERRWARN_pargs += _gcc=-Wno-clobbered
70 CERRWARN_pargs += _gcc=-Wno-type-limits
71 #
72 CERRWARN += $(CERRWARN_$(PROG))
73 #
74 # pargs depends on ../../common/elfcap components
75 # pmap depends on pmap components
76 #
77 ELFCAP = $(SRC)/common/elfcap
78 PMAP = $(SRC)/cmd/ptools/pmap
79 #
80 CPPFLAGS_pargs = -I$(ELFCAP)
81 OBJS_pargs = elfcap.o
82 SRCS_pargs = $(ELFCAP)/elfcap.c
83 #
84 CPPFLAGS_pmap = -I$(PMAP)
85 OBJS_pmap = pmap_common.o
86 SRCS_pmap = $(PMAP)/pmap_common.c
87 #
88 CPPFLAGS_pmadvise = -I$(PMAP)
89 OBJS_pmadvise = pmap_common.o
90 SRCS_pmadvise = $(PMAP)/pmap_common.c
91 #
92 CPPFLAGS += $(CPPFLAGS_$(PROG))
93 OBJS += $(OBJS_$(PROG))
94 SRCS += $(SRCS_$(PROG))
95 #
96 INSTALL_NEW=
97 INSTALL_LEGACY=$(RM) $(ROOTPROCBSYMLINK) ; \
98 $(LN) -s ../../bin/$(PROG) $(ROOTPROCBSYMLINK)
99 #
100 .KEEP_STATE:
101 #
102 elfcap.o: $(ELFCAP)/elfcap.c
103 $(COMPILE.c) -o $@ $(ELFCAP)/elfcap.c
104 #
105 pmap_common.o: $(PMAP)/pmap_common.c
106 $(COMPILE.c) -o $@ $(PMAP)/pmap_common.c
107 #
108 %.o: ../%.c
109 $(COMPILE.c) $<
110 #
111 all: $(PROG)
112 #
113 ROOTBINPROG=$(ROOTBIN)/$(PROG)
114 ROOTPROCBSYMLINK=$(ROOT)/usr/proc/bin/$(PROG)
115 #
116 $(PROG): $$$(OBJS)
117 $(LINK.c) $$(OBJS) -o $@ $(LDLIBS)
118 $(POST_PROCESS)
119 #
120 #
121 # Install the ptool, symlinking it into /usr/proc/bin if PTOOL_TYPE is set
122 # to LEGACY.
123 #
124 install: all $(ROOTISAPROG)

```

```
125     -$(RM) $(ROOTBINPROG)
126     -$(LN) $(ISAEEXEC) $(ROOTBINPROG)
127     -$(INSTALL_$(PTOOL_TYPE))
```

```
129 clean:
130     $(RM) $(OBJS)
```

```
132 lint:
133     $(LINT.c) $(SRCS) $(LDLIBS)
```

```

*****
6751 Thu Jun 30 22:01:14 2016
new/usr/src/cmd/ptools/psecflags/psecflags.c
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
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 /* Copyright 2015, Richard Lowe. */

14 #include <err.h>
15 #include <errno.h>
16 #include <grp.h>
17 #include <libintl.h>
18 #include <procfs.h>
19 #include <project.h>
20 #include <pwd.h>
21 #include <stdio.h>
22 #include <stdlib.h>
23 #include <string.h>
24 #include <sys/secflags.h>
25 #include <sys/types.h>

27 #include <libproc.h>
28 #include <libzonecfg.h>

30 extern const char *__progrname;

32 static void
33 print_flags(const char *set, secflagset_t flags)
34 {
35     char buf[1024];

37     secflags_to_str(flags, buf, sizeof (buf));
38     (void) printf("\t%s:\t%s\n", set, buf);
39 }

41 /*
42  * Structure defining idtypes known to the priocntl command
43  * along with the corresponding names.
44  * The idtype values themselves are defined in <sys/procset.h>.
45  */
46 static struct idtypes {
47     idtype_t   type;
48     char       *name;
49 } idtypes [] = {
50     { P_ALL,      "all"      },
51     { P_CTID,    "contract" },
52     { P_CTID,    "ctid"     },
53     { P_GID,     "gid"      },
54     { P_GID,     "group"    },
55     { P_PGID,   "pgid"     },
56     { P_PID,    "pid"      },
57     { P_PPID,   "ppid"     },
58     { P_PROJID, "project"  },

```

```

59     { P_PROJID,  "projid"   },
60     { P_SID,    "session",  },
61     { P_SID,    "sid"      },
62     { P_SID,    "sid"      },
63     { P_TASKID, "taskid"   },
64     { P_UID,    "uid"      },
65     { P_UID,    "user"     },
66     { P_ZONEID, "zone"     },
67     { P_ZONEID, "zoneid"   },
68     { 0,        NULL       },
69 };

71 static int
72 str2idtype(char *idtypnm, idtype_t *idtypep)
73 {
74     struct idtypes *curp;

76     for (curp = idtypes; curp->name != NULL; curp++) {
77         if (strncasecmp(curp->name, idtypnm,
78             strlen(curp->name)) == 0) {
79             *idtypep = curp->type;
80             return (0);
81         }
82     }
83     return (-1);
84 }

86 static id_t
87 getid(idtype_t type, char *value)
88 {
89     struct passwd *pwd;
90     struct group *grp;
91     id_t ret;
92     char *endp;

94     switch (type) {
95     case P_UID:
96         if ((pwd = getpwnam(value)) != NULL)
97             return (pwd->pw_uid);
98         break;
99     case P_GID:
100        if ((grp = getgrnam(value)) != NULL)
101            return (grp->gr_gid);
102        break;
103     case P_PROJID:
104        if ((ret = getprojidbyname(value)) != (id_t)-1)
105            return (ret);
106        break;
107     case P_ZONEID:
108        if (zone_get_id(value, &ret) == 0)
109            return (ret);
110        break;
111     default:
112        break;
113     }

115     errno = 0;

117     ret = (id_t)strtoul(value, &endp, 10);

119     if ((errno != 0) || (*endp != '\0'))
120         return ((id_t)-1);

122     return (ret);
123 }

```

```

125 int
126 main(int argc, char **argv)
127 {
128     secflagdelta_t act;
129     psecflagwhich_t which = PSF_INHERIT;
130     int ret = 0;
131     int pgrab_flags = PGRAB_RDONLY;
132     int opt;
133     char *idtypename = NULL;
134     idtype_t idtype = P_PID;
135     boolean_t usage = B_FALSE;
136     boolean_t e_flag = B_FALSE;
137     boolean_t l_flag = B_FALSE;
138     boolean_t s_flag = B_FALSE;
139     int errc = 0;

141     while ((opt = getopt(argc, argv, "eFi:ls:")) != -1) {
142         switch (opt) {
143             case 'e':
144                 e_flag = B_TRUE;
145                 break;
146             case 'F':
147                 pgrab_flags |= PGRAB_FORCE;
148                 break;
149             case 'i':
150                 idtypename = optarg;
151                 break;
152             case 's':
153                 s_flag = B_TRUE;
154                 if ((strlen(optarg) >= 2) &&
155                     ((optarg[1] == '='))) {
156                     switch (optarg[0]) {
157                         case 'L':
158                             which = PSF_LOWER;
159                             break;
160                         case 'U':
161                             which = PSF_UPPER;
162                             break;
163                         case 'I':
164                             which = PSF_INHERIT;
165                             break;
166                         case 'E':
167                             errx(1, "the effective flags cannot "
168                                 "be changed", optarg[0]);
169                         default:
170                             errx(1, "unknown security flag "
171                                 "set: '%c'", optarg[0]);
172                     }
173                 }
174                 optarg += 2;
175             }

177             if (secflags_parse(NULL, optarg, &act) == -1)
178                 errx(1, "couldn't parse security flags: %s",
179                     optarg);
180             break;
181             case 'l':
182                 l_flag = B_TRUE;
183                 break;
184             default:
185                 usage = B_TRUE;
186                 break;
187         }
188     }

190     argc -= optind;

```

```

191     argv += optind;

193     if (l_flag && ((idtypename != NULL) || s_flag || (argc != 0)))
194         usage = B_TRUE;
195     if ((idtypename != NULL) && !s_flag)
196         usage = B_TRUE;
197     if (e_flag && !s_flag)
198         usage = B_TRUE;
199     if (!l_flag && argc <= 0)
200         usage = B_TRUE;

202     if (usage) {
203         (void) fprintf(stderr,
204             gettext("usage:\t%s [-F] { pid | core } ... \n"),
205             __progname);
206         (void) fprintf(stderr,
207             gettext("\t%s -s spec [-i idtype] id ... \n"),
208             __progname);
209         (void) fprintf(stderr,
210             gettext("\t%s -s spec -e command [arg]... \n"),
211             __progname);
212         (void) fprintf(stderr, gettext("\t%s -l \n"), __progname);
213         return (2);
214     }

216     if (l_flag) {
217         secflag_t i;
218         const char *name;

220         for (i = 0; (name = secflag_to_str(i)) != NULL; i++)
221             (void) printf("%s\n", name);
222         return (0);
223     } else if (s_flag && e_flag) {
224         /*
225          * Don't use the strerror() message for EPERM, "Not Owner"
226          * which is misleading.
227          */
228         errc = psecflags(P_PID, P_MYID, which, &act);
229         switch (errc) {
230             case 0:
231                 break;
232             case EPERM:
233                 errx(1, gettext("failed setting "
234                     "security-flags: Permission denied"));
235                 break;
236             default:
237                 err(1, gettext("failed setting security-flags"));
238         }

240         (void) execvp(argv[0], &argv[0]);
241         err(1, "%s", argv[0]);
242     } else if (s_flag) {
243         int i;
244         id_t id;

246         if (idtypename != NULL)
247             if (str2idtype(idtypename, &idtype) == -1)
248                 errx(1, gettext("No such id type: '%s'",
249                     idtypename));

251         for (i = 0; i < argc; i++) {
252             if ((id = getid(idtype, argv[i])) == (id_t)-1) {
253                 errx(1, gettext("invalid or non-existent "
254                     "identifier: '%s'", argv[i]));
255             }

```

```

257         /*
258          * Don't use the strerror() message for EPERM, "Not
259          * Owner" which is misleading.
260          */
261         if (psecflags(idtype, id, which, &act) != 0) {
262             switch (errno) {
263                 case EPERM:
264                     errx(1, gettext("failed setting "
265                                 "security-flags: "
266                                 "Permission denied"));
267                     break;
268                 default:
269                     err(1, gettext("failed setting "
270                                 "security-flags"));
271             }
272         }
273     }
274
275     return (0);
276 }
277
278 /* Display the flags for the given pids */
279 while (argc-- > 0) {
280     struct ps_prochandle *Pr;
281     const char *arg;
282     psinfo_t psinfo;
283     prsecflags_t *psf;
284     int gcode;
285
286     if ((Pr = proc_arg_grab(arg = *argv++, PR_ARG_ANY,
287                          pgrab_flags, &gcode)) == NULL) {
288         warnx(gettext("cannot examine %s: %s"),
289              arg, Pgrab_error(gcode));
290         ret = 1;
291         continue;
292     }
293
294     (void) memcpy(&psinfo, Ppsinfo(Pr), sizeof (psinfo_t));
295     proc_unctrl_psinfo(&psinfo);
296
297     if (Pstate(Pr) == PS_DEAD) {
298         (void) printf(gettext("core '%s' of %d:\t%.70s\n"),
299                      arg, (int)psinfo.pr_pid, psinfo.pr_psargs);
300     } else {
301         (void) printf("%d:\t%.70s\n",
302                      (int)psinfo.pr_pid, psinfo.pr_psargs);
303     }
304
305     if (Psecflags(Pr, &psf) != 0)
306         err(1, gettext("cannot read secflags of %s"), arg);
307
308     print_flags("E", psf->pr_effective);
309     print_flags("I", psf->pr_inherit);
310     print_flags("L", psf->pr_lower);
311     print_flags("U", psf->pr_upper);
312
313     Psecflags_free(psf);
314     Prelease(Pr, 0);
315 }
316
317     return (ret);
318 }
319 #endif /* ! codereview */

```

```

*****
53662 Thu Jun 30 22:01:15 2016
new/usr/src/cmd/sgs/dump/common/dump.c
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
_____unchanged_portion_omitted_____

1065 /*
1066  * Print dynamic linking information.  Input is an ELF
1067  * file descriptor, the SCNTAB structure, the number of
1068  * sections, and the filename.
1069  */
1070 static void
1071 dump_dynamic(Elf *elf_file, SCNTAB *p_scns, int num_scns, char *filename)
1072 {
1073 #define pdyn_Fmtptr    "%#llx"

1075     Elf_Data      *dyn_data;
1076     GElf_Dyn      p_dyn;
1077     GElf_Phdr     p_phdr;
1078     GElf_Ehdr     p_ehdr;
1079     int            index = 1;
1080     int            lib_scns = num_scns;
1081     SCNTAB        *l_scns = p_scns;
1082     int            header_num = 0;
1083     const char    *str;

1085     (void) gelf_getehdr(elf_file, &p_ehdr);

1087     if (!p_flag)
1088         (void) printf("\n **** DYNAMIC SECTION INFORMATION ****\n");

1090     for (; num_scns > 0; num_scns--, p_scns++) {
1091         GElf_Word    link;
1092         int          ii;

1095         if (p_scns->p_shdr.sh_type != SHT_DYNAMIC)
1096             continue;

1098         if (!p_flag) {
1099             (void) printf("%s:\n", p_scns->scn_name);
1100             (void) printf("[INDEX]\tTag      Value\n");
1101         }

1103         if ((dyn_data = elf_getdata(p_scns->p_sd, NULL)) == 0) {
1104             (void) fprintf(stderr, "%s: %s: no data in "
1105                 "%s section\n", prog_name, filename,
1106                 p_scns->scn_name);
1107             return;
1108         }

1110         link = p_scns->p_shdr.sh_link;
1111         ii = 0;

1113         (void) gelf_getdyn(dyn_data, ii++, &p_dyn);
1114         while (p_dyn.d_tag != DT_NULL) {
1115             union {
1116                 Conv_inv_buf_t      inv;
1117                 Conv_dyn_flag_buf_t  dyn_flag;
1118                 Conv_dyn_flag1_buf_t  dyn_flag1;
1119                 Conv_dyn_feature1_buf_t  dyn_feature1;

```

```

1120             Conv_dyn_posflag1_buf_t  dyn_posflag1;
1121             } conv_buf;

1123         (void) printf("[%d]\t%-15.15s ", index++,
1124             conv_dyn_tag(p_dyn.d_tag,
1125                 p_ehdr.e_ident[EI_OSABI], p_ehdr.e_machine,
1126                 DUMP_CONVFMT, &conv_buf.inv));

1128     /*
1129     * It would be nice to use a table driven loop
1130     * here, but the address space is too sparse
1131     * and irregular.  A switch is simple and robust.
1132     */
1133     switch (p_dyn.d_tag) {
1134     /*
1135     * Items with an address value
1136     */
1137     case DT_PLTGOT:
1138     case DT_HASH:
1139     case DT_STRTAB:
1140     case DT_RELA:
1141     case DT_SYMTAB:
1142     case DT_INIT:
1143     case DT_FINI:
1144     case DT_REL:
1145     case DT_DEBUG:
1146     case DT_TEXTREL:
1147     case DT_JMPREL:
1148     case DT_INIT_ARRAY:
1149     case DT_FINI_ARRAY:
1150     case DT_INIT_ARRAYSZ:
1151     case DT_FINI_ARRAYSZ:
1152     case DT_PREINIT_ARRAY:
1153     case DT_PREINIT_ARRAYSZ:
1154     case DT_SUNW_RTLDINF:
1155     case DT_SUNW_CAP:
1156     case DT_SUNW_CAPINFO:
1157     case DT_SUNW_CAPCHAIN:
1158     case DT_SUNW_SYMTAB:
1159     case DT_SUNW_SYMSORT:
1160     case DT_SUNW_TLSSORT:
1161     case DT_PLTPAD:
1162     case DT_MOVETAB:
1163     case DT_SYMINFO:
1164     case DT_RELACOUNT:
1165     case DT_RELCOUNT:
1166     case DT_VERSYM:
1167     case DT_VERDEF:
1168     case DT_VERDEFNUM:
1169     case DT_VERNEED:
1170         (void) printf(pdyn_Fmtptr,
1171             EC_ADDR(p_dyn.d_un.d_ptr));
1172         break;

1174     /*
1175     * Items with a string value
1176     */
1177     case DT_NEEDED:
1178     case DT_SONAME:
1179     case DT_RPATH:
1180     case DT_RUNPATH:
1181     case DT_SUNW_AUXILIARY:
1182     case DT_SUNW_FILTER:
1183     case DT_CONFIG:
1184     case DT_DEPAUDIT:
1185     case DT_AUDIT:

```

```

1186     case DT_AUXILIARY:
1187     case DT_USED:
1188     case DT_FILTER:
1189         if (v_flag) { /* Look up the string */
1190             str = (char *)elf_strptr(elf_file, link,
1191                                     p_dyn.d_un.d_ptr);
1192             if (!(str && *str))
1193                 str = (char *)UNKNOWN;
1194             (void) printf("%s", str);
1195         } else { /* Show the address */
1196             (void) printf(pdyn_Fmtptr,
1197                          EC_ADDR(p_dyn.d_un.d_ptr));
1198         }
1199         break;

1201     /*
1202     * Items with a literal value
1203     */
1204     case DT_PLTRELSZ:
1205     case DT_RELASZ:
1206     case DT_RELAENT:
1207     case DT_STRSZ:
1208     case DT_SYMENT:
1209     case DT_RELSZ:
1210     case DT_RELENT:
1211     case DT_PLTREL:
1212     case DT_BIND_NOW:
1213     case DT_CHECKSUM:
1214     case DT_PLTPADSZ:
1215     case DT_MOVEENT:
1216     case DT_MOVESZ:
1217     case DT_SYMINSZ:
1218     case DT_SYMINENT:
1219     case DT_VERNEEDNUM:
1220     case DT_SPARC_REGISTER:
1221     case DT_SUNW_SYMSZ:
1222     case DT_SUNW_SORTENT:
1223     case DT_SUNW_SYMSORTSZ:
1224     case DT_SUNW_TLSSORTSZ:
1225     case DT_SUNW_STRPAD:
1226     case DT_SUNW_CAPCHAINENT:
1227     case DT_SUNW_CAPCHAINSZ:
1228     case DT_SUNW_ASLR:
1229 #endif /* ! codereview */
1230         (void) printf(pdyn_Fmtptr,
1231                     EC_XWORD(p_dyn.d_un.d_val));
1232         break;

1234     /*
1235     * Integer items that are bitmasks, or which
1236     * can be otherwise formatted in symbolic form.
1237     */
1238     case DT_FLAGS:
1239     case DT_FEATURE_1:
1240     case DT_POSFLAG_1:
1241     case DT_FLAGS_1:
1242     case DT_SUNW_LDMACH:
1243         str = NULL;
1244         if (v_flag) {
1245             switch (p_dyn.d_tag) {
1246             case DT_FLAGS:
1247                 str = conv_dyn_flag(
1248                     p_dyn.d_un.d_val,
1249                     DUMP_CONVFMT,
1250                     &conv_buf.dyn_flag);
1251                 break;

```

```

1252     case DT_FEATURE_1:
1253         str = conv_dyn_feature1(
1254             p_dyn.d_un.d_val,
1255             DUMP_CONVFMT,
1256             &conv_buf.dyn_feature1);
1257         break;
1258     case DT_POSFLAG_1:
1259         str = conv_dyn_posflag1(
1260             p_dyn.d_un.d_val,
1261             DUMP_CONVFMT,
1262             &conv_buf.dyn_posflag1);
1263         break;
1264     case DT_FLAGS_1:
1265         str = conv_dyn_flag1(
1266             p_dyn.d_un.d_val, 0,
1267             &conv_buf.dyn_flag1);
1268         break;
1269     case DT_SUNW_LDMACH:
1270         str = conv_ehdr_mach(
1271             p_dyn.d_un.d_val, 0,
1272             &conv_buf.inv);
1273         break;
1274     }
1275     if (str) { /* Show as string */
1276         (void) printf("%s", str);
1277     } else { /* Numeric form */
1278         (void) printf(pdyn_Fmtptr,
1279                     EC_ADDR(p_dyn.d_un.d_ptr));
1280     }
1281     }
1282     break;

1284     /*
1285     * Deprecated items with a literal value
1286     */
1287     case DT_DEPRECATED_SPARC_REGISTER:
1288         (void) printf(pdyn_Fmtptr,
1289                     " (deprecated value)",
1290                     EC_XWORD(p_dyn.d_un.d_val));
1291         break;

1293     /* Ignored items */
1294     case DT_SYMBOLIC:
1295         (void) printf(" (ignored)");
1296         break;
1297     }
1298     (void) printf("\n");
1299     (void) gelf_getdyn(dyn_data, ii++, &p_dyn);
1300 }
1301 }

1303     /*
1304     * Check for existence of static shared library information.
1305     */
1306     while (header_num < p_ehdr.e_phnum) {
1307         (void) gelf_getphdr(elf_file, header_num, &p_phdr);
1308         if (p_phdr.p_type == PT_SHLIB) {
1309             while (--lib_scns > 0) {
1310                 if (strcmp(l_scns->scn_name, ".lib") == 0) {
1311                     print_static(l_scns, filename);
1312                 }
1313                 l_scns++;
1314             }
1315         }
1316         header_num++;
1317     }

```

```

1318 #undef pdyn_Fmtptr
1319 }

1321 /*
1322 * Print the ELF header. Input is an ELF file descriptor
1323 * and the filename. If f_flag is set, the ELF header is
1324 * printed to stdout, otherwise the function returns after
1325 * setting the pointer to the ELF header. Any values which
1326 * are not known are printed in decimal. Fields must be updated
1327 * as new values are added.
1328 */
1329 static GElf_Ehdr *
1330 dump_elf_header(Elf *elf_file, char *filename, GElf_Ehdr * elf_head_p)
1331 {
1332     int class;
1333     int field;

1335     if (gelf_getehdr(elf_file, elf_head_p) == NULL) {
1336         (void) fprintf(stderr, "%s: %s: %s\n", prog_name, filename,
1337             elf_errmsg(-1));
1338         return (NULL);
1339     }

1341     class = (int)elf_head_p->e_ident[4];

1343     if (class == ELFCLASS64)
1344         field = 21;
1345     else
1346         field = 13;

1348     if (!f_flag)
1349         return (elf_head_p);

1351     if (!p_flag) {
1352         (void) printf("\n          **** ELF HEADER ****\n");
1353         (void) printf("%-11s%-11s%-11s%-11s%-11s\n",
1354             field, "Class", "Data", field, "Type");
1355         (void) printf("%-11s%-11s%-11s%-11s%-11s\n",
1356             field, "Entry", "Phoff", field, "Shoff");
1357         (void) printf("%-11s%-11s%-11s%-11s%-11s\n",
1358             field, "Phentsize", "Phnum", field, "Shentsz");
1359     }

1361     if (!v_flag) {
1362         (void) printf("%-11d%-11d%-11d%-11d\n",
1363             field, elf_head_p->e_ident[4], elf_head_p->e_ident[5],
1364             field, (int)elf_head_p->e_type, (int)elf_head_p->e_machine,
1365             elf_head_p->e_version);
1366     } else {
1367         Conv_inv_buf_t inv_buf;

1369         (void) printf("%-11s", field,
1370             conv_ehdr_class(class, DUMP_CONVFMT, &inv_buf));
1371         (void) printf("%-11s",
1372             conv_ehdr_data(elf_head_p->e_ident[5], DUMP_CONVFMT,
1373             &inv_buf));
1374         (void) printf("%-11s", field,
1375             conv_ehdr_type(elf_head_p->e_ident[EI_OSABI],
1376             elf_head_p->e_type, DUMP_CONVFMT, &inv_buf));
1377         (void) printf("%-11s",
1378             conv_ehdr_mach(elf_head_p->e_machine, DUMP_CONVFMT,
1379             &inv_buf));
1380         (void) printf("%-11s",
1381             conv_ehdr_vers(elf_head_p->e_version, DUMP_CONVFMT,
1382             &inv_buf));
1383     }

```

```

1384     (void) printf("%-11x%-11llx%-11lx%-11lx%-11lx\n",
1385         field, EC_ADDR(elf_head_p->e_entry), EC_OFF(elf_head_p->e_phoff),
1386         field, EC_OFF(elf_head_p->e_shoff), EC_WORD(elf_head_p->e_flags),
1387         EC_WORD(elf_head_p->e_ehsize));
1388     if (!v_flag || (elf_head_p->e_shstrndx != SHN_XINDEX)) {
1389         (void) printf("%-11x%-11u%-11x%-11u\n",
1390             field, EC_WORD(elf_head_p->e_phentsize),
1391             EC_WORD(elf_head_p->e_phnum),
1392             field, EC_WORD(elf_head_p->e_shentsize),
1393             EC_WORD(elf_head_p->e_shnum),
1394             EC_WORD(elf_head_p->e_shstrndx));
1395     } else {
1396         (void) printf("%-11x%-11u%-11x%-11uXINDEX\n",
1397             field, EC_WORD(elf_head_p->e_phentsize),
1398             EC_WORD(elf_head_p->e_phnum),
1399             field, EC_WORD(elf_head_p->e_shentsize),
1400             EC_WORD(elf_head_p->e_shnum));
1401     }
1402     if ((elf_head_p->e_shnum == 0) && (elf_head_p->e_shoff > 0)) {
1403         Elf_Scn *scn;
1404         GElf_Shdr shdr0;
1405         int field;

1407         if (gelf_getclass(elf_file) == ELFCLASS64)
1408             field = 21;
1409         else
1410             field = 13;
1411         if (!p_flag) {
1412             (void) printf("\n          **** SECTION HEADER[0] **
1413             \n          {Elf Extensions} ****\n");
1414             (void) printf(
1415                 "[No]\tType\tFlags\t%-11s\t%-11s\t%-11sName\n",
1416                 field, "Addr", field, "Offset", field,
1417                 "Size(shnum)",
1418                 /* compatibility: tab for elf32 */
1419                 ((field == 13) ? "\t" : " "));
1420             (void) printf("\tLn(strndx) Info\t%-11s Entsize\n",
1421                 field, "Adralgn");
1422         }
1423         if ((scn = elf_getscn(elf_file, 0)) == NULL) {
1424             (void) fprintf(stderr,
1425                 "%s: %s: elf_getscn failed: %s\n",
1426                 prog_name, filename, elf_errmsg(-1));
1427             return (NULL);
1428         }
1429         if (gelf_getshdr(scn, &shdr0) == 0) {
1430             (void) fprintf(stderr,
1431                 "%s: %s: gelf_getshdr: %s\n",
1432                 prog_name, filename, elf_errmsg(-1));
1433             return (NULL);
1434         }
1435         (void) printf("[0]\t%u\t%llu\t", EC_WORD(shdr0.sh_type),
1436             EC_XWORD(shdr0.sh_flags));

1438         (void) printf("%-11x %-11lx %-11u%-11u\n",
1439             field, EC_ADDR(shdr0.sh_addr),
1440             field, EC_OFF(shdr0.sh_offset),
1441             field, EC_XWORD(shdr0.sh_size),
1442             /* compatibility: tab for elf32 */
1443             ((field == 13) ? "\t" : " "));
1444         field, EC_WORD(shdr0.sh_name));

1446         (void) printf("\t%u\t%u\t%-11x %-11lx\n",
1447             EC_WORD(shdr0.sh_link),
1448             EC_WORD(shdr0.sh_info),
1449             field, EC_XWORD(shdr0.sh_addralign),

```



```

1450         field, EC_XWORD(shdr0.sh_entsize));
1451     }
1452     (void) printf("\n");
1454     return (elf_head_p);
1455 }

1457 /*
1458  * Print section contents. Input is an ELF file descriptor,
1459  * the ELF header, the SCNTAB structure,
1460  * the number of symbols, and the filename.
1461  * The number of sections,
1462  * and the offset into the SCNTAB structure will be
1463  * set in dump_section if d_flag or n_flag are set.
1464  * If v_flag is set, sections which can be interpreted will
1465  * be interpreted, otherwise raw data will be output in hexadecimal.
1466  */
1467 static void
1468 print_section(Elf *elf_file,
1469              GElf_Ehdr *p_ehdr, SCNTAB *p, int num_scns, char *filename)
1470 {
1471     unsigned char *p_sec;
1472     int i;
1473     size_t size;

1475     for (i = 0; i < num_scns; i++, p++) {
1476         GElf_Shdr shdr;

1478         size = 0;
1479         if (s_flag && !v_flag)
1480             p_sec = (unsigned char *)get_rawscn(p->p_sd, &size);
1481         else
1482             p_sec = (unsigned char *)get_scndata(p->p_sd, &size);

1484         if ((gelf_getshdr(p->p_sd, &shdr) != NULL) &&
1485             (shdr.sh_type == SHT_NOBITS)) {
1486             continue;
1487         }
1488         if (s_flag && !v_flag) {
1489             (void) printf("\n%s:\n", p->scn_name);
1490             print_rawdata(p_sec, size);
1491             continue;
1492         }
1493         if (shdr.sh_type == SHT_SYMTAB) {
1494             dump_symbol_table(elf_file, p, filename);
1495             continue;
1496         }
1497         if (shdr.sh_type == SHT_DYNSYM) {
1498             dump_symbol_table(elf_file, p, filename);
1499             continue;
1500         }
1501         if (shdr.sh_type == SHT_STRTAB) {
1502             dump_string_table(p, 1);
1503             continue;
1504         }
1505         if (shdr.sh_type == SHT_RELA) {
1506             dump_reloc_table(elf_file, p_ehdr, p, 1, filename);
1507             continue;
1508         }
1509         if (shdr.sh_type == SHT_REL) {
1510             dump_reloc_table(elf_file, p_ehdr, p, 1, filename);
1511             continue;
1512         }
1513         if (shdr.sh_type == SHT_DYNAMIC) {
1514             dump_dynamic(elf_file, p, 1, filename);
1515             continue;

```

```

1516     }

1518         (void) printf("\n%s:\n", p->scn_name);
1519         print_rawdata(p_sec, size);
1520     }
1521     (void) printf("\n");
1522 }

1524 /*
1525  * Print section contents. This function does not print the contents
1526  * of the sections but sets up the parameters and then calls
1527  * print_section to print the contents. Calling another function to print
1528  * the contents allows both -d and -n to work correctly
1529  * simultaneously. Input is an ELF file descriptor, the ELF header,
1530  * the SCNTAB structure, the number of sections, and the filename.
1531  * Set the range of sections if d_flag, and set section name if
1532  * n_flag.
1533  */
1534 static void
1535 dump_section(Elf *elf_file,
1536             GElf_Ehdr *p_ehdr, SCNTAB *s, int num_scns, char *filename)
1537 {
1538     SCNTAB *n_range, *d_range; /* for use with -n and -d modifiers */
1539     int i;
1540     int found_it = 0; /* for use with -n section_name */

1542     if (n_flag) {
1543         n_range = s;

1545         for (i = 0; i < num_scns; i++, n_range++) {
1546             if ((strcmp(name, n_range->scn_name)) != 0)
1547                 continue;
1548             else {
1549                 found_it = 1;
1550                 print_section(elf_file, p_ehdr,
1551                             n_range, 1, filename);
1552             }
1553         }

1555         if (!found_it) {
1556             (void) fprintf(stderr, "%s: %s: %s not found\n",
1557                          prog_name, filename, name);
1558         }
1559     } /* end n_flag */

1561     if (d_flag) {
1562         d_range = s;
1563         d_num = check_range(d_low, d_hi, num_scns, filename);
1564         if (d_num < 0)
1565             return;
1566         d_range += d_low - 1;

1568         print_section(elf_file, p_ehdr, d_range, d_num, filename);
1569     } /* end d_flag */

1571     if (!n_flag && !d_flag)
1572         print_section(elf_file, p_ehdr, s, num_scns, filename);
1573 }

1575 /*
1576  * Print the section header table. This function does not print the contents
1577  * of the section headers but sets up the parameters and then calls
1578  * print_shdr to print the contents. Calling another function to print
1579  * the contents allows both -d and -n to work correctly
1580  * simultaneously. Input is the SCNTAB structure,
1581  * the number of sections from the ELF header, and the filename.

```

```

1582 * Set the range of section headers to print if d_flag, and set
1583 * name of section header to print if n_flag.
1584 */
1585 static void
1586 dump_shdr(Elf *elf_file, SCNTAB *s, int num_scns, char *filename)
1587 {
1588
1589     SCNTAB *n_range, *d_range;      /* for use with -n and -d modifiers */
1590     int field;
1591     int i;
1592     int found_it = 0; /* for use with -n section_name */
1593
1594     if (gelf_getclass(elf_file) == ELFCLASS64)
1595         field = 21;
1596     else
1597         field = 13;
1598
1599     if (!p_flag) {
1600         (void) printf("\n          **** SECTION HEADER TABLE ****\n");
1601         (void) printf("[No]\tType\tFlags\t%-*s %-*s %-*s\n",
1602             field, "Addr", field, "Offset", field, "Size",
1603             /* compatibility: tab for elf32 */
1604             (field == 13) ? "\t" : " ");
1605         (void) printf("\tLink\tInfo\t%-*s Entsize\n\n",
1606             field, "Adralgn");
1607     }
1608
1609     if (n_flag) {
1610         n_range = s;
1611
1612         for (i = 1; i <= num_scns; i++, n_range++) {
1613             if ((strcmp(name, n_range->scn_name)) != 0)
1614                 continue;
1615             else {
1616                 found_it = 1;
1617                 print_shdr(elf_file, n_range, 1, i);
1618             }
1619         }
1620
1621         if (!found_it) {
1622             (void) fprintf(stderr, "%s: %s: %s not found\n",
1623                 prog_name, filename, name);
1624         }
1625     } /* end n_flag */
1626
1627     if (d_flag) {
1628         d_range = s;
1629         d_num = check_range(d_low, d_hi, num_scns, filename);
1630         if (d_num < 0)
1631             return;
1632         d_range += d_low - 1;
1633
1634         print_shdr(elf_file, d_range, d_num, d_low);
1635     } /* end d_flag */
1636
1637     if (!n_flag && !d_flag)
1638         print_shdr(elf_file, s, num_scns, 1);
1639 }
1640
1641 /*
1642 * Process all of the command line options (except
1643 * for -a, -g, -f, and -o). All of the options processed
1644 * by this function require the presence of the section
1645 * header table and will not be processed if it is not present.
1646 * Set up a buffer containing section name, section header,
1647 * and section descriptor for each section in the file. This

```

```

1648 * structure is used to avoid duplicate calls to libelf functions.
1649 * Structure members for the symbol table, the debugging information,
1650 * and the line number information are global. All of the
1651 * rest are local.
1652 */
1653 static void
1654 dump_section_table(Elf *elf_file, GElf_Ehdr *elf_head_p, char *filename)
1655 {
1656
1657     static SCNTAB *buffer, *p_scns;
1658     Elf_Scn *scn = 0;
1659     char *s_name = NULL;
1660     int found = 0;
1661     unsigned int num_scns;
1662     size_t shstrndx;
1663     size_t shnum;
1664
1665     if (elf_getshdrnum(elf_file, &shnum) == -1) {
1666         (void) fprintf(stderr,
1667             "%s: %s: elf_getshdrnum failed: %s\n",
1668             prog_name, filename, elf_errmsg(-1));
1669         return;
1670     }
1671     if (elf_getshdrstrndx(elf_file, &shstrndx) == -1) {
1672         (void) fprintf(stderr,
1673             "%s: %s: elf_getshdrstrndx failed: %s\n",
1674             prog_name, filename, elf_errmsg(-1));
1675         return;
1676     }
1677
1678     if ((buffer = calloc(shnum, sizeof (SCNTAB))) == NULL) {
1679         (void) fprintf(stderr, "%s: %s: cannot calloc space\n",
1680             prog_name, filename);
1681         return;
1682     }
1683     /* LINTED */
1684     num_scns = (int)shnum - 1;
1685
1686     p_syntab = (SCNTAB *)0;
1687     p_dynsym = (SCNTAB *)0;
1688     p_scns = buffer;
1689     p_head_scns = buffer;
1690
1691     while ((scn = elf_nextscn(elf_file, scn)) != 0) {
1692         if ((gelf_getshdr(scn, &buffer->p_shdr)) == 0) {
1693             (void) fprintf(stderr,
1694                 "%s: %s: %s\n", prog_name, filename,
1695                 elf_errmsg(-1));
1696             return;
1697         }
1698         s_name = (char *)
1699             elf_strptr(elf_file, shstrndx, buffer->p_shdr.sh_name);
1700         buffer->scn_name = s_name ? s_name : (char *)UNKNOWN;
1701         buffer->p_sd = scn;
1702
1703         if (buffer->p_shdr.sh_type == SHT_SYMTAB) {
1704             found += 1;
1705             p_syntab = buffer;
1706         }
1707         if (buffer->p_shdr.sh_type == SHT_DYNSYM)
1708             p_dynsym = buffer;
1709         buffer++;
1710     }
1711
1712     /*

```



```

1846         "%b %d %H:%M:%S %Y",
1847         localtime(
1848         &(p_ar->ar_date))) == 0) {
1849             (void) fprintf(stderr,
1850             "%s: %s: don't have enough space to store the date\n", prog_name, filename);
1851             exit(1);
1852         }
1853         (void) printf(
1854         "\t%s %6d %6d 0%.6ho 0x%.8lx %-s\n\n",
1855         buf, (int)p_ar->ar_uid,
1856         (int)p_ar->ar_gid,
1857         (int)p_ar->ar_mode,
1858         p_ar->ar_size, p_ar->ar_name);
1859     }
1860 }
1861 }
1862 cmd = elf_next(arf);
1863 (void) elf_end(arf);
1864 } /* end while */

1866 err = elf_errno();
1867 if (err != 0) {
1868     (void) fprintf(stderr,
1869     "%s: %s: %s\n", prog_name, filename, elf_errmsg(err));
1870 }
1871 }

1873 /*
1874 * Process member files of an archive. This function provides
1875 * a loop through an archive equivalent the processing of
1876 * each file for individual object files.
1877 */
1878 static void
1879 dump_ar_files(int fd, Elf *elf_file, char *filename)
1880 {
1881     Elf_Arhdr *p_ar;
1882     Elf *arf;
1883     Elf_Cmd cmd;
1884     Elf_Kind file_type;
1885     GElf_Ehdr elf_head;
1886     char *fullname;

1888     cmd = ELF_C_READ;
1889     while ((arf = elf_begin(fd, cmd, elf_file)) != 0) {
1890         size_t len;

1892         p_ar = elf_getarhdr(arf);
1893         if (p_ar == NULL) {
1894             (void) fprintf(stderr, "%s: %s: %s\n",
1895             prog_name, filename, elf_errmsg(-1));
1896             return;
1897         }
1898         if (p_ar->ar_name[0] == '/') {
1899             cmd = elf_next(arf);
1900             (void) elf_end(arf);
1901             continue;
1902         }

1904         len = strlen(filename) + strlen(p_ar->ar_name) + 3;
1905         if ((fullname = malloc(len)) == NULL)
1906             return;
1907         (void) snprintf(fullname, len, "%s[%s]", filename,
1908         p_ar->ar_name);
1909         (void) printf("\n%s:\n", fullname);
1910         file_type = elf_kind(arf);
1911         if (file_type == ELF_K_ELF) {

```

```

1912         if (dump_elf_header(arf, fullname, &elf_head) == NULL)
1913             return;
1914         if (o_flag)
1915             dump_exec_header(arf,
1916             (unsigned)elf_head.e_phnum, fullname);
1917         if (x_flag)
1918             dump_section_table(arf, &elf_head, fullname);
1919     } else {
1920         (void) fprintf(stderr, "%s: %s: invalid file type\n",
1921         prog_name, fullname);
1922         cmd = elf_next(arf);
1923         (void) elf_end(arf);
1924         continue;
1925     }

1927     cmd = elf_next(arf);
1928     (void) elf_end(arf);
1929 } /* end while */
1930 }

1932 /*
1933 * Takes a filename as input. Test first for a valid version
1934 * of libelf.a and exit on error. Process each valid file
1935 * or archive given as input on the command line. Check
1936 * for file type. If it is an archive, process the archive-
1937 * specific options first, then files within the archive.
1938 * If it is an ELF object file, process it; otherwise
1939 * warn that it is an invalid file type.
1940 * All options except the archive-specific and program
1941 * execution header are processed in the function, dump_section_table.
1942 */
1943 static void
1944 each_file(char *filename)
1945 {
1946     Elf *elf_file;
1947     GElf_Ehdr elf_head;
1948     int fd;
1949     Elf_Kind file_type;

1951     struct stat buf;

1953     Elf_Cmd cmd;
1954     errno = 0;

1956     if (stat(filename, &buf) == -1) {
1957         int err = errno;
1958         (void) fprintf(stderr, "%s: %s: %s", prog_name, filename,
1959         strerror(err));
1960         return;
1961     }

1963     if ((fd = open((filename), O_RDONLY)) == -1) {
1964         (void) fprintf(stderr, "%s: %s: cannot read\n", prog_name,
1965         filename);
1966         return;
1967     }
1968     cmd = ELF_C_READ;
1969     if ((elf_file = elf_begin(fd, cmd, (Elf *)0)) == NULL) {
1970         (void) fprintf(stderr, "%s: %s: %s\n", prog_name, filename,
1971         elf_errmsg(-1));
1972         return;
1973     }

1975     file_type = elf_kind(elf_file);
1976     if (file_type == ELF_K_AR) {
1977         if (a_flag || g_flag) {

```

```

1978     dump_ar_hdr(fd, elf_file, filename);
1979     elf_file = elf_begin(fd, cmd, (Elf *)0);
1980     }
1981     if (z_flag)
1982         dump_ar_files(fd, elf_file, filename);
1983 } else {
1984     if (file_type == ELF_K_ELF) {
1985         (void) printf("\n%s:\n", filename);
1986         if (dump_elf_header(elf_file, filename, &elf_head)) {
1987             if (o_flag)
1988                 dump_exec_header(elf_file,
1989                                 (unsigned)elf_head.e_phnum,
1990                                 filename);
1991             if (x_flag)
1992                 dump_section_table(elf_file,
1993                                   &elf_head, filename);
1994         }
1995     } else {
1996         (void) fprintf(stderr, "%s: %s: invalid file type\n",
1997                       prog_name, filename);
1998     }
1999 }
2000 (void) elf_end(elf_file);
2001 (void) close(fd);
2002 }

2004 /*
2005  * Sets up flags for command line options given and then
2006  * calls each_file() to process each file.
2007  */
2008 int
2009 main(int argc, char *argv[], char *envp[])
2010 {
2011     char *optstr = OPTSTR; /* option string used by getopt() */
2012     int optchar;

2014     /*
2015      * Check for a binary that better fits this architecture.
2016      */
2017     (void) conv_check_native(argv, envp);

2019     prog_name = argv[0];

2021     (void) setlocale(LC_ALL, "");
2022     while ((optchar = getopt(argc, argv, optstr)) != -1) {
2023         switch (optchar) {
2024             case 'a':
2025                 a_flag = 1;
2026                 x_flag = 1;
2027                 break;
2028             case 'g':
2029                 g_flag = 1;
2030                 x_flag = 1;
2031                 break;
2032             case 'v':
2033                 v_flag = 1;
2034                 break;
2035             case 'p':
2036                 p_flag = 1;
2037                 break;
2038             case 'f':
2039                 f_flag = 1;
2040                 z_flag = 1;
2041                 break;
2042             case 'o':
2043                 o_flag = 1;

```

```

2044         z_flag = 1;
2045         break;
2046     case 'h':
2047         h_flag = 1;
2048         x_flag = 1;
2049         z_flag = 1;
2050         break;
2051     case 's':
2052         s_flag = 1;
2053         x_flag = 1;
2054         z_flag = 1;
2055         break;
2056     case 'd':
2057         d_flag = 1;
2058         x_flag = 1;
2059         z_flag = 1;
2060         set_range(optarg, &d_low, &d_hi);
2061         break;
2062     case 'n':
2063         n_flag++;
2064         x_flag = 1;
2065         z_flag = 1;
2066         name = optarg;
2067         break;
2068     case 'r':
2069         r_flag = 1;
2070         x_flag = 1;
2071         z_flag = 1;
2072         break;
2073     case 't':
2074         t_flag = 1;
2075         x_flag = 1;
2076         z_flag = 1;
2077         break;
2078     case 'C':
2079         C_flag = 1;
2080         t_flag = 1;
2081         x_flag = 1;
2082         z_flag = 1;
2083         break;
2084     case 'T':
2085         T_flag = 1;
2086         x_flag = 1;
2087         z_flag = 1;
2088         set_range(optarg, &T_low, &T_hi);
2089         break;
2090     case 'c':
2091         c_flag = 1;
2092         x_flag = 1;
2093         z_flag = 1;
2094         break;
2095     case 'L':
2096         L_flag = 1;
2097         x_flag = 1;
2098         z_flag = 1;
2099         break;
2100     case 'V':
2101         V_flag = 1;
2102         (void) fprintf(stderr, "dump: %s %s\n",
2103                       (const char *)SGU_PKG,
2104                       (const char *)SGU_REL);
2105         break;
2106     case '?':
2107         errflag += 1;
2108         break;
2109     default:

```

```
2110         break;
2111     }
2112 }
2113
2114 if (errflag || (optind >= argc) || (!z_flag && !x_flag)) {
2115     if (!(v_flag && (argc == 2))) {
2116         usage();
2117         exit(269);
2118     }
2119 }
2120
2121 if (elf_version(EV_CURRENT) == EV_NONE) {
2122     (void) fprintf(stderr, "%s: libelf is out of date\n",
2123         prog_name);
2124     exit(101);
2125 }
2126
2127 while (optind < argc) {
2128     each_file(argv[optind]);
2129     optind++;
2130 }
2131 return (0);
2132 }
```

```

*****
57012 Thu Jun 30 22:01:16 2016
new/usr/src/cmd/sgs/elfdump/common/corenote.c
Code review comments from jeffpc
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */

22 /*
23  * Copyright 2008 Sun Microsystems, Inc. All rights reserved.
24  * Use is subject to license terms.
25 */
26 /*
27  * Copyright 2012 DEY Storage Systems, Inc. All rights reserved.
28  * Copyright (c) 2013, Joyent, Inc. All rights reserved.
29 */

31 #include <stdlib.h>
32 #include <stdio.h>
33 #include <string.h>
34 #include <sys/types.h>
35 #include <unistd.h>
36 #include <sys/corect1.h>
37 #include <procfs.h>
38 #endif /* ! codereview */
39 #include <msg.h>
40 #include <elfdump.h>
41 #include <struct_layout.h>
42 #include <conv.h>

45 /*
46  * This module contains the code that displays data from the note
47  * sections found in Solaris core files. The format of these
48  * note sections are described in the core(4) manpage.
49 */

54 /*
55  * Much of the code in this file uses the "%*s" format to set
56  * the left margin indentation. This macro combines the indent
57  * integer argument and the NULL string that follows it.

```

```

58 */
59 #define INDENT state->ns_indent, MSG_ORIG(MSG_STR_EMPTY)

61 /*
62  * Indent unit, used for each nesting
63 */
64 #define INDENT_STEP 4

66 /*
67  * The PRINT macros are convenience wrappers on print_num(),
68  * print_subtype(), and print_strbuf(). They reduce code
69  * clutter by hiding the boilerplate arguments.
70  *
71  * Assumptions:
72  *   - A variable named "layout" exists in the compilation
73  *     environment, referencing the layout information for the
74  *     current type.
75  *   - The variable "state" references the current note state.
76 */
77 #define PRINT_DEC(_title, _field) \
78     print_num(state, _title, &layout->field, SL_FMT_NUM_DEC)
79 #define PRINT_DEC_2UP(_title1, _field1, _title2, _field2) \
80     print_num_2up(state, _title1, &layout->field1, SL_FMT_NUM_DEC, \
81                 _title2, &layout->field2, SL_FMT_NUM_DEC)
82 #define PRINT_HEX(_title, _field) \
83     print_num(state, _title, &layout->field, SL_FMT_NUM_HEX)
84 #define PRINT_HEX_2UP(_title1, _field1, _title2, _field2) \
85     print_num_2up(state, _title1, &layout->field1, SL_FMT_NUM_HEX, \
86                 _title2, &layout->field2, SL_FMT_NUM_HEX)
87 #define PRINT_ZHEX(_title, _field) \
88     print_num(state, _title, &layout->field, SL_FMT_NUM_ZHEX)
89 #define PRINT_ZHEX_2UP(_title1, _field1, _title2, _field2) \
90     print_num_2up(state, _title1, &layout->field1, SL_FMT_NUM_ZHEX, \
91                 _title2, &layout->field2, SL_FMT_NUM_ZHEX)
92 #define PRINT_SUBTYPE(_title, _field, _func) \
93     print_subtype(state, _title, &layout->field, _func)
94 #define PRINT_STRBUF(_title, _field) \
95     print_strbuf(state, _title, &layout->field)

99 /*
100  * Structure used to maintain state data for a core note, or a subregion
101  * (sub-struct) of a core note. These values would otherwise need to be
102  * passed to nearly every routine.
103 */
104 typedef struct {
105     Half          ns_mach;          /* ELF machine type of core file */
106     const sl_arch_layout_t *ns_arch; /* structure layout def for mach */
107     int          ns_swap;          /* True if byte swapping is needed */
108     int          ns_indent;        /* Left margin indentation */
109     int          ns_vcol;          /* Column where value starts */
110     int          ns_t2col;         /* Column where 2up title starts */
111     int          ns_v2col;         /* Column where 2up value starts */
112     const char   *ns_data;         /* Pointer to struct data area */
113     Word         ns_len;           /* Length of struct data area */
114 } note_state_t;

116 /*
117  * Standard signature for a dump function used to process a note
118  * or a sub-structure within a note.
119 */
120 typedef void (* dump_func_t)(note_state_t *state, const char *title);

```

```

127 /*
128 * Some core notes contain string buffers of fixed size
129 * that are expected to contain NULL terminated strings.
130 * If the NULL is there, we can print these strings directly.
131 * However, the potential exists for a corrupt file to have
132 * a non-terminated buffer. This routine examines the given
133 * string, and if the string is terminated, the string itself
134 * is returned. Otherwise, it is copied to a static buffer,
135 * and a pointer to the buffer is returned.
136 */
137 static const char *
138 safe_str(const char *str, size_t n)
139 {
140     static char    buf[512];
141     char          *s;
142     size_t        i;
143
144     if (n == 0)
145         return (MSG_ORIG(MSG_STR_EMPTY));
146
147     for (i = 0; i < n; i++)
148         if (str[i] == '\0')
149             return (str);
150
151     i = (n >= sizeof (buf)) ? (sizeof (buf) - 4) : (n - 1);
152     (void) memcpy(buf, str, i);
153     s = buf + i;
154     if (n >= sizeof (buf)) {
155         *s++ = '.';
156         *s++ = '.';
157         *s++ = '.';
158     }
159     *s = '\0';
160     return (buf);
161 }
162
163 /*
164 * Convenience wrappers on top of the corresponding sl_XXX() functions.
165 */
166 static Word
167 extract_as_word(note_state_t *state, const sl_field_t *fdesc)
168 {
169     return (sl_extract_as_word(state->ns_data, state->ns_swap, fdesc));
170 }
171 static Lword
172 extract_as_lword(note_state_t *state, const sl_field_t *fdesc)
173 {
174     return (sl_extract_as_lword(state->ns_data, state->ns_swap, fdesc));
175 }
176
177 unchanged_portion_omitted

```

```

429 /*
430 * Output information from auxv_t structure.
431 */
432 static void
433 dump_auxv(note_state_t *state, const char *title)
434 {
435     const sl_auxv_layout_t *layout = state->ns_arch->auxv;
436     union {
437         Conv_cap_val_hw1_buf_t    hw1;
438         Conv_cap_val_hw2_buf_t    hw2;

```

```

439         Conv_cnote_auxv_af_buf_t    auxv_af;
440         Conv_ehdr_flags_buf_t        ehdr_flags;
441         Conv_secflags_buf_t          secflags;
442 #endif /* ! codereview */
443         Conv_inv_buf_t                inv;
444     } conv_buf;
445     sl_fmtbuf_t    buf;
446     int            ndx, ndx_start;
447     Word           sizeof_auxv;
448
449     sizeof_auxv = layout->sizeof_struct.sl_f_eltlen;
450
451     indent_enter(state, title, &layout->sizeof_struct);
452
453     /*
454     * Immediate indent_exit() restores the indent level to
455     * that of the title. We include indentation as part of
456     * the index string, which is right justified, and don't
457     * want the usual indentation spacing.
458     */
459     indent_exit(state);
460
461     ndx = 0;
462     while (state->ns_len > sizeof_auxv) {
463         char            index[(MAXNDXSIZE * 2) + 1];
464         sl_fmt_num_t    num_fmt = SL_FMT_NUM_ZHEX;
465         const char      *vstr = NULL;
466         Word            w;
467         int             type;
468         sl_field_t      a_type_next;
469
470         type = extract_as_word(state, &layout->a_type);
471         ndx_start = ndx;
472         switch (type) {
473             case AT_NULL:
474                 a_type_next = layout->a_type;
475                 a_type_next.sl_f_offset += sizeof_auxv;
476                 while ((state->ns_len - sizeof_auxv) >= sizeof_auxv) {
477                     type = extract_as_word(state, &a_type_next);
478                     if (type != AT_NULL)
479                         break;
480                     ndx++;
481                     state->ns_data += sizeof_auxv;
482                     state->ns_len -= sizeof_auxv;
483                 }
484                 num_fmt = SL_FMT_NUM_HEX;
485                 break;
486
487             case AT_IGNORE:
488             case AT_SUN_IFLUSH:
489                 num_fmt = SL_FMT_NUM_HEX;
490                 break;
491
492             case AT_EXECFD:
493             case AT_PHENT:
494             case AT_PHNUM:
495             case AT_PAGESZ:
496             case AT_SUN_UID:
497             case AT_SUN_RUID:
498             case AT_SUN_GID:
499             case AT_SUN_LGID:
500             case AT_SUN_LPAGESZ:
501                 num_fmt = SL_FMT_NUM_DEC;
502                 break;

```



```

506     case AT_FLAGS: /* processor flags */
507         w = extract_as_word(state, &layout->a_val);
508         vstr = conv_ehdr_flags(state->ns_mach, w,
509                               0, &conv_buf.ehdr_flags);
510         break;

512     case AT_SUN_HWCAP:
513         w = extract_as_word(state, &layout->a_val);
514         vstr = conv_cap_val_hwl(w, state->ns_mach,
515                                 0, &conv_buf.hwl);
516         /*
517          * conv_cap_val_hwl() produces output like:
518          *
519          *      0xffff [ flg1 flg2 0xff]
520          *
521          * where the first hex value is the complete value,
522          * and the second is the leftover bits. We only
523          * want the part in brackets, and failing that,
524          * would rather fall back to formatting the full
525          * value ourselves.
526          */
527         while ((*vstr != '\0') && (*vstr != '['))
528             vstr++;
529         if (*vstr != '[')
530             vstr = NULL;
531         num_fmt = SL_FMT_NUM_HEX;
532         break;

533     case AT_SUN_HWCAP2:
534         w = extract_as_word(state, &layout->a_val);
535         vstr = conv_cap_val_hw2(w, state->ns_mach,
536                                 0, &conv_buf.hw2);
537         /*
538          * conv_cap_val_hw2() produces output like:
539          *
540          *      0xffff [ flg1 flg2 0xff]
541          *
542          * where the first hex value is the complete value,
543          * and the second is the leftover bits. We only
544          * want the part in brackets, and failing that,
545          * would rather fall back to formatting the full
546          * value ourselves.
547          */
548         while ((*vstr != '\0') && (*vstr != '['))
549             vstr++;
550         if (*vstr != '[')
551             vstr = NULL;
552         num_fmt = SL_FMT_NUM_HEX;
553         break;

557     case AT_SUN_AUXFLAGS:
558         w = extract_as_word(state, &layout->a_val);
559         vstr = conv_cnote_auxv_af(w, 0, &conv_buf.auxv_af);
560         num_fmt = SL_FMT_NUM_HEX;
561         break;
562     }

564     if (ndx == ndx_start)
565         (void) snprintf(index, sizeof (index),
566                        MSG_ORIG(MSG_FMT_INDEX2), EC_WORD(ndx));
567     else
568         (void) snprintf(index, sizeof (index),
569                        MSG_ORIG(MSG_FMT_INDEXRNG),
570                        EC_WORD(ndx_start), EC_WORD(ndx));

```

```

572         if (vstr == NULL)
573             vstr = fmt_num(state, &layout->a_val, num_fmt, buf);
574         dbg_print(0, MSG_ORIG(MSG_CNOTE_FMT_AUXVLINE), INDENT, index,
575                 state->ns_vcol - state->ns_indent,
576                 conv_cnote_auxv_type(type, CONV_FMT_DECIMAL,
577                 &conv_buf.inv), vstr);

579         state->ns_data += sizeof_auxv;
580         state->ns_len -= sizeof_auxv;
581         ndx++;
582     }
583 }

586 /*
587  * Output information from fltset_t structure.
588  */
589 static void
590 dump_fltset(note_state_t *state, const char *title)
591 {
592     #define NELTS 4

594     const sl_fltset_layout_t      *layout = state->ns_arch->fltset;
595     Conv_cnote_fltset_buf_t buf;
596     sl_field_t                    fdesc;
597     uint32_t                      mask[NELTS];
598     int                            i, nelts;

600     if (!data_present(state, &layout->sizeof_struct))
601         return;

603     fdesc = layout->word;
604     nelts = fdesc.slf_nelts;
605     if (nelts > NELTS) /* Type has grown? Show what we understand */
606         nelts = NELTS;
607     for (i = 0; i < nelts; i++) {
608         mask[i] = extract_as_word(state, &fdesc);
609         fdesc.slf_offset += fdesc.slf_eltlen;
610     }

612     print_str(state, title, conv_cnote_fltset(mask, nelts, 0, &buf));

614 #undef NELTS
615 }

618 /*
619  * Output information from sigset_t structure.
620  */
621 static void
622 dump_sigset(note_state_t *state, const char *title)
623 {
624     #define NELTS 4

626     const sl_sigset_layout_t      *layout = state->ns_arch->sigset;
627     Conv_cnote_sigset_buf_t buf;
628     sl_field_t                    fdesc;
629     uint32_t                      mask[NELTS];
630     int                            i, nelts;

632     if (!data_present(state, &layout->sizeof_struct))
633         return;

635     fdesc = layout->sigbits;
636     nelts = fdesc.slf_nelts;

```

```

637     if (nelts > NELTS)      /* Type has grown? Show what we understand */
638         nelts = NELTS;
639     for (i = 0; i < nelts; i++) {
640         mask[i] = extract_as_word(state, &fdesc);
641         fdesc.slf_offset += fdesc.slf_eltlen;
642     }
644     print_str(state, title, conv_cnote_sigset(mask, nelts, 0, &buf));
646 #undef NELTS
647 }
650 /*
651  * Output information from sigaction structure.
652  */
653 static void
654 dump_sigaction(note_state_t *state, const char *title)
655 {
656     const sl_sigaction_layout_t    *layout = state->ns_arch->sigaction;
657     Conv_cnote_sa_flags_buf_t      conv_buf;
658     Word        w;
660     indent_enter(state, title, &layout->sa_flags);
662     if (data_present(state, &layout->sa_flags)) {
663         w = extract_as_word(state, &layout->sa_flags);
664         print_str(state, MSG_ORIG(MSG_CNOTE_T_SA_FLAGS),
665                 conv_cnote_sa_flags(w, 0, &conv_buf));
666     }
668     PRINT_ZHEX_2UP(MSG_ORIG(MSG_CNOTE_T_SA_HANDLER), sa_hand,
669                 MSG_ORIG(MSG_CNOTE_T_SA_SIGACTION), sa_sigact);
670     PRINT_SUBTYPE(MSG_ORIG(MSG_CNOTE_T_SA_MASK), sa_mask, dump_sigset);
672     indent_exit(state);
673 }
676 /*
677  * Output information from siginfo structure.
678  */
679 static void
680 dump_siginfo(note_state_t *state, const char *title)
681 {
682     const sl_siginfo_layout_t      *layout = state->ns_arch->siginfo;
683     Conv_inv_buf_t      inv_buf;
684     Word        w;
685     int         v_si_code, v_si_signo;
687     if (!data_present(state, &layout->sizeof_struct))
688         return;
690     indent_enter(state, title, &layout->f_si_signo);
692     v_si_signo = extract_as_sword(state, &layout->f_si_signo);
693     print_str(state, MSG_ORIG(MSG_CNOTE_T_SI_SIGNO),
694             conv_cnote_signal(v_si_signo, CONV_FMT_DECIMAL, &inv_buf));
696     w = extract_as_word(state, &layout->f_si_errno);
697     print_str(state, MSG_ORIG(MSG_CNOTE_T_SI_ERRNO),
698             conv_cnote_errno(w, CONV_FMT_DECIMAL, &inv_buf));
700     v_si_code = extract_as_sword(state, &layout->f_si_code);
701     print_str(state, MSG_ORIG(MSG_CNOTE_T_SI_CODE),
702             conv_cnote_si_code(state->ns_mach, v_si_signo, v_si_code,

```

```

703         CONV_FMT_DECIMAL, &inv_buf));
705     if ((v_si_signo == 0) || (v_si_code == SI_NOINFO)) {
706         indent_exit(state);
707         return;
708     }
710     /* User generated signals have (si_code <= 0) */
711     if (v_si_code <= 0) {
712         PRINT_DEC(MSG_ORIG(MSG_CNOTE_T_SI_PID), f_si_pid);
713         PRINT_DEC(MSG_ORIG(MSG_CNOTE_T_SI_UID), f_si_uid);
714         PRINT_DEC(MSG_ORIG(MSG_CNOTE_T_SI_CTIME), f_si_ctid);
715         PRINT_DEC(MSG_ORIG(MSG_CNOTE_T_SI_ZONEID), f_si_zoneid);
716         switch (v_si_code) {
717             case SI_QUEUE:
718             case SI_TIMER:
719             case SI_ASYNCIO:
720             case SI_MESGQ:
721                 indent_enter(state, MSG_ORIG(MSG_CNOTE_T_SI_VALUE),
722                         &layout->f_si_value_int);
723                 PRINT_ZHEX(MSG_ORIG(MSG_CNOTE_T_SIVAL_INT),
724                         f_si_value_int);
725                 PRINT_ZHEX(MSG_ORIG(MSG_CNOTE_T_SIVAL_PTR),
726                         f_si_value_ptr);
727                 indent_exit(state);
728                 break;
729             }
730         indent_exit(state);
731         return;
732     }
734     /*
735     * Remaining cases are kernel generated signals. Output any
736     * signal or code specific information.
737     */
738     if (v_si_code == SI_RCTL)
739         PRINT_HEX(MSG_ORIG(MSG_CNOTE_T_SI_ENTITY), f_si_entity);
740     switch (v_si_signo) {
741         case SIGILL:
742         case SIGFPE:
743         case SIGSEGV:
744         case SIGBUS:
745             PRINT_ZHEX(MSG_ORIG(MSG_CNOTE_T_SI_ADDR), f_si_addr);
746             break;
747         case SIGCHLD:
748             PRINT_DEC(MSG_ORIG(MSG_CNOTE_T_SI_PID), f_si_pid);
749             PRINT_DEC(MSG_ORIG(MSG_CNOTE_T_SI_STATUS), f_si_status);
750             break;
751         case SIGPOLL:
752             PRINT_DEC(MSG_ORIG(MSG_CNOTE_T_SI_BAND), f_si_band);
753             break;
754     }
756     indent_exit(state);
757 }
760 /*
761  * Output information from stack_t structure.
762  */
763 static void
764 dump_stack(note_state_t *state, const char *title)
765 {
766     const sl_stack_layout_t        *layout = state->ns_arch->stack;
767     Conv_cnote_ss_flags_buf_t      conv_buf;
768     Word        w;

```

```

770     indent_enter(state, title, &layout->ss_size);

772     print_num_2up(state, MSG_ORIG(MSG_CNOTE_T_SS_SP), &layout->ss_sp,
773                 SL_FMT_NUM_ZHEX, MSG_ORIG(MSG_CNOTE_T_SS_SIZE), &layout->ss_size,
774                 SL_FMT_NUM_HEX);

776     if (data_present(state, &layout->ss_flags)) {
777         w = extract_as_word(state, &layout->ss_flags);
778         print_str(state, MSG_ORIG(MSG_CNOTE_T_SS_FLAGS),
779                 conv_cnote_ss_flags(w, 0, &conv_buf));
780     }

782     indent_exit(state);
783 }

786 /*
787  * Output information from sysset_t structure.
788  */
789 static void
790 dump_sysset(note_state_t *state, const char *title)
791 {
792 #define NELTS 16

794     const sl_sysset_layout_t      *layout = state->ns_arch->sysset;
795     Conv_cnote_sysset_buf_t buf;
796     sl_field_t                    fdesc;
797     uint32_t                      mask[NELTS];
798     int                            i, nelts;

800     if (!data_present(state, &layout->sizeof_struct))
801         return;

803     fdesc = layout->word;
804     nelts = fdesc.slf_nelts;
805     if (nelts > NELTS) /* Type has grown? Show what we understand */
806         nelts = NELTS;
807     for (i = 0; i < nelts; i++) {
808         mask[i] = extract_as_word(state, &fdesc);
809         fdesc.slf_offset += fdesc.slf_eltlen;
810     }

812     print_str(state, title, conv_cnote_sysset(mask, nelts, 0, &buf));

814 #undef NELTS
815 }

818 /*
819  * Output information from timestruc_t structure.
820  */
821 static void
822 dump_timestruc(note_state_t *state, const char *title)
823 {
824     const sl_timestruc_layout_t *layout = state->ns_arch->timestruc;

826     indent_enter(state, title, &layout->tv_sec);

828     PRINT_DEC_2UP(MSG_ORIG(MSG_CNOTE_T_TV_SEC), tv_sec,
829                 MSG_ORIG(MSG_CNOTE_T_TV_NSEC), tv_nsec);

831     indent_exit(state);
832 }

834 /*

```

```

835  * Output information from prsecflags_t structure.
836  */
837 static void
838 dump_secflags(note_state_t *state, const char *title)
839 {
840     const sl_prsecflags_layout_t *layout = state->ns_arch->prsecflags;
841     Conv_secflags_buf_t inv;
842     Lword lw;
843     Word w;

845     indent_enter(state, title, &layout->pr_version);

847     w = extract_as_word(state, &layout->pr_version);

849     if (w != PRSECFLAGS_VERSION_1) {
850         PRINT_DEC(MSG_INTL(MSG_NOTE_BAD_SECFLAGS_VER), pr_version);
851         dump_hex_bytes(state->ns_data, state->ns_len, state->ns_indent,
852                       4, 3);
853     } else {
854         PRINT_DEC(MSG_ORIG(MSG_CNOTE_T_PR_VERSION), pr_version);
855         lw = extract_as_lword(state, &layout->pr_effective);
856         print_str(state, MSG_ORIG(MSG_CNOTE_T_PR_EFFECTIVE),
857                 conv_prsecflags(lw, 0, &inv));

859         lw = extract_as_lword(state, &layout->pr_inherit);
860         print_str(state, MSG_ORIG(MSG_CNOTE_T_PR_INHERIT),
861                 conv_prsecflags(lw, 0, &inv));

863         lw = extract_as_lword(state, &layout->pr_lower);
864         print_str(state, MSG_ORIG(MSG_CNOTE_T_PR_LOWER),
865                 conv_prsecflags(lw, 0, &inv));

867         lw = extract_as_lword(state, &layout->pr_upper);
868         print_str(state, MSG_ORIG(MSG_CNOTE_T_PR_UPPER),
869                 conv_prsecflags(lw, 0, &inv));
870     }

872     indent_exit(state);
873 }
874 #endif /* ! codereview */

876 /*
877  * Output information from utsname structure.
878  */
879 static void
880 dump_utsname(note_state_t *state, const char *title)
881 {
882     const sl_utsname_layout_t      *layout = state->ns_arch->utsname;

884     indent_enter(state, title, &layout->sysname);

886     PRINT_STRBUF(MSG_ORIG(MSG_CNOTE_T_UTS_SYSNAME), sysname);
887     PRINT_STRBUF(MSG_ORIG(MSG_CNOTE_T_UTS_NODENAME), nodename);
888     PRINT_STRBUF(MSG_ORIG(MSG_CNOTE_T_UTS_RELEASE), release);
889     PRINT_STRBUF(MSG_ORIG(MSG_CNOTE_T_UTS_VERSION), version);
890     PRINT_STRBUF(MSG_ORIG(MSG_CNOTE_T_UTS_MACHINE), machine);

892     indent_exit(state);
893 }

896 /*
897  * Dump register contents
898  */
899 static void
900 dump_prgregset(note_state_t *state, const char *title)

```

```

901 {
902     sl_field_t    fdesc1, fdesc2;
903     sl_fmtbuf_t  buf1, buf2;
904     Conv_inv_buf_t inv_buf1, inv_buf2;
905     Word         w;
906
907     fdesc1 = fdesc2 = state->ns_arch->prgregset->elt0;
908     indent_enter(state, title, &fdesc1);
909
910     for (w = 0; w < fdesc1.sl_felts; ) {
911         if (w == (fdesc1.sl_felts - 1)) {
912             /* One last register is left */
913             if (!data_present(state, &fdesc1))
914                 break;
915             dbg_print(0, MSG_ORIG(MSG_CNOTE_FMT_LINE),
916                     INDENT, state->ns_vcol - state->ns_indent,
917                     conv_cnote_pr_regname(state->ns_mach, w,
918                     CONV_FMT_DECIMAL, &inv_buf1),
919                     fmt_num(state, &fdesc1, SL_FMT_NUM_ZHEX, buf1));
920             fdesc1.sl_foffset += fdesc1.sl_feltlen;
921             w++;
922             continue;
923         }
924
925         /* There are at least 2 more registers left. Show 2 up */
926         fdesc2.sl_foffset = fdesc1.sl_foffset + fdesc1.sl_feltlen;
927         if (!(data_present(state, &fdesc1) &&
928             data_present(state, &fdesc2)))
929             break;
930         dbg_print(0, MSG_ORIG(MSG_CNOTE_FMT_LINE_2UP), INDENT,
931                 state->ns_vcol - state->ns_indent,
932                 conv_cnote_pr_regname(state->ns_mach, w,
933                 CONV_FMT_DECIMAL, &inv_buf1),
934                 state->ns_t2col - state->ns_vcol,
935                 fmt_num(state, &fdesc1, SL_FMT_NUM_ZHEX, buf1),
936                 state->ns_v2col - state->ns_t2col,
937                 conv_cnote_pr_regname(state->ns_mach, w + 1,
938                 CONV_FMT_DECIMAL, &inv_buf2),
939                 fmt_num(state, &fdesc2, SL_FMT_NUM_ZHEX, buf2));
940         fdesc1.sl_foffset += 2 * fdesc1.sl_feltlen;
941         w += 2;
942     }
943
944     indent_exit(state);
945 }
946
947 /*
948  * Output information from lwpstatus_t structure.
949  */
950 static void
951 dump_lwpstatus(note_state_t *state, const char *title)
952 {
953     const sl_lwpstatus_layout_t *layout = state->ns_arch->lwpstatus;
954     Word w, w2;
955     int32_t i;
956     union {
957         Conv_inv_buf_t inv;
958         Conv_cnote_pr_flags_buf_t flags;
959     } conv_buf;
960
961     indent_enter(state, title, &layout->pr_flags);
962
963     if (data_present(state, &layout->pr_flags)) {
964         w = extract_as_word(state, &layout->pr_flags);
965         print_str(state, MSG_ORIG(MSG_CNOTE_T_PR_FLAGS),
966                 conv_cnote_pr_flags(w, 0, &conv_buf.flags));

```

```

967     }
968
969     PRINT_DEC(MSG_ORIG(MSG_CNOTE_T_PR_LWPPID), pr_lwppid);
970
971     if (data_present(state, &layout->pr_why)) {
972         w = extract_as_word(state, &layout->pr_why);
973         print_str(state, MSG_ORIG(MSG_CNOTE_T_PR_WHY),
974                 conv_cnote_pr_why(w, 0, &conv_buf.inv));
975
976         if (data_present(state, &layout->pr_what)) {
977             w2 = extract_as_word(state, &layout->pr_what);
978             print_str(state, MSG_ORIG(MSG_CNOTE_T_PR_WHAT),
979                     conv_cnote_pr_what(w, w2, 0, &conv_buf.inv));
980         }
981     }
982
983     if (data_present(state, &layout->pr_cursig)) {
984         w = extract_as_word(state, &layout->pr_cursig);
985         print_str(state, MSG_ORIG(MSG_CNOTE_T_PR_CURSIG),
986                 conv_cnote_signal(w, CONV_FMT_DECIMAL, &conv_buf.inv));
987     }
988
989     PRINT_SUBTYPE(MSG_ORIG(MSG_CNOTE_T_PR_INFO), pr_info, dump_siginfo);
990     PRINT_SUBTYPE(MSG_ORIG(MSG_CNOTE_T_PR_LWPPEND), pr_lwppend,
991                 dump_sigset);
992     PRINT_SUBTYPE(MSG_ORIG(MSG_CNOTE_T_PR_LWPHOLD), pr_lwphold,
993                 dump_sigset);
994     PRINT_SUBTYPE(MSG_ORIG(MSG_CNOTE_T_PR_ACTION), pr_action,
995                 dump_sigaction);
996     PRINT_SUBTYPE(MSG_ORIG(MSG_CNOTE_T_PR_ALTSTACK), pr_altstack,
997                 dump_stack);
998
999     PRINT_ZHEX(MSG_ORIG(MSG_CNOTE_T_PR_OLDCONTEXT), pr_oldcontext);
1000
1001     if (data_present(state, &layout->pr_syscall)) {
1002         w = extract_as_word(state, &layout->pr_syscall);
1003         print_str(state, MSG_ORIG(MSG_CNOTE_T_PR_SYSCALL),
1004                 conv_cnote_syscall(w, CONV_FMT_DECIMAL, &conv_buf.inv));
1005     }
1006
1007     PRINT_DEC(MSG_ORIG(MSG_CNOTE_T_PR_NSYSARG), pr_nsysarg);
1008
1009     if (data_present(state, &layout->pr_errno)) {
1010         w = extract_as_word(state, &layout->pr_errno);
1011         print_str(state, MSG_ORIG(MSG_CNOTE_T_PR_ERRNO),
1012                 conv_cnote_errno(w, CONV_FMT_DECIMAL, &conv_buf.inv));
1013     }
1014
1015     if (data_present(state, &layout->pr_nsysarg)) {
1016         w2 = extract_as_word(state, &layout->pr_nsysarg);
1017         print_array(state, &layout->pr_sysarg, SL_FMT_NUM_ZHEX, w2, 1,
1018                 MSG_ORIG(MSG_CNOTE_T_PR_SYSARG));
1019     }
1020
1021     PRINT_HEX_2UP(MSG_ORIG(MSG_CNOTE_T_PR_RVAL1), pr_rval1,
1022                 MSG_ORIG(MSG_CNOTE_T_PR_RVAL2), pr_rval2);
1023     PRINT_STRBUF(MSG_ORIG(MSG_CNOTE_T_PR_CLNAME), pr_clname);
1024     PRINT_SUBTYPE(MSG_ORIG(MSG_CNOTE_T_PR_TSTAMP), pr_tstamp,
1025                 dump_timestruc);
1026     PRINT_SUBTYPE(MSG_ORIG(MSG_CNOTE_T_PR_UTIME), pr_utime, dump_timestruc);
1027     PRINT_SUBTYPE(MSG_ORIG(MSG_CNOTE_T_PR_STIME), pr_stime, dump_timestruc);
1028
1029     if (data_present(state, &layout->pr_errpriv)) {
1030         i = extract_as_sword(state, &layout->pr_errpriv);
1031         print_str(state, MSG_ORIG(MSG_CNOTE_T_PR_ERRPRIV),
1032                 conv_cnote_priv(i, CONV_FMT_DECIMAL, &conv_buf.inv));

```

```

1033     }
1035     PRINT_ZHEX_2UP(MSG_ORIG(MSG_CNOTE_T_PR_USTACK), pr_ustack,
1036                 MSG_ORIG(MSG_CNOTE_T_PR_INSTR), pr_instr);
1038     /*
1039     * In order to line up all the values in a single column,
1040     * we would have to set vcol to a very high value, which results
1041     * in ugly looking output that runs off column 80. So, we use
1042     * two levels of vcol, one for the contents so far, and a
1043     * higher one for the pr_reg sub-struct.
1044     */
1045     state->ns_vcol += 3;
1046     state->ns_t2col += 3;
1047     state->ns_v2col += 2;
1048     PRINT_SUBTYPE(MSG_ORIG(MSG_CNOTE_T_PR_REG), pr_reg, dump_prgregset);
1049     state->ns_vcol -= 3;
1050     state->ns_t2col -= 3;
1051     state->ns_v2col -= 2;
1053     /*
1054     * The floating point register state is complex, and highly
1055     * platform dependent. For now, we simply display it as
1056     * a hex dump. This can be replaced if better information
1057     * is required.
1058     */
1059     if (data_present(state, &layout->pr_fpreg)) {
1060         indent_enter(state, MSG_ORIG(MSG_CNOTE_T_PR_FPREG),
1061                     &layout->pr_fpreg);
1062         dump_hex_bytes(layout->pr_fpreg.slf_offset + state->ns_data,
1063                       layout->pr_fpreg.slf_elllen, state->ns_indent, 4, 3);
1064         indent_exit(state);
1065     }
1067     indent_exit(state);
1068 }
1071 /*
1072 * Output information from pstatus_t structure.
1073 */
1074 static void
1075 dump_pstatus(note_state_t *state, const char *title)
1076 {
1077     const sl_pstatus_layout_t    *layout = state->ns_arch->pstatus;
1078     Word                          w;
1079     union {
1080         Conv_inv_buf_t           inv;
1081         Conv_cnote_pr_flags_buf_t flags;
1082     } conv_buf;
1084     indent_enter(state, title, &layout->pr_flags);
1086     if (data_present(state, &layout->pr_flags)) {
1087         w = extract_as_word(state, &layout->pr_flags);
1088         print_str(state, MSG_ORIG(MSG_CNOTE_T_PR_FLAGS),
1089                 conv_cnote_pr_flags(w, 0, &conv_buf.flags));
1090     }
1092     PRINT_DEC(MSG_ORIG(MSG_CNOTE_T_PR_NLWP), pr_nlwp);
1093     PRINT_DEC_2UP(MSG_ORIG(MSG_CNOTE_T_PR_PID), pr_pid,
1094                 MSG_ORIG(MSG_CNOTE_T_PR_PPID), pr_ppid);
1095     PRINT_DEC_2UP(MSG_ORIG(MSG_CNOTE_T_PR_PGID), pr_pgid,
1096                 MSG_ORIG(MSG_CNOTE_T_PR_SID), pr_sid);
1097     PRINT_DEC_2UP(MSG_ORIG(MSG_CNOTE_T_PR_ASLWPID), pr_aslwpid,
1098                 MSG_ORIG(MSG_CNOTE_T_PR_AGENTID), pr_agentid);

```

```

1099     PRINT_SUBTYPE(MSG_ORIG(MSG_CNOTE_T_PR_SIGPEND), pr_sigpend,
1100                 dump_sigset);
1101     print_num_2up(state, MSG_ORIG(MSG_CNOTE_T_PR_BRKBASE),
1102                 &layout->pr_brkbase, SL_FMT_NUM_ZHEX,
1103                 MSG_ORIG(MSG_CNOTE_T_PR_BRKSIZE),
1104                 &layout->pr_brksize, SL_FMT_NUM_HEX);
1105     print_num_2up(state, MSG_ORIG(MSG_CNOTE_T_PR_STKBASE),
1106                 &layout->pr_stkbase, SL_FMT_NUM_ZHEX,
1107                 MSG_ORIG(MSG_CNOTE_T_PR_STKSIZE),
1108                 &layout->pr_stksize, SL_FMT_NUM_HEX);
1109     PRINT_SUBTYPE(MSG_ORIG(MSG_CNOTE_T_PR_UTIME), pr_utime, dump_timestruc);
1110     PRINT_SUBTYPE(MSG_ORIG(MSG_CNOTE_T_PR_STIME), pr_stime, dump_timestruc);
1111     PRINT_SUBTYPE(MSG_ORIG(MSG_CNOTE_T_PR_CUTIME), pr_cutime,
1112                 dump_timestruc);
1113     PRINT_SUBTYPE(MSG_ORIG(MSG_CNOTE_T_PR_CSTIME), pr_cstime,
1114                 dump_timestruc);
1115     PRINT_SUBTYPE(MSG_ORIG(MSG_CNOTE_T_PR_SIGTRACE), pr_sigtrace,
1116                 dump_sigset);
1117     PRINT_SUBTYPE(MSG_ORIG(MSG_CNOTE_T_PR_FLTTRACE), pr_fltrace,
1118                 dumpfltset);
1119     PRINT_SUBTYPE(MSG_ORIG(MSG_CNOTE_T_PR_SYSENTRY), pr_sysentry,
1120                 dump_sysset);
1121     PRINT_SUBTYPE(MSG_ORIG(MSG_CNOTE_T_PR_SYSEXIT), pr_sysexit,
1122                 dump_sysset);
1124     if (data_present(state, &layout->pr_dmodel)) {
1125         w = extract_as_word(state, &layout->pr_dmodel);
1126         print_str(state, MSG_ORIG(MSG_CNOTE_T_PR_DMODEL),
1127                 conv_cnote_pr_dmodel(w, 0, &conv_buf.inv));
1128     }
1130     PRINT_DEC_2UP(MSG_ORIG(MSG_CNOTE_T_PR_TASKID), pr_taskid,
1131                 MSG_ORIG(MSG_CNOTE_T_PR_PROJID), pr_projid);
1132     PRINT_DEC_2UP(MSG_ORIG(MSG_CNOTE_T_PR_NZOMB), pr_nzomb,
1133                 MSG_ORIG(MSG_CNOTE_T_PR_ZONEID), pr_zoneid);
1135     /*
1136     * In order to line up all the values in a single column,
1137     * we would have to set vcol to a very high value, which results
1138     * in ugly looking output that runs off column 80. So, we use
1139     * two levels of vcol, one for the contents so far, and a
1140     * higher one for the pr_lwp sub-struct.
1141     */
1142     state->ns_vcol += 5;
1143     state->ns_t2col += 5;
1144     state->ns_v2col += 5;
1146 #endif /* ! codereview */
1147     PRINT_SUBTYPE(MSG_ORIG(MSG_CNOTE_T_PR_LWP), pr_lwp, dump_lwpstatus);
1148     state->ns_vcol -= 5;
1149     state->ns_t2col -= 5;
1150     state->ns_v2col -= 5;
1152     indent_exit(state);
1153 }
1156 /*
1157 * Output information from prstatus_t (<sys/old_procfs.h>) structure.
1158 */
1159 static void
1160 dump_prstatus(note_state_t *state, const char *title)
1161 {
1162     const sl_prstatus_layout_t    *layout = state->ns_arch->prstatus;
1163     Word                          w, w2;
1164     int                            i;

```

```

1165     union {
1166         Conv_inv_buf_t      inv;
1167         Conv_cnote_old_pr_flags_buf_t  flags;
1168     } conv_buf;
1170     indent_enter(state, title, &layout->pr_flags);
1172     if (data_present(state, &layout->pr_flags)) {
1173         w = extract_as_word(state, &layout->pr_flags);
1174         print_str(state, MSG_ORIG(MSG_CNOTE_T_PR_FLAGS),
1175                 conv_cnote_old_pr_flags(w, 0, &conv_buf.flags));
1176     }
1178     if (data_present(state, &layout->pr_why)) {
1179         w = extract_as_word(state, &layout->pr_why);
1180         print_str(state, MSG_ORIG(MSG_CNOTE_T_PR_WHY),
1181                 conv_cnote_pr_why(w, 0, &conv_buf.inv));
1184         if (data_present(state, &layout->pr_what)) {
1185             w2 = extract_as_word(state, &layout->pr_what);
1186             print_str(state, MSG_ORIG(MSG_CNOTE_T_PR_WHAT),
1187                     conv_cnote_pr_what(w, w2, 0, &conv_buf.inv));
1188         }
1189     }
1191     PRINT_SUBTYPE(MSG_ORIG(MSG_CNOTE_T_PR_INFO), pr_info, dump_siginfo);
1193     if (data_present(state, &layout->pr_cursig)) {
1194         w = extract_as_word(state, &layout->pr_cursig);
1195         print_str(state, MSG_ORIG(MSG_CNOTE_T_PR_CURSIG),
1196                 conv_cnote_signal(w, CONV_FMT_DECIMAL, &conv_buf.inv));
1197     }
1199     PRINT_DEC(MSG_ORIG(MSG_CNOTE_T_PR_NLWP), pr_nlwp);
1200     PRINT_SUBTYPE(MSG_ORIG(MSG_CNOTE_T_PR_SIGPEND), pr_sigpend,
1201                 dump_sigset);
1202     PRINT_SUBTYPE(MSG_ORIG(MSG_CNOTE_T_PR_SIGHOLD), pr_sighold,
1203                 dump_sigset);
1204     PRINT_SUBTYPE(MSG_ORIG(MSG_CNOTE_T_PR_ALTSTACK), pr_altstack,
1205                 dump_stack);
1206     PRINT_SUBTYPE(MSG_ORIG(MSG_CNOTE_T_PR_ACTION), pr_action,
1207                 dump_sigaction);
1208     PRINT_DEC_2UP(MSG_ORIG(MSG_CNOTE_T_PR_PID), pr_pid,
1209                 MSG_ORIG(MSG_CNOTE_T_PR_PPID), pr_ppid);
1210     PRINT_DEC_2UP(MSG_ORIG(MSG_CNOTE_T_PR_PGRP), pr_pgrp,
1211                 MSG_ORIG(MSG_CNOTE_T_PR_SID), pr_sid);
1212     PRINT_SUBTYPE(MSG_ORIG(MSG_CNOTE_T_PR_UTIME), pr_utime, dump_timestruc);
1213     PRINT_SUBTYPE(MSG_ORIG(MSG_CNOTE_T_PR_STIME), pr_stime, dump_timestruc);
1214     PRINT_SUBTYPE(MSG_ORIG(MSG_CNOTE_T_PR_CUTIME), pr_cutime,
1215                 dump_timestruc);
1216     PRINT_SUBTYPE(MSG_ORIG(MSG_CNOTE_T_PR_CSTIME), pr_cstime,
1217                 dump_timestruc);
1218     PRINT_STRBUF(MSG_ORIG(MSG_CNOTE_T_PR_CLNAME), pr_clname);
1220     if (data_present(state, &layout->pr_syscall)) {
1221         w = extract_as_word(state, &layout->pr_syscall);
1222         print_str(state, MSG_ORIG(MSG_CNOTE_T_PR_SYSCALL),
1223                 conv_cnote_syscall(w, CONV_FMT_DECIMAL, &conv_buf.inv));
1224     }
1226     PRINT_DEC(MSG_ORIG(MSG_CNOTE_T_PR_NSYSARG), pr_nsysarg);
1228     if (data_present(state, &layout->pr_nsysarg)) {
1229         w2 = extract_as_word(state, &layout->pr_nsysarg);
1230         print_array(state, &layout->pr_sysarg, SL_FMT_NUM_ZHEX, w2, 1,

```

```

1231         MSG_ORIG(MSG_CNOTE_T_PR_SYSARG));
1232     }
1234     PRINT_DEC(MSG_ORIG(MSG_CNOTE_T_PR_WHO), pr_who);
1235     PRINT_SUBTYPE(MSG_ORIG(MSG_CNOTE_T_PR_LWPPEND), pr_sigpend,
1236                 dump_sigset);
1237     PRINT_ZHEX(MSG_ORIG(MSG_CNOTE_T_PR_OLDCONTEXT), pr_oldcontext);
1238     print_num_2up(state, MSG_ORIG(MSG_CNOTE_T_PR_BRKBASE),
1239                 &layout->pr_brkbase, SL_FMT_NUM_ZHEX,
1240                 MSG_ORIG(MSG_CNOTE_T_PR_BRKSIZE),
1241                 &layout->pr_brksize, SL_FMT_NUM_HEX);
1242     print_num_2up(state, MSG_ORIG(MSG_CNOTE_T_PR_STKBASE),
1243                 &layout->pr_stkbase, SL_FMT_NUM_ZHEX,
1244                 MSG_ORIG(MSG_CNOTE_T_PR_STKSIZE),
1245                 &layout->pr_stksize, SL_FMT_NUM_HEX);
1246     PRINT_DEC(MSG_ORIG(MSG_CNOTE_T_PR_PROCESSOR), pr_processor);
1248     if (data_present(state, &layout->pr_bind)) {
1249         i = extract_as_word(state, &layout->pr_bind);
1250         print_str(state, MSG_ORIG(MSG_CNOTE_T_PR_BIND),
1251                 conv_cnote_psetid(i, CONV_FMT_DECIMAL, &conv_buf.inv));
1252     }
1254     PRINT_ZHEX(MSG_ORIG(MSG_CNOTE_T_PR_INSTR), pr_instr);
1255     PRINT_SUBTYPE(MSG_ORIG(MSG_CNOTE_T_PR_REG), pr_reg, dump_prgregset);
1257     indent_exit(state);
1258 }
1261 /*
1262  * Print percent from 16-bit binary fraction [0 .. 1]
1263  * Round up .01 to .1 to indicate some small percentage (the 0x7000 below).
1264  *
1265  * Note: This routine was copied from ps(1) and then modified.
1266  */
1267 static const char *
1268 prtpct_value(note_state_t *state, const sl_field_t *fdesc,
1269             sl_fmtbuf_t buf)
1270 {
1271     uint_t value;          /* need 32 bits to compute with */
1273     value = extract_as_word(state, fdesc);
1274     value = ((value * 1000) + 0x7000) >> 15;      /* [0 .. 1000] */
1275     if (value >= 1000)
1276         value = 999;
1278     (void) snprintf(buf, sizeof(sl_fmtbuf_t),
1279                 MSG_ORIG(MSG_CNOTE_FMT_P RTPCT), value / 10, value % 10);
1281     return (buf);
1282 }
1286 /*
1287  * Version of prtpct() used for a 2-up display of two adjacent percentages.
1288  */
1289 static void
1290 prtpct_2up(note_state_t *state, const sl_field_t *fdesc1,
1291            const char *title1, const sl_field_t *fdesc2, const char *title2)
1292 {
1293     sl_fmtbuf_t    buf1, buf2;
1295     if (!(data_present(state, fdesc1) &&
1296         data_present(state, fdesc2)))

```

```

1297         return;
1299     dbg_print(0, MSG_ORIG(MSG_CNOTE_FMT_LINE_2UP), INDENT,
1300             state->ns_vcol - state->ns_indent, title1,
1301             state->ns_t2col - state->ns_vcol,
1302             prtpct_value(state, fdesc1, buf1),
1303             state->ns_v2col - state->ns_t2col, title2,
1304             prtpct_value(state, fdesc2, buf2));
1305 }

1308 /*
1309  * The psinfo_t and prpsinfo_t structs have pr_state and pr_sname
1310  * fields that we wish to print in a 2up format. The pr_state is
1311  * an integer, while pr_sname is a single character.
1312  */
1313 static void
1314 print_state_sname_2up(note_state_t *state,
1315     const sl_field_t *state_fdesc,
1316     const sl_field_t *sname_fdesc)
1317 {
1318     sl_fmtbuf_t    buf1, buf2;
1319     int            sname;

1321     /*
1322     * If the field slf_offset and extent fall past the end of the
1323     * available data, then return without doing anything. That note
1324     * is from an older core file that doesn't have all the fields
1325     * that we know about.
1326     */
1327     if (!(data_present(state, state_fdesc) &&
1328         data_present(state, sname_fdesc)))
1329         return;

1331     sname = extract_as_sword(state, sname_fdesc);
1332     buf2[0] = sname;
1333     buf2[1] = '\0';

1335     dbg_print(0, MSG_ORIG(MSG_CNOTE_FMT_LINE_2UP), INDENT,
1336             state->ns_vcol - state->ns_indent, MSG_ORIG(MSG_CNOTE_T_PR_STATE),
1337             state->ns_t2col - state->ns_vcol,
1338             fmt_num(state, state_fdesc, SL_FMT_NUM_DEC, buf1),
1339             state->ns_v2col - state->ns_t2col, MSG_ORIG(MSG_CNOTE_T_PR_SNAME),
1340             buf2);
1341 }

1343 /*
1344  * Output information from lwpsinfo_t structure.
1345  */
1346 static void
1347 dump_lwpsinfo(note_state_t *state, const char *title)
1348 {
1349     const sl_lwpsinfo_layout_t    *layout = state->ns_arch->lwpsinfo;
1350     Word                            w;
1351     int32_t                          i;
1352     union {
1353         Conv_cnote_proc_flag_buf_t    proc_flag;
1354         Conv_inv_buf_t                inv;
1355     } conv_buf;

1357     indent_enter(state, title, &layout->pr_flag);

1359     if (data_present(state, &layout->pr_flag)) {
1360         w = extract_as_word(state, &layout->pr_flag);
1361         print_str(state, MSG_ORIG(MSG_CNOTE_T_PR_FLAG),
1362             conv_cnote_proc_flag(w, 0, &conv_buf.proc_flag));

```

```

1363     }

1365     print_num_2up(state, MSG_ORIG(MSG_CNOTE_T_PR_LWPID), &layout->pr_lwpid,
1366         SL_FMT_NUM_DEC, MSG_ORIG(MSG_CNOTE_T_PR_ADDR), &layout->pr_addr,
1367         SL_FMT_NUM_ZHEX);
1368     PRINT_HEX(MSG_ORIG(MSG_CNOTE_T_PR_WCHAN), pr_wchan);

1370     if (data_present(state, &layout->pr_stype)) {
1371         w = extract_as_word(state, &layout->pr_stype);
1372         print_str(state, MSG_ORIG(MSG_CNOTE_T_PR_STYPE),
1373             conv_cnote_pr_stype(w, CONV_FMT_DECIMAL, &conv_buf.inv));
1374     }

1376     print_state_sname_2up(state, &layout->pr_state, &layout->pr_sname);

1378     PRINT_DEC(MSG_ORIG(MSG_CNOTE_T_PR_NICE), pr_nice);

1380     if (data_present(state, &layout->pr_syscall)) {
1381         w = extract_as_word(state, &layout->pr_syscall);
1382         print_str(state, MSG_ORIG(MSG_CNOTE_T_PR_SYSCALL),
1383             conv_cnote_syscall(w, CONV_FMT_DECIMAL, &conv_buf.inv));
1384     }

1386     PRINT_DEC_2UP(MSG_ORIG(MSG_CNOTE_T_PR_OLDPRI), pr_oldpri,
1387         MSG_ORIG(MSG_CNOTE_T_PR_CPU), pr_cpu);

1389     if (data_present(state, &layout->pr_pri) &&
1390         data_present(state, &layout->pr_pctcpu)) {
1391         sl_fmtbuf_t    buf1, buf2;

1393         dbg_print(0, MSG_ORIG(MSG_CNOTE_FMT_LINE_2UP), INDENT,
1394             state->ns_vcol - state->ns_indent,
1395             MSG_ORIG(MSG_CNOTE_T_PR_PRI),
1396             state->ns_t2col - state->ns_vcol,
1397             fmt_num(state, &layout->pr_pri, SL_FMT_NUM_DEC, buf1),
1398             state->ns_v2col - state->ns_t2col,
1399             MSG_ORIG(MSG_CNOTE_T_PR_PCTCPU),
1400             prtpct_value(state, &layout->pr_pctcpu, buf2));
1401     }

1403     PRINT_SUBTYPE(MSG_ORIG(MSG_CNOTE_T_PR_START), pr_start, dump_timestruc);
1404     PRINT_SUBTYPE(MSG_ORIG(MSG_CNOTE_T_PR_TIME), pr_time, dump_timestruc);
1405     PRINT_STRBUF(MSG_ORIG(MSG_CNOTE_T_PR_CLNAME), pr_clname);
1406     PRINT_STRBUF(MSG_ORIG(MSG_CNOTE_T_PR_NAME), pr_name);
1407     PRINT_DEC_2UP(MSG_ORIG(MSG_CNOTE_T_PR_ONPRO), pr_onpro,
1408         MSG_ORIG(MSG_CNOTE_T_PR_BINDPRO), pr_bindpro);

1410     if (data_present(state, &layout->pr_bindpset)) {
1411         i = extract_as_sword(state, &layout->pr_bindpset);
1412         print_str(state, MSG_ORIG(MSG_CNOTE_T_PR_BINDPSET),
1413             conv_cnote_psetid(i, CONV_FMT_DECIMAL, &conv_buf.inv));
1414     }

1416     PRINT_DEC(MSG_ORIG(MSG_CNOTE_T_PR_LGRP), pr_lgrp);

1418     indent_exit(state);
1419 }

1422 /*
1423  * Output information from psinfo_t structure.
1424  */
1425 static void
1426 dump_psinfo(note_state_t *state, const char *title)
1427 {
1428     const sl_psinfo_layout_t    *layout = state->ns_arch->psinfo;

```

```

1429     Word                               w;
1430     union {
1431         Conv_cnote_proc_flag_buf_t     proc_flag;
1432         Conv_inv_buf_t                 inv;
1433     } conv_buf;
1434
1435     indent_enter(state, title, &layout->pr_flag);
1436
1437     if (data_present(state, &layout->pr_flag)) {
1438         w = extract_as_word(state, &layout->pr_flag);
1439         print_str(state, MSG_ORIG(MSG_CNOTE_T_PR_FLAG),
1440                 conv_cnote_proc_flag(w, 0, &conv_buf.proc_flag));
1441     }
1442
1443     PRINT_DEC(MSG_ORIG(MSG_CNOTE_T_PR_NLWP), pr_nlwp);
1444     PRINT_DEC_2UP(MSG_ORIG(MSG_CNOTE_T_PR_PID), pr_pid,
1445                 MSG_ORIG(MSG_CNOTE_T_PR_PPID), pr_ppid);
1446     PRINT_DEC_2UP(MSG_ORIG(MSG_CNOTE_T_PR_PGID), pr_pgid,
1447                 MSG_ORIG(MSG_CNOTE_T_PR_SID), pr_sid);
1448     PRINT_DEC_2UP(MSG_ORIG(MSG_CNOTE_T_PR_UID), pr_uid,
1449                 MSG_ORIG(MSG_CNOTE_T_PR_EUID), pr_euid);
1450     PRINT_DEC_2UP(MSG_ORIG(MSG_CNOTE_T_PR_GID), pr_gid,
1451                 MSG_ORIG(MSG_CNOTE_T_PR_EGID), pr_egid);
1452     print_num_2up(state, MSG_ORIG(MSG_CNOTE_T_PR_ADDR), &layout->pr_addr,
1453                 SL_FMT_NUM_ZHEX, MSG_ORIG(MSG_CNOTE_T_PR_SIZE), &layout->pr_size,
1454                 SL_FMT_NUM_HEX);
1455     print_num_2up(state, MSG_ORIG(MSG_CNOTE_T_PR_RSSIZE),
1456                 &layout->pr_rssize, SL_FMT_NUM_HEX, MSG_ORIG(MSG_CNOTE_T_PR_TTYDEV),
1457                 &layout->pr_ttydev, SL_FMT_NUM_DEC);
1458     prtpct_2up(state, &layout->pr_pctcpu, MSG_ORIG(MSG_CNOTE_T_PR_PCTCPU),
1459                 &layout->pr_pctmem, MSG_ORIG(MSG_CNOTE_T_PR_PCTMEM));
1460     PRINT_SUBTYPE(MSG_ORIG(MSG_CNOTE_T_PR_START), pr_start, dump_timestruc);
1461     PRINT_SUBTYPE(MSG_ORIG(MSG_CNOTE_T_PR_TIME), pr_time, dump_timestruc);
1462     PRINT_SUBTYPE(MSG_ORIG(MSG_CNOTE_T_PR_CTIME), pr_ctime, dump_timestruc);
1463     PRINT_STRBUF(MSG_ORIG(MSG_CNOTE_T_PR_FNAME), pr_fname);
1464     PRINT_STRBUF(MSG_ORIG(MSG_CNOTE_T_PR_PSARGS), pr_psargs);
1465     print_num_2up(state, MSG_ORIG(MSG_CNOTE_T_PR_WSTAT), &layout->pr_wstat,
1466                 SL_FMT_NUM_HEX, MSG_ORIG(MSG_CNOTE_T_PR_ARGC), &layout->pr_argc,
1467                 SL_FMT_NUM_DEC);
1468     PRINT_ZHEX_2UP(MSG_ORIG(MSG_CNOTE_T_PR_ARGV), pr_argv,
1469                 MSG_ORIG(MSG_CNOTE_T_PR_ENVV), pr_envv);
1470
1471     if (data_present(state, &layout->pr_dmodel)) {
1472         w = extract_as_word(state, &layout->pr_dmodel);
1473         print_str(state, MSG_ORIG(MSG_CNOTE_T_PR_DMODEL),
1474                 conv_cnote_pr_dmodel(w, 0, &conv_buf.inv));
1475     }
1476
1477     PRINT_DEC_2UP(MSG_ORIG(MSG_CNOTE_T_PR_TASKID), pr_taskid,
1478                 MSG_ORIG(MSG_CNOTE_T_PR_PROJID), pr_projid);
1479     PRINT_DEC_2UP(MSG_ORIG(MSG_CNOTE_T_PR_NZOMB), pr_nzomb,
1480                 MSG_ORIG(MSG_CNOTE_T_PR_POOLID), pr_poolid);
1481     PRINT_DEC_2UP(MSG_ORIG(MSG_CNOTE_T_PR_ZONEID), pr_zoneid,
1482                 MSG_ORIG(MSG_CNOTE_T_PR_CONTRACT), pr_contract);
1483
1484     PRINT_SUBTYPE(MSG_ORIG(MSG_CNOTE_T_PR_LWP), pr_lwp, dump_lwpsinfo);
1485
1486     indent_exit(state);
1487 }
1488
1489 /*
1490  * Output information from prpsinfo_t structure.
1491  */
1492 static void
1493 dump_prpsinfo(note_state_t *state, const char *title)
1494 {

```

```

1495     const sl_prpsinfo_layout_t         *layout = state->ns_arch->prpsinfo;
1496     Word                               w;
1497     union {
1498         Conv_cnote_proc_flag_buf_t     proc_flag;
1499         Conv_inv_buf_t                 inv;
1500     } conv_buf;
1501
1502     indent_enter(state, title, &layout->pr_state);
1503
1504     print_state_sname_2up(state, &layout->pr_state, &layout->pr_sname);
1505     PRINT_DEC_2UP(MSG_ORIG(MSG_CNOTE_T_PR_ZOMB), pr_zomb,
1506                 MSG_ORIG(MSG_CNOTE_T_PR_NICE), pr_nice);
1507
1508     if (data_present(state, &layout->pr_flag)) {
1509         w = extract_as_word(state, &layout->pr_flag);
1510         print_str(state, MSG_ORIG(MSG_CNOTE_T_PR_FLAG),
1511                 conv_cnote_proc_flag(w, 0, &conv_buf.proc_flag));
1512     }
1513
1514     PRINT_DEC_2UP(MSG_ORIG(MSG_CNOTE_T_PR_UID), pr_uid,
1515                 MSG_ORIG(MSG_CNOTE_T_PR_GID), pr_gid);
1516     PRINT_DEC_2UP(MSG_ORIG(MSG_CNOTE_T_PR_PID), pr_pid,
1517                 MSG_ORIG(MSG_CNOTE_T_PR_PPID), pr_ppid);
1518     PRINT_DEC_2UP(MSG_ORIG(MSG_CNOTE_T_PR_PGRP), pr_pgrp,
1519                 MSG_ORIG(MSG_CNOTE_T_PR_SID), pr_sid);
1520     print_num_2up(state, MSG_ORIG(MSG_CNOTE_T_PR_ADDR), &layout->pr_addr,
1521                 SL_FMT_NUM_ZHEX, MSG_ORIG(MSG_CNOTE_T_PR_SIZE), &layout->pr_size,
1522                 SL_FMT_NUM_HEX);
1523     PRINT_HEX_2UP(MSG_ORIG(MSG_CNOTE_T_PR_RSSIZE), pr_rssize,
1524                 MSG_ORIG(MSG_CNOTE_T_PR_WCHAN), pr_wchan);
1525     PRINT_SUBTYPE(MSG_ORIG(MSG_CNOTE_T_PR_START), pr_start, dump_timestruc);
1526     PRINT_SUBTYPE(MSG_ORIG(MSG_CNOTE_T_PR_TIME), pr_time, dump_timestruc);
1527     PRINT_DEC_2UP(MSG_ORIG(MSG_CNOTE_T_PR_PRI), pr_pri,
1528                 MSG_ORIG(MSG_CNOTE_T_PR_OLDPRI), pr_olddpri);
1529     PRINT_DEC(MSG_ORIG(MSG_CNOTE_T_PR_CPU), pr_cpu);
1530     PRINT_DEC_2UP(MSG_ORIG(MSG_CNOTE_T_PR_OTTYDEV), pr_ottydev,
1531                 MSG_ORIG(MSG_CNOTE_T_PR_LTTYDEV), pr_lttydev);
1532     PRINT_STRBUF(MSG_ORIG(MSG_CNOTE_T_PR_CLNAME), pr_clname);
1533     PRINT_STRBUF(MSG_ORIG(MSG_CNOTE_T_PR_FNAME), pr_fname);
1534     PRINT_STRBUF(MSG_ORIG(MSG_CNOTE_T_PR_PSARGS), pr_psargs);
1535
1536     if (data_present(state, &layout->pr_syscall)) {
1537         w = extract_as_word(state, &layout->pr_syscall);
1538         print_str(state, MSG_ORIG(MSG_CNOTE_T_PR_SYSCALL),
1539                 conv_cnote_syscall(w, CONV_FMT_DECIMAL, &conv_buf.inv));
1540     }
1541
1542     PRINT_SUBTYPE(MSG_ORIG(MSG_CNOTE_T_PR_CTIME), pr_ctime, dump_timestruc);
1543     PRINT_HEX_2UP(MSG_ORIG(MSG_CNOTE_T_PR_BYSIZE), pr_bysize,
1544                 MSG_ORIG(MSG_CNOTE_T_PR_BYRSSIZE), pr_byrssize);
1545     print_num_2up(state, MSG_ORIG(MSG_CNOTE_T_PR_ARGC), &layout->pr_argc,
1546                 SL_FMT_NUM_DEC, MSG_ORIG(MSG_CNOTE_T_PR_ARGV), &layout->pr_argv,
1547                 SL_FMT_NUM_ZHEX);
1548     print_num_2up(state, MSG_ORIG(MSG_CNOTE_T_PR_ENVV), &layout->pr_envv,
1549                 SL_FMT_NUM_ZHEX, MSG_ORIG(MSG_CNOTE_T_PR_WSTAT), &layout->pr_wstat,
1550                 SL_FMT_NUM_HEX);
1551     prtpct_2up(state, &layout->pr_pctcpu, MSG_ORIG(MSG_CNOTE_T_PR_PCTCPU),
1552                 &layout->pr_pctmem, MSG_ORIG(MSG_CNOTE_T_PR_PCTMEM));
1553     PRINT_DEC_2UP(MSG_ORIG(MSG_CNOTE_T_PR_EUID), pr_euid,
1554                 MSG_ORIG(MSG_CNOTE_T_PR_EGID), pr_egid);
1555     PRINT_DEC(MSG_ORIG(MSG_CNOTE_T_PR_ASLEPID), pr_aslepid);
1556
1557     if (data_present(state, &layout->pr_dmodel)) {
1558         w = extract_as_word(state, &layout->pr_dmodel);
1559         print_str(state, MSG_ORIG(MSG_CNOTE_T_PR_DMODEL),
1560

```



```

1561         conv_cnote_pr_dmodel(w, 0, &conv_buf.inv));
1562     }

1564     indent_exit(state);
1565 }

1568 /*
1569  * Output information from prcred_t structure.
1570  */
1571 static void
1572 dump_prcred(note_state_t *state, const char *title)
1573 {
1574     const sl_prcred_layout_t *layout = state->ns_arch->prcred;
1575     Word                     ngroups;

1577     indent_enter(state, title, &layout->pr_euid);

1579     PRINT_DEC_2UP(MSG_ORIG(MSG_CNOTE_T_PR_EUID), pr_euid,
1580                 MSG_ORIG(MSG_CNOTE_T_PR_RUID), pr_ruid);
1581     PRINT_DEC_2UP(MSG_ORIG(MSG_CNOTE_T_PR_SUID), pr_suid,
1582                 MSG_ORIG(MSG_CNOTE_T_PR_EGID), pr_egid);
1583     PRINT_DEC_2UP(MSG_ORIG(MSG_CNOTE_T_PR_RGID), pr_rgid,
1584                 MSG_ORIG(MSG_CNOTE_T_PR_SGID), pr_sgid);
1585     PRINT_DEC(MSG_ORIG(MSG_CNOTE_T_PR_NGROUPS), pr_ngroups);

1587     if (data_present(state, &layout->pr_ngroups)) {
1588         ngroups = extract_as_word(state, &layout->pr_ngroups);
1589         print_array(state, &layout->pr_groups, SL_FMT_NUM_DEC, ngroups,
1590                   0, MSG_ORIG(MSG_CNOTE_T_PR_GROUPS));
1591     }

1593     indent_exit(state);
1594 }

1597 /*
1598  * Output information from prpriv_t structure.
1599  */
1600 static void
1601 dump_prpriv(note_state_t *state, const char *title)
1602 {
1603     const sl_prpriv_layout_t *layout = state->ns_arch->prpriv;
1604     Word                     nsets;

1606     indent_enter(state, title, &layout->pr_nsets);

1608     PRINT_DEC(MSG_ORIG(MSG_CNOTE_T_PR_NSETS), pr_nsets);
1609     PRINT_HEX(MSG_ORIG(MSG_CNOTE_T_PR_SETSIZE), pr_setsize);
1610     PRINT_HEX(MSG_ORIG(MSG_CNOTE_T_PR_INFOSIZE), pr_infosize);

1612     if (data_present(state, &layout->pr_nsets)) {
1613         nsets = extract_as_word(state, &layout->pr_nsets);
1614         print_array(state, &layout->pr_sets, SL_FMT_NUM_ZHEX, nsets,
1615                   0, MSG_ORIG(MSG_CNOTE_T_PR_SETS));
1616     }

1618     indent_exit(state);
1619 }

1621 static void
1622 dump_prfdinfo(note_state_t *state, const char *title)
1623 {
1624     const sl_prfdinfo_layout_t *layout = state->ns_arch->prfdinfo;
1625     char buf[1024];
1626     uint32_t fileflags, mode;

```

```

1628     indent_enter(state, title, &layout->pr_fd);

1630     PRINT_DEC(MSG_ORIG(MSG_CNOTE_T_PR_FD), pr_fd);
1631     mode = extract_as_word(state, &layout->pr_mode);

1633     print_str(state, MSG_ORIG(MSG_CNOTE_T_PR_MODE),
1634              conv_cnote_filemode(mode, 0, buf, sizeof (buf)));

1636     PRINT_DEC_2UP(MSG_ORIG(MSG_CNOTE_T_PR_UID), pr_uid,
1637                 MSG_ORIG(MSG_CNOTE_T_PR_GID), pr_gid);

1639     PRINT_DEC_2UP(MSG_ORIG(MSG_CNOTE_T_PR_MAJOR), pr_major,
1640                 MSG_ORIG(MSG_CNOTE_T_PR_MINOR), pr_minor);
1641     PRINT_DEC_2UP(MSG_ORIG(MSG_CNOTE_T_PR_RMAJOR), pr_rmajor,
1642                 MSG_ORIG(MSG_CNOTE_T_PR_RMINOR), pr_rminor);

1644     PRINT_DEC(MSG_ORIG(MSG_CNOTE_T_PR_INO), pr_ino);

1646     PRINT_DEC_2UP(MSG_ORIG(MSG_CNOTE_T_PR_SIZE), pr_size,
1647                 MSG_ORIG(MSG_CNOTE_T_PR_OFFSET), pr_offset);

1649     fileflags = extract_as_word(state, &layout->pr_fileflags);

1651     print_str(state, MSG_ORIG(MSG_CNOTE_T_PR_FILEFLAGS),
1652              conv_cnote_fileflags(fileflags, 0, buf, sizeof (buf)));

1654     PRINT_DEC(MSG_ORIG(MSG_CNOTE_T_PR_FD_FLAGS), pr_fdflags);

1656     PRINT_STRBUF(MSG_ORIG(MSG_CNOTE_T_PR_PATH), pr_path);

1658     indent_exit(state);
1659 }

1661 /*
1662  * Output information from priv_impl_info_t structure.
1663  */
1664 static void
1665 dump_priv_impl_info(note_state_t *state, const char *title)
1666 {
1667     const sl_priv_impl_info_layout_t *layout;

1669     layout = state->ns_arch->priv_impl_info;
1670     indent_enter(state, title, &layout->priv_headersize);

1672     PRINT_HEX_2UP(MSG_ORIG(MSG_CNOTE_T_PRIV_HEADERSIZE), priv_headersize,
1673                 MSG_ORIG(MSG_CNOTE_T_PRIV_FLAGS), priv_flags);

1675     print_num_2up(state, MSG_ORIG(MSG_CNOTE_T_PRIV_NSETS),
1676                 &layout->priv_nsets, SL_FMT_NUM_DEC,
1677                 MSG_ORIG(MSG_CNOTE_T_PRIV_SETSIZE), &layout->priv_setsize,
1678                 SL_FMT_NUM_HEX);
1679     print_num_2up(state, MSG_ORIG(MSG_CNOTE_T_PRIV_MAX), &layout->priv_max,
1680                 SL_FMT_NUM_DEC, MSG_ORIG(MSG_CNOTE_T_PRIV_INFOSIZE),
1681                 &layout->priv_infosize, SL_FMT_NUM_HEX);
1682     PRINT_HEX(MSG_ORIG(MSG_CNOTE_T_PRIV_GLOBALINFOSIZE),
1683              priv_globalinfosize);

1685     indent_exit(state);
1686 }

1689 /*
1690  * Dump information from an asrset_t array. This data
1691  * structure is specific to sparcv9, and does not appear
1692  * on any other platform.

```

```

1693 *
1694 * asrset_t is a simple array, defined in <sys/regset.h> as
1695 *   typedef int64_t asrset_t[16];   %asr16 -> %asr31
1696 *
1697 * As such, we do not make use of the struct_layout facilities
1698 * for this routine.
1699 */
1700 static void
1701 dump_asrset(note_state_t *state, const char *title)
1702 {
1703     static const sl_field_t ftemplate = { 0, sizeof (int64_t), 16, 0 };
1704     sl_field_t      fdesc1, fdesc2;
1705     sl_fmtbuf_t     buf1, buf2;
1706     char            index1[MAXNDXSIZE * 2], index2[MAXNDXSIZE * 2];
1707     Word            w, nelts;
1708
1709     fdesc1 = fdesc2 = ftemplate;
1710
1711     /* We expect 16 values, but will print whatever is actually there */
1712     nelts = state->ns_len / ftemplate.slf_eltlen;
1713     if (nelts == 0)
1714         return;
1715
1716     indent_enter(state, title, &fdesc1);
1717
1718     for (w = 0; w < nelts; ) {
1719         (void) snprintf(index1, sizeof (index1),
1720             MSG_ORIG(MSG_FMT_ASRINDEX), w + 16);
1721
1722         if (w == (nelts - 1)) {
1723             /* One last register is left */
1724             dbg_print(0, MSG_ORIG(MSG_CNNOTE_FMT_LINE),
1725                 INDENT, state->ns_vcol - state->ns_indent, index1,
1726                 fmt_num(state, &fdesc1, SL_FMT_NUM_ZHEX, buf1));
1727             fdesc1.slf_offset += fdesc1.slf_eltlen;
1728             w++;
1729             continue;
1730         }
1731
1732         /* There are at least 2 more registers left. Show 2 up */
1733         (void) snprintf(index2, sizeof (index2),
1734             MSG_ORIG(MSG_FMT_ASRINDEX), w + 17);
1735
1736         fdesc2.slf_offset = fdesc1.slf_offset + fdesc1.slf_eltlen;
1737         dbg_print(0, MSG_ORIG(MSG_CNNOTE_FMT_LINE_2UP), INDENT,
1738             state->ns_vcol - state->ns_indent, index1,
1739             state->ns_t2col - state->ns_vcol,
1740             fmt_num(state, &fdesc1, SL_FMT_NUM_ZHEX, buf1),
1741             state->ns_v2col - state->ns_t2col, index2,
1742             fmt_num(state, &fdesc2, SL_FMT_NUM_ZHEX, buf2));
1743         fdesc1.slf_offset += 2 * fdesc1.slf_eltlen;
1744         w += 2;
1745     }
1746
1747     indent_exit(state);
1748 }
1749
1750 corenote_ret_t
1751 corenote(Half mach, int do_swap, Word type,
1752     const char *desc, Word descsz)
1753 {
1754     note_state_t      state;
1755
1756     /*
1757      * Get the per-architecture layout definition
1758      */

```

```

1759     state.ns_mach = mach;
1760     state.ns_arch = sl_mach(state.ns_mach);
1761     if (sl_mach(state.ns_mach) == NULL)
1762         return (CORENOTE_R_BADARCH);
1763
1764     state.ns_swap = do_swap;
1765     state.ns_indent = 4;
1766     state.ns_t2col = state.ns_v2col = 0;
1767     state.ns_data = desc;
1768     state.ns_len = descsz;
1769
1770     switch (type) {
1771     case NT_PRSTATUS: /* prstatus_t <sys/old_procfs.h> */
1772         state.ns_vcol = 26;
1773         state.ns_t2col = 46;
1774         state.ns_v2col = 60;
1775         dump_prstatus(&state, MSG_ORIG(MSG_CNNOTE_DESC_PRSTATUS_T));
1776         return (CORENOTE_R_OK);
1777
1778     case NT_PRFPREG: /* prfpregset_t <sys/procfs_isa.h> */
1779         return (CORENOTE_R_OK_DUMP);
1780
1781     case NT_PRPSINFO: /* prpsinfo_t <sys/old_procfs.h> */
1782         state.ns_vcol = 20;
1783         state.ns_t2col = 41;
1784         state.ns_v2col = 54;
1785         dump_prpsinfo(&state, MSG_ORIG(MSG_CNNOTE_DESC_PRPSINFO_T));
1786         return (CORENOTE_R_OK);
1787
1788     case NT_PRRREG: /* prxregset_t <sys/procfs_isa.h> */
1789         return (CORENOTE_R_OK_DUMP);
1790
1791     case NT_PLATFORM: /* string from sysinfo(SI_PLATFORM) */
1792         dbg_print(0, MSG_ORIG(MSG_CNNOTE_DESC));
1793         dbg_print(0, MSG_ORIG(MSG_FMT_INDENT), safe_str(desc, descsz));
1794         return (CORENOTE_R_OK);
1795
1796     case NT_AUXV: /* auxv_t array <sys/auxv.h> */
1797         state.ns_vcol = 18;
1798         dump_auxv(&state, MSG_ORIG(MSG_CNNOTE_DESC_AUXV_T));
1799         return (CORENOTE_R_OK);
1800
1801     case NT_GWINDOWS: /* gwindows_t SPARC only */
1802         return (CORENOTE_R_OK_DUMP);
1803
1804     case NT_ASRS: /* asrset_t <sys/regset> sparcv9 only */
1805         state.ns_vcol = 18;
1806         state.ns_t2col = 38;
1807         state.ns_v2col = 46;
1808         dump_asrset(&state, MSG_ORIG(MSG_CNNOTE_DESC_ASRSET_T));
1809         return (CORENOTE_R_OK);
1810
1811     case NT_LDT: /* ssd array <sys/sysi86.h> IA32 only */
1812         return (CORENOTE_R_OK_DUMP);
1813
1814     case NT_PSTATUS: /* pstatus_t <sys/procfs.h> */
1815         state.ns_vcol = 22;
1816         state.ns_t2col = 42;
1817         state.ns_v2col = 54;
1818         dump_pstatus(&state, MSG_ORIG(MSG_CNNOTE_DESC_PSTATUS_T));
1819         return (CORENOTE_R_OK);
1820
1821     case NT_PSINFO: /* psinfo_t <sys/procfs.h> */
1822         state.ns_vcol = 25;
1823         state.ns_t2col = 45;
1824         state.ns_v2col = 58;

```

```

1825     dump_psinfo(&state, MSG_ORIG(MSG_CNOTE_DESC_PSINFO_T));
1826     return (CORENOTE_R_OK);

1828     case NT_PRCRED:           /* prcred_t <sys/procfs.h> */
1829         state.ns_vcol = 20;
1830         state.ns_t2col = 34;
1831         state.ns_v2col = 44;
1832         dump_prcred(&state, MSG_ORIG(MSG_CNOTE_DESC_PRCRED_T));
1833         return (CORENOTE_R_OK);

1835     case NT_UTSNAME:         /* struct utsname <sys/utsname.h> */
1836         state.ns_vcol = 18;
1837         dump_utsname(&state, MSG_ORIG(MSG_CNOTE_DESC_STRUCT_UTSNAME));
1838         return (CORENOTE_R_OK);

1840     case NT_LWPSTATUS:       /* lwpstatus_t <sys/procfs.h> */
1841         state.ns_vcol = 24;
1842         state.ns_t2col = 44;
1843         state.ns_v2col = 54;
1844         dump_lwpstatus(&state, MSG_ORIG(MSG_CNOTE_DESC_LWPSTATUS_T));
1845         return (CORENOTE_R_OK);

1847     case NT_LWPSINFO:        /* lwpsinfo_t <sys/procfs.h> */
1848         state.ns_vcol = 22;
1849         state.ns_t2col = 42;
1850         state.ns_v2col = 54;
1851         dump_lwpsinfo(&state, MSG_ORIG(MSG_CNOTE_DESC_LWPSINFO_T));
1852         return (CORENOTE_R_OK);

1854     case NT_PRPRIV:          /* prpriv_t <sys/procfs.h> */
1855         state.ns_vcol = 21;
1856         state.ns_t2col = 34;
1857         state.ns_v2col = 38;
1858         dump_prpriv(&state, MSG_ORIG(MSG_CNOTE_DESC_PRPRIV_T));
1859         return (CORENOTE_R_OK);

1861     case NT_PRPRIVINFO:      /* priv_impl_info_t <sys/priv.h> */
1862         state.ns_vcol = 29;
1863         state.ns_t2col = 41;
1864         state.ns_v2col = 56;
1865         dump_priv_impl_info(&state,
1866             MSG_ORIG(MSG_CNOTE_DESC_PRIV_IMPL_INFO_T));
1867         return (CORENOTE_R_OK);

1869     case NT_CONTENT:         /* core_content_t <sys/corectl.h> */
1870         if (sizeof (core_content_t) > descsz)
1871             return (CORENOTE_R_BADDATA);
1872         {
1873             static sl_field_t fdesc = { 0, 8, 0, 0 };
1874             Conv_cnote_cc_content_buf_t conv_buf;
1875             core_content_t content;

1877             state.ns_vcol = 8;
1878             indent_enter(&state,
1879                 MSG_ORIG(MSG_CNOTE_DESC_CORE_CONTENT_T),
1880                 &fdesc);
1881             content = extract_as_lword(&state, &fdesc);
1882             print_str(&state, MSG_ORIG(MSG_STR_EMPTY),
1883                 conv_cnote_cc_content(content, 0, &conv_buf));
1884             indent_exit(&state);
1885         }
1886         return (CORENOTE_R_OK);

1888     case NT_ZONENAME:        /* string from getzonenamebyid(3C) */
1889         dbg_print(0, MSG_ORIG(MSG_NOTE_DESC));
1890         dbg_print(0, MSG_ORIG(MSG_FMT_INDENT), safe_str(desc, descsz));

```

```

1891         return (CORENOTE_R_OK);

1894     case NT_FDINFO:          /* fdinfo_t <sys/procfs.h> */
1895         state.ns_vcol = 22;
1896         state.ns_t2col = 41;
1897         state.ns_v2col = 54;
1898         dump_prfdinfo(&state, MSG_ORIG(MSG_CNOTE_DESC_PRFDINFO_T));
1899         return (CORENOTE_R_OK);

1901     case NT_SPYMASTER:      /* spymaster_t <sys/procfs.h> */
1902         state.ns_vcol = 25;
1903         state.ns_t2col = 45;
1904         state.ns_v2col = 58;
1905         dump_psinfo(&state, MSG_ORIG(MSG_CNOTE_DESC_PSINFO_T));
1906         return (CORENOTE_R_OK);

1908     case NT_SECFLAGS:        /* secflags_t <sys/procfs.h> */
1909         state.ns_vcol = 23;
1910         state.ns_t2col = 41;
1911         state.ns_v2col = 54;
1912         dump_secflags(&state, MSG_ORIG(MSG_CNOTE_DESC_PRSECFLAGS_T));
1913         return (CORENOTE_R_OK);
1914 #endif /* ! codereview */
1915     }

1917     return (CORENOTE_R_BADTYPE);
1918 }

```



```

125 @ MSG_ERR_BADHASH      "%s: %s: bad hash entry: symbol %s: exists in bucket \
126                          %d, should be bucket %ld\n"
127 @ MSG_ERR_NODYNSYM     "%s: %s: associated SHT_DYNSYM section not found\n"
128 @ MSG_ERR_BADNDXSEC    "%s: %s: unexpected section type associated with \
129                          index section: %s\n"
130 @ MSG_ERR_BADSYMNDX     "%s: %s: bad symbol index: %d\n"
131 @ MSG_ERR_BADVER        "%s: %s: index[%d]: version %d is out of range: \
132                          version definitions available: 0-%d\n"
133 @ MSG_ERR_NOTSTRTAB     "%s: section[%d] is not a string table as expected \
134                          by section[%d]\n";

136 @ MSG_ERR_LDYNNOTADJ   "%s: bad dynamic symbol table layout: %s and %s \
137                          sections are not adjacent\n"
138 @ MSG_ERR_SECMEMOVER    "%s: memory overlap between section[%d]: %s: %llx:%llx \
139                          and section[%d]: %s: %llx:%llx\n"
140 @ MSG_ERR_SHDRMEMOVER   "%s: memory overlap between section header table: \
141                          %llx:%llx and section[%d]: %s: %llx:%llx\n"
142 @ MSG_ERR_MULTDYN       "%s: %d dynamic sections seen (1 expected)\n"
143 @ MSG_ERR_DYNNOBCKSEC   "%s: object lacks %s section required by %s dynamic \
144                          entry\n"
145 @ MSG_ERR_DYNBADADDR    "%s: %s (%#llx) does not match \
146                          shdr[%d: %s].sh_addr (%#llx)\n"
147 @ MSG_ERR_DYNBADSIZE    "%s: %s (%#llx) does not match \
148                          shdr[%d: %s].sh_size (%#llx)\n"
149 @ MSG_ERR_DYNBADENTSIZE "%s: %s (%#llx) does not match \
150                          shdr[%d: %s].sh_entsize (%#llx)\n"
151 @ MSG_ERR_DYNSYMVAL     "%s: %s: symbol value does not match \
152                          %s entry: %s: value: %#llx\n"
153 @ MSG_ERR_MALSTR        "%s: %s: malformed string table, initial or final \
154                          byte\n"
155 @ MSG_ERR_MULTTEHFRMHDR "%s: [%d: %s] multiple .eh_frame_hdr sections seen \
156                          (1 expected)\n"
157 @ MSG_ERR_BADEHFRMPTR   "%s: section[%d: %s] FramePtr (%#llx) does not match \
158                          shdr[%d: %s].sh_addr (%#llx)\n"
159 @ MSG_ERR_BADSORT       "%s: %s: index[%d]: invalid sort order\n"
160 @ MSG_ERR_BADSIDYNNDX   "%s: [%d: %s][%d]: dynamic section index out of \
161                          range (0 - %d): %d\n";
162 @ MSG_ERR_BADSIDYNTAG   "%s: [%d: %s][%d]: dynamic element \
163                          [%d: %s][%d] should have type %s: %s\n";
164 @ MSG_ERR_BADCIEFDELEN  "%s: %s: invalid CIE/FDE length: %#llx at %#llx\n"

167 @ MSG_WARN_INVINTERP1   "%s: PT_INTERP header has no associated section\n"
168 @ MSG_WARN_INVINTERP2   "%s: interp section: %s: and PT_INTERP program \
169                          header have conflicting size or offsets\n"
170 @ MSG_WARN_INVCAP1       "%s: PT_SUNWCAP header has no associated section\n"
171 @ MSG_WARN_INVCAP2       "%s: capabilities section[%d]: %s: requires PT_CAP \
172                          program header\n"
173 @ MSG_WARN_INVCAP3       "%s: capabilities section[%d]: %s: and PT_CAP program \
174                          header have conflicting size or offsets\n"
175 @ MSG_WARN_INVCAP4       "%s: capabilities section[%d]: %s: requires string \
176                          table: invalid sh_info: %d\n";
177 @ MSG_WARN_INADDR32SF1   "%s: capabilities section %s: software capability \
178                          ADDR32: is ineffective within a 32-bit object\n"
179 @ MSG_WARN_MULTTEHFRM    "%s: section[%d: %s]: %s object has multiple \
180                          .eh_frame sections\n"

182 @ MSG_INFO_LINUXOSABI   "%s: %s object has Linux .note.ABI-tag section. \
183                          Assuming %s\n"

185 @ MSG_ERR_DWOVRFLW      "%s: %s: encoded DWARF data exceeds section size\n"
186 @ MSG_ERR_DWBADENC      "%s: %s: bad DWARF encoding: %#x\n"
187 @ MSG_ERR_DWNOCIE       "%s: %s: no CIE prior to FDE\n"

189 # exception_range_entry table entries.
190 # TRANSLATION_NOTE - the following entries provide for a series of one or more

```

```

191 # standard 32-bit and 64-bit .exception_ranges table entries that align with
192 # the initial title.

194 @ MSG_EXR_TITLE_32      "      index      offset      ret_addr \
195                          length      handler      type_blk"
196 @ MSG_EXR_ENTRY_32     "%10.10s 0x%8.8llx 0x%8.8llx 0x%8.8llx 0x%8.8llx \
197                          0x%8.8llx"
198 @ MSG_EXR_TITLE_64     "      index      offset      ret_addr \
199                          length      handler      type_blk"
200 @ MSG_EXR_ENTRY_64     "%10.10s 0x%16.16llx 0x%16.16llx 0x%16.16llx \
201                          0x%16.16llx 0x%16.16llx"

203 # Elf Output Messages

205 @ MSG_ELF_SHDR          "Section Header[%d]:  sh_name: %s"
206 @ MSG_ELF_PHDR          "Program Header[%d]: "

208 @ MSG_ELF_SCN_CAP       "Capabilities Section: %s"
209 @ MSG_ELF_SCN_CAPCHAIN  "Capabilities Chain Section: %s"
210 @ MSG_ELF_SCN_INTERP    "Interpreter Section: %s"
211 @ MSG_ELF_SCN_VERDEF    "Version Definition Section: %s"
212 @ MSG_ELF_SCN_VERNEED   "Version Needed Section: %s"
213 @ MSG_ELF_SCN_SYMTAB    "Symbol Table Section: %s"
214 @ MSG_ELF_SCN_RELOC     "Relocation Section: %s"
215 @ MSG_ELF_SCN_UNWIND    "Unwind Section: %s"
216 @ MSG_ELF_SCN_DYNAMIC   "Dynamic Section: %s"
217 @ MSG_ELF_SCN_NOTE      "Note Section: %s"
218 @ MSG_ELF_SCN_HASH      "Hash Section: %s"
219 @ MSG_ELF_SCN_SYMINFO   "Syminfo Section: %s"
220 @ MSG_ELF_SCN_GOT       "Global Offset Table Section: %s"
221 @ MSG_ELF_SCN_GRP       "Group Section: %s"
222 @ MSG_ELF_SCN_MOVE      "Move Section: %s"
223 @ MSG_ELF_SCN_SYMSORT1  "Symbol Sort Section: %s (%s)"
224 @ MSG_ELF_SCN_SYMSORT2  "Symbol Sort Section: %s (%s / %s)"

226 @ MSG_OBJ_CAP_TITLE     " Object Capabilities:"
227 @ MSG_SYM_CAP_TITLE     " Symbol Capabilities:"
228 @ MSG_CAPINFO_ENTRIES   " Symbols:"
229 @ MSG_CAPCHAIN_TITLE    " Capabilities family: %s"
230 @ MSG_CAPCHAIN_ENTRY    " chainndx symndx name"
231 @ MSG_ERR_INVCAP        "%s: capabilities section: %s: contains symbol \
232                          capabilities groups, but no capabilities information \
233                          section is defined: invalid sh_link: %d\n"
234 @ MSG_ERR_INVCAPINFO1   "%s: capabilities information section: %s: no symbol \
235                          table is defined: invalid sh_link: %d\n"
236 @ MSG_ERR_INVCAPINFO2   "%s: capabilities information section: %s: no \
237                          capabilities chain is defined: invalid sh_info: %d\n"
238 @ MSG_ERR_INVCAPINFO3   "%s: capabilities information section: %s: index %d: \
239                          bad capabilities chain index defined: %d\n"
240 @ MSG_ERR_CHBADSYMNDX   "%s: bad symbol reference %d: from capability chain: \
241                          %s entry: %d\n"

243 @ MSG_ELF_HASH_BKTS1    "%10.10s buckets contain %8d symbols"
244 @ MSG_ELF_HASH_BKTS2    "%10.10s buckets %8d symbols (globals)"
245 @ MSG_ELF_HASH_INFO     " bucket symndx name"
246 @ MSG_HASH_OVERFLW      "%s: warning: section %s: too many symbols to count, \
247                          bucket=%d count=%d"
248 @ MSG_ELF_ERR_SHDR      "\tunable to obtain section header: shstrtab[%lld]\n"
249 @ MSG_ELF_ERR_DATA       "\tunable to obtain section data: shstrtab[%lld]\n"
250 @ MSG_ELF_ERR_SCN        "\tunable to obtain section header: section[%d]\n"
251 @ MSG_ELF_ERR_SCNDATA    "\tunable to obtain section data: section[%d]\n"
252 @ MSG_ARCHIVE_SYMTAB_32 "\nSymbol Table: (archive, 32-bit offsets)"
253 @ MSG_ARCHIVE_SYMTAB_64 "\nSymbol Table: (archive, 64-bit offsets)"
254 @ MSG_ARCHIVE_FIELDS_32 "      index      offset      member name and symbol"
255 @ MSG_ARCHIVE_FIELDS_64 "      index      offset      member name and symbol"

```

```

257 @ MSG_GOT_MULTIPLE      "%s: multiple relocations against \
258                          the same GOT entry ndx: %d addr: 0x%llx\n"
259 @ MSG_GOT_UNEXPECTED    "%s: warning: section %s: section unexpected within \
260                          relocatable object\n"

262 # Miscellaneous clutter

264 @ MSG_STR_NULL           "(null)"
265 @ MSG_STR_DEPRECATED    "(deprecated value)"
266 @ MSG_STR_UNKNOWN       "<unknown>"
267 @ MSG_STR_SECTION       "%s (section)"
268 @ MSG_STR_CHECKSUM      "elf checksum: 0x%lx"

270 @ MSG_FMT_SCNNDX        "section[%d]"
271 @ MSG_FMT_NOTEENTNDX   "  entry [%d]";

274 @ MSG_ERR_MALLOC       "%s: malloc: %s\n"
275 @ MSG_ERR_OPEN          "%s: open: %s\n"
276 @ MSG_ERR_READ          "%s: read: %s\n"
277 @ MSG_ERR_WRITE         "%s: write: %s\n"
278 @ MSG_ERR_BAD_T_SHT     "%s: unrecognized section header type: %s\n"
279 @ MSG_ERR_BAD_T_PT      "%s: unrecognized program header type: %s\n"
280 @ MSG_ERR_BAD_T_OSABI   "%s: unrecognized operating system ABI: %s\n"
281 @ MSG_ERR_ambiguous_MATCH "%s: ambiguous use of -I, -N, or -T. Remove \
282                          -p option or section selection option(s)\n"

284 #
285 # SHT_MOVE messages
286 #
287 @ MSG_MOVE_TITLE        "      symndx      offset      size repeat stride      \
288                          value with respect to"
289 @ MSG_MOVE_ENTRY        "%10.10s %10.11x %6d %6d %6d %16.11x %s"

291 #
292 # SHT_GROUP messages
293 #
294 @ MSG_GRP_TITLE         "      index      flags / section      signature symbol"
295 @ MSG_GRP_SIGNATURE     "      [0] %24s %s"
296 @ MSG_GRP_INVALIDSCN   "<invalid section>"

298 #
299 # SHT_NOTE messages
300 #
301 @ MSG_NOTE_BADDATASZ   "%s: %s: note header exceeds section size. \
302                          offset: 0x%x\n"
303 @ MSG_NOTE_BADNMSZ    "%s: %s: note name value exceeds section size. \
304                          offset: 0x%x namesize: 0x%x\n"
305 @ MSG_NOTE_BADDESZ    "%s: %s: note data size exceeds section size. \
306                          offset: 0x%x datasize: 0x%x\n"
307 @ MSG_NOTE_BADCOREARCH "%s: elfdump core file note support not available for \
308                          architecture: %s\n"
309 @ MSG_NOTE_BADCOREDATA "%s: elfdump core file note data truncated or \
310                          otherwise malformed\n"
311 @ MSG_NOTE_BADCORETYPE "%s: unknown note type %#x\n"

313 @ MSG_NOTE_BAD_SECFLAGS_VER "unknown prsecflags_t version: "

315 #endif /* ! codereview */
316 @ _END_

318 # The following strings represent reserved words, files, pathnames and symbols.
319 # Reference to this strings is via the MSG_ORIG() macro, and thus no message
320 # translation is required.

322 @ MSG_STR_OSQBRKT      "[ "

```

```

323 @ MSG_STR_CSQBRKT      "]"

325 @ MSG_GRP_COMDAT       "  COMDAT  "
326 @ MSG_GRP_ENTRY       "%10.10s %s [%lld]\n"
327 @ MSG_GRP_UNKNOWN      "  0x%x  "

329 @ MSG_ELF_GOT          ".got"
330 @ MSG_ELF_INIT         ".init"
331 @ MSG_ELF_FINI         ".fini"
332 @ MSG_ELF_INTERP       ".interp"

334 @ MSG_ELF_GETEHDR     "elf_getehdr"
335 @ MSG_ELF_GETPHDR     "elf_getphdr"
336 @ MSG_ELF_GETSHDR     "elf_getshdr"
337 @ MSG_ELF_GETSCN      "elf_getscn"
338 @ MSG_ELF_GETDATA     "elf_getdata"
339 @ MSG_ELF_GETARHDR    "elf_getarhdr"
340 @ MSG_ELF_GETARSYM    "elf_getarsym"
341 @ MSG_ELF_RAND        "elf_rand"
342 @ MSG_ELF_BEGIN       "elf_begin"
343 @ MSG_ELF_GETPHDRNUM  "elf_getphdrnum"
344 @ MSG_ELF_GETSHDRNUM  "elf_getshdrnum"
345 @ MSG_ELF_GETSHDRSTRNDX "elf_getshdrstrndx"
346 @ MSG_ELF_XLATETOM    "elf_xlatetom"
347 @ MSG_ELF_ARSYM       "ARSYM"

349 @ MSG_SYM_INIT        "_init"
350 @ MSG_SYM_FINI        "_fini"
351 @ MSG_SYM_GOT         "_GLOBAL_OFFSET_TABLE_"

353 @ MSG_STR_OPTIONS      "CddeGgHhiI:klmN:nO:PprSst:uvw:y"

355 @ MSG_STR_8SP         "      "
356 @ MSG_STR_EMPTY       ""
357 @ MSG_STR_CORE        "CORE"
358 @ MSG_STR_NOTEABITAG  ".note.ABI-tag"
359 @ MSG_STR_GNU         "GNU"
360 @ MSG_STR_LOC         "loc"
361 @ MSG_STR_INITLOC     "initloc"

363 @ MSG_FMT_INDENT      "      %s"
364 @ MSG_FMT_INDEX       "[%lld]"
365 @ MSG_FMT_INDEX2      "[%d]"
366 @ MSG_FMT_ASRINDEX    "[% asr%d ]"
367 @ MSG_FMT_INDEXRNG    "[%d-%d]"
368 @ MSG_FMT_INTEGER     "%d"
369 @ MSG_FMT_HASH_INFO   "%10.10s %-10s %s"
370 @ MSG_FMT_CHAIN_INFO  "%10.10s %-10s %s"
371 @ MSG_FMT_ARSYM1_32   "%10.10s 0x%8.8llx (%s):%s"
372 @ MSG_FMT_ARSYM2_32   "%10.10s 0x%8.8llx"
373 @ MSG_FMT_ARSYM1_64   "%10.10s 0x%16.16llx (%s):%s"
374 @ MSG_FMT_ARSYM2_64   "%10.10s 0x%16.16llx"
375 @ MSG_FMT_ARNAME      "%s(%s)"
376 @ MSG_FMT_NLSTR       "\n%s:"
377 @ MSG_FMT_NLSTRNL    "\n%s:\n"
378 @ MSG_FMT_SECSYM      "%.*s%s"

380 @ MSG_HEXDUMP_ROW     "%*s%-*s%s"
381 @ MSG_HEXDUMP_TOK     "%2.2x"

383 @ MSG_SUNW_OST_SGS    "SUNW_OST_SGS"

385 # Unwind info

387 @ MSG_SCN_FRM         ".eh_frame"
388 @ MSG_SCN_FRMHDR     ".eh_frame_hdr"

```

```

389 @ MSG_SCN_EXRANGE      ".exception_ranges"
391 @ MSG_UNW_FRMHDR       "Frame Header:"
392 @ MSG_UNW_FRMVERS      "  Version: %d"
393 @ MSG_UNW_FRPTRENC     "  FramePtrEnc: %-20s  FramePtr: %#llx"
394 @ MSG_UNW_FDCNENC      "  FdcCntEnc:   %-20s  FdcCnt: %lld"
395 @ MSG_UNW_TABENC       "  TableEnc:    %-20s"
396 @ MSG_UNW_BINSRTAB1    "  Binary Search Table:"
397 @ MSG_UNW_BINSRTAB2_32 "    InitialLoc  FdeLoc"
398 @ MSG_UNW_BINSRTAB2_64 "    InitialLoc  FdeLoc"
399 @ MSG_UNW_BINSRTABENT_32 "  0x%08llx  0x%08llx"
400 @ MSG_UNW_BINSRTABENT_64 "  0x%016llx 0x%016llx"
401 @ MSG_UNW_ZEROTERM     "ZERO terminator: [0x00000000]"
402 @ MSG_UNW_CIE          "CIE: [%#llx]"
403 @ MSG_UNW_CIELENGTH   "  length: 0x%02x  cieid: %d"
404 @ MSG_UNW_CIEVERS     "  version: %d  augmentation: '%s'"
405 @ MSG_UNW_CIECALGN    "  codealign: %#llx  dataalign: %lld \
406                        retaddr: %d"
407 @ MSG_UNW_CIEAXVAL     "  Augmentation Data:"
408 @ MSG_UNW_CIEAXSZ     "    size: %lld"
409 @ MSG_UNW_CIEAXPERS   "    personality:"
410 @ MSG_UNW_CIEAXPERSENC "      encoding: 0x%02x %s"
411 @ MSG_UNW_CIEAXPERSRTN "      routine:  %#08llx"
412 @ MSG_UNW_CIEAXCENC   "    code pointer encoding: 0x%02x %s"
413 @ MSG_UNW_CIEAXLSDA  "    lsda encoding: 0x%02x %s"
414 @ MSG_UNW_CIEAXUNEC  "    Unexpected aug val: %c"
415 @ MSG_UNW_CIECFI     "  CallFrameInstructions:"

417 @ MSG_UNW_FDE         "  FDE: [%#llx]"
418 @ MSG_UNW_FDELENGTH  "    length:  %#x  cieptr:  %#x"
419 @ MSG_UNW_FDEINITLOC "    initloc:  %#llx  addrrange:  %#llx  endloc:  %#llx"
420 @ MSG_UNW_FDEAXVAL   "  Augmentation Data:"
421 @ MSG_UNW_FDEAXSIZE  "    size:  %#llx"
422 @ MSG_UNW_FDEAXLSDA "    lsda:  %#llx"
423 @ MSG_UNW_FDECFI     "  CallFrameInstructions:"

425 # Unwind section Call Frame Instructions. These all start with a leading
426 # "%*s%s", used to insert leading white space and the opcode name.

428 @ MSG_CFA_ADV_LOC     "%*s%s: %s + %llu => %#llx"
429 @ MSG_CFA_CFAOFF     "%*s%s: %s, cfa%+lld"
430 @ MSG_CFA_CFASET     "%*s%s: cfa=%#llx"
431 @ MSG_CFA_LLD        "%*s%s: %lld"
432 @ MSG_CFA_LLU        "%*s%s: %llu"
433 @ MSG_CFA_REG        "%*s%s: %s"
434 @ MSG_CFA_REG_OFFLLD "%*s%s: %s, offset=%lld"
435 @ MSG_CFA_REG_OFFLLU "%*s%s: %s, offset=%llu"
436 @ MSG_CFA_REG_REG    "%*s%s: %s, %s"
437 @ MSG_CFA_SIMPLE     "%*s%s"
438 @ MSG_CFA_SIMPLEREP "%*s%s [%d]"
439 @ MSG_CFA_EBLK      "%*s%s: expr(%llu bytes)"
440 @ MSG_CFA_REG_EBLK  "%*s%s: %s, expr(%llu bytes)"

442 # Architecture specific register name formats

444 @ MSG_REG_FMT_BASIC  "r%d"
445 @ MSG_REG_FMT_NAME   "r%d (%s)"

448 # Note messages

450 @ MSG_NOTE_TYPE      "  type:  %#x"
451 @ MSG_NOTE_TYPE_STR  "  type:  %s"
452 @ MSG_NOTE_NAMESZ   "  namesz:  %#x"
453 @ MSG_NOTE_NAME     "  name:    "
454 @ MSG_NOTE_DESCSZ   "  descsz:  %#x"

```

```

456 @ MSG_NOTE_DESC      "  desc:"
457 @ MSG_CNOTE_DESC_ASRSET_T "desc: (asrset_t)"
458 @ MSG_CNOTE_DESC_AUXV_T "desc: (auxv_t)"
459 @ MSG_CNOTE_DESC_CORE_CONTENT_T "desc: (core_content_t)"
460 @ MSG_CNOTE_DESC_LWPSINFO_T "desc: (lwpsinfo_t)"
461 @ MSG_CNOTE_DESC_LWPSTATUS_T "desc: (lwpstatus_t)"
462 @ MSG_CNOTE_DESC_PRCRED_T "desc: (prcred_t)"
463 @ MSG_CNOTE_DESC_PRIV_IMPL_INFO_T "desc: (priv_impl_info_t)"
464 @ MSG_CNOTE_DESC_PRPRIV_T "desc: (prpriv_t)"
465 @ MSG_CNOTE_DESC_PRPSINFO_T "desc: (prpsinfo_t)"
466 @ MSG_CNOTE_DESC_PRSTATUS_T "desc: (prstatus_t)"
467 @ MSG_CNOTE_DESC_PSINFO_T "desc: (psinfo_t)"
468 @ MSG_CNOTE_DESC_PSTATUS_T "desc: (pstatus_t)"
469 @ MSG_CNOTE_DESC_STRUCT_UTSNAME "desc: (struct utsname)"
470 @ MSG_CNOTE_DESC_PRFDINFO_T "desc: (prfdinfo_t)"
471 @ MSG_CNOTE_DESC_PRSECFLAGS_T "desc: (prsecflags_t)"

473 @ MSG_CNOTE_FMT_LINE   "%*s%-*s%s"
474 @ MSG_CNOTE_FMT_LINE_2UP "%*s%-*s%-*s%s"
475 @ MSG_CNOTE_FMT_D      "%d"
476 @ MSG_CNOTE_FMT_LLD    "%lld"
477 @ MSG_CNOTE_FMT_U      "%u"
478 @ MSG_CNOTE_FMT_LLU    "%llu"
479 @ MSG_CNOTE_FMT_X      "%#x"
480 @ MSG_CNOTE_FMT_LLL    "%#llx"
481 @ MSG_CNOTE_FMT_Z2X    "0x%2.2x"
482 @ MSG_CNOTE_FMT_Z4X    "0x%4.4x"
483 @ MSG_CNOTE_FMT_Z8X    "0x%8.8x"
484 @ MSG_CNOTE_FMT_Z16LLX "0x%16.16llx"
485 @ MSG_CNOTE_FMT_TITLE  "%*s%s"
486 @ MSG_CNOTE_FMT_AUXVLINE "%*s%10.10s %-*s %s"
487 @ MSG_CNOTE_FMT_PRTPTCT "%u.%u%"

489 @ MSG_CNOTE_T_PRIV_FLAGS "priv_flags:"
490 @ MSG_CNOTE_T_PRIV_GLOBALINFOSIZE "priv_globalinfosize:"
491 @ MSG_CNOTE_T_PRIV_HEADERSIZE "priv_headersize:"
492 @ MSG_CNOTE_T_PRIV_INFOSIZE "priv_infosize:"
493 @ MSG_CNOTE_T_PRIV_MAX     "priv_max:"
494 @ MSG_CNOTE_T_PRIV_NSETS   "priv_nsets:"
495 @ MSG_CNOTE_T_PRIV_SETSIZE "priv_setsize:"
496 @ MSG_CNOTE_T_PR_ACTION   "pr_action:"
497 @ MSG_CNOTE_T_PR_ADDR     "pr_addr:"
498 @ MSG_CNOTE_T_PR_AGENTID   "pr_agentid:"
499 @ MSG_CNOTE_T_PR_ALTSTACK "pr_altstack:"
500 @ MSG_CNOTE_T_PR_ARGC     "pr_argc:"
501 @ MSG_CNOTE_T_PR_ARGV     "pr_argv:"
502 @ MSG_CNOTE_T_PR_ASLEWPID "pr_aslwpid:"
503 @ MSG_CNOTE_T_PR_BIND     "pr_bind:"
504 @ MSG_CNOTE_T_PR_BINDPRO  "pr_bindpro:"
505 @ MSG_CNOTE_T_PR_BINDPSET "pr_bindpset:"
506 @ MSG_CNOTE_T_PR_BRKBASE  "pr_brkbase:"
507 @ MSG_CNOTE_T_PR_BRKSIZE  "pr_brksize:"
508 @ MSG_CNOTE_T_PR_BYRSSIZE "pr_byrssize:"
509 @ MSG_CNOTE_T_PR_BYSIZE   "pr_bysize:"
510 @ MSG_CNOTE_T_PR_CLNAME   "pr_clname:"
511 @ MSG_CNOTE_T_PR_CONTRACT "pr_contract:"
512 @ MSG_CNOTE_T_PR_CPU      "pr_cpu:"
513 @ MSG_CNOTE_T_PR_CSTIME   "pr_cstime:"
514 @ MSG_CNOTE_T_PR_CTIME    "pr_ctime:"
515 @ MSG_CNOTE_T_PR_CURSIG   "pr_cursig:"
516 @ MSG_CNOTE_T_PR_CUTIME  "pr_cutime:"
517 @ MSG_CNOTE_T_PR_DMODEL   "pr_dmodel:"
518 @ MSG_CNOTE_T_PR_EGID     "pr_egid:"
519 @ MSG_CNOTE_T_PR_ENVP     "pr_envp:"

```

```

520 @ MSG_CNOTE_T_PR_ERRNO      "pr_errno:"
521 @ MSG_CNOTE_T_PR_ERRPRIV    "pr_errpriv:"
522 @ MSG_CNOTE_T_PR_EUID       "pr_euid:"
523 @ MSG_CNOTE_T_PR_FLAG       "pr_flag:"
524 @ MSG_CNOTE_T_PR_FLAGS      "pr_flags:"
525 @ MSG_CNOTE_T_PR_FLTTRACE    "prflttrace:"
526 @ MSG_CNOTE_T_PR_FNAME      "pr_fname:"
527 @ MSG_CNOTE_T_PR_FPREG      "pr_fpreg:"
528 @ MSG_CNOTE_T_PR_GID        "pr_gid:"
529 @ MSG_CNOTE_T_PR_GROUPS     "pr_groups:"
530 @ MSG_CNOTE_T_PR_INFO       "pr_info:"
531 @ MSG_CNOTE_T_PR_INFOSIZE    "pr_infosize:"
532 @ MSG_CNOTE_T_PR_INSTR      "pr_instr:"
533 @ MSG_CNOTE_T_PR_LGRP       "pr_lgrp:"
534 @ MSG_CNOTE_T_PR_LTTYDEV     "pr_lttydev:"
535 @ MSG_CNOTE_T_PR_LWP        "pr_lwp:"
536 @ MSG_CNOTE_T_PR_LWPHOLD     "pr_lwphold:"
537 @ MSG_CNOTE_T_PR_LWPID      "pr_lwpid:"
538 @ MSG_CNOTE_T_PR_LWPPEND     "pr_lwppend:"
539 @ MSG_CNOTE_T_PR_NAME       "pr_name:"
540 @ MSG_CNOTE_T_PR_NGROUPS     "pr_ngroups:"
541 @ MSG_CNOTE_T_PR_NICE       "pr_nice:"
542 @ MSG_CNOTE_T_PR_NLWP       "pr_nlwp:"
543 @ MSG_CNOTE_T_PR_NSETS       "pr_nsets:"
544 @ MSG_CNOTE_T_PR_NSYSARG     "pr_nsysarg:"
545 @ MSG_CNOTE_T_PR_NZOMB       "pr_nzomb:"
546 @ MSG_CNOTE_T_PR_OLDCONTEXT  "pr_oldcontext:"
547 @ MSG_CNOTE_T_PR_OLDPRI     "pr_olddpri:"
548 @ MSG_CNOTE_T_PR_ONPRO       "pr_onpro:"
549 @ MSG_CNOTE_T_PR_OTTYDEV     "pr_ottydev:"
550 @ MSG_CNOTE_T_PR_PCTCPU      "pr_pctcpu:"
551 @ MSG_CNOTE_T_PR_PCTMEM      "pr_pctmem:"
552 @ MSG_CNOTE_T_PR_PGID       "pr_pgid:"
553 @ MSG_CNOTE_T_PR_PGRP       "pr_pgrp:"
554 @ MSG_CNOTE_T_PR_PID        "pr_pid:"
555 @ MSG_CNOTE_T_PR_POOLID      "pr_poolid:"
556 @ MSG_CNOTE_T_PR_PPID       "pr_ppid:"
557 @ MSG_CNOTE_T_PR_PRI        "pr_pri:"
558 @ MSG_CNOTE_T_PR_PROCESSOR   "pr_processor:"
559 @ MSG_CNOTE_T_PR_PROJID     "pr_projid:"
560 @ MSG_CNOTE_T_PR_PSARGS     "pr_psargs:"
561 @ MSG_CNOTE_T_PR_REG        "pr_reg:"
562 @ MSG_CNOTE_T_PR_RGID       "pr_rgid:"
563 @ MSG_CNOTE_T_PR_RSSIZE     "pr_rssize:"
564 @ MSG_CNOTE_T_PR_RUID       "pr_ruid:"
565 @ MSG_CNOTE_T_PR_RVALL1     "pr_rvall1:"
566 @ MSG_CNOTE_T_PR_RVAL2     "pr_rval2:"
567 @ MSG_CNOTE_T_PR_SETS       "pr_sets:"
568 @ MSG_CNOTE_T_PR_SETSIZE    "pr_setsize:"
569 @ MSG_CNOTE_T_PR_SGID       "pr_sgid:"
570 @ MSG_CNOTE_T_PR_SID        "pr_sid:"
571 @ MSG_CNOTE_T_PR_SIGHOLD     "pr_sighold:"
572 @ MSG_CNOTE_T_PR_SIGPEND    "pr_sigpend:"
573 @ MSG_CNOTE_T_PR_SIGTRACE   "pr_sigtrace:"
574 @ MSG_CNOTE_T_PR_SIZE       "pr_size:"
575 @ MSG_CNOTE_T_PR_SNAME      "pr_sname:"
576 @ MSG_CNOTE_T_PR_START      "pr_start:"
577 @ MSG_CNOTE_T_PR_STATE      "pr_state:"
578 @ MSG_CNOTE_T_PR_STIME      "pr_stime:"
579 @ MSG_CNOTE_T_PR_STKBASE    "pr_stkbase:"
580 @ MSG_CNOTE_T_PR_STKSIZE    "pr_stksize:"
581 @ MSG_CNOTE_T_PR_STYPE      "pr_stype:"
582 @ MSG_CNOTE_T_PR_SUID       "pr_suid:"
583 @ MSG_CNOTE_T_PR_SYSARG     "pr_sysarg:"
584 @ MSG_CNOTE_T_PR_SYSCALL    "pr_syscall:"
585 @ MSG_CNOTE_T_PR_SYSENTRY   "pr_sysentry:"

```

```

586 @ MSG_CNOTE_T_PR_SYSEXIT    "pr_sysexit:"
587 @ MSG_CNOTE_T_PR_TASKID     "pr_taskid:"
588 @ MSG_CNOTE_T_PR_TIME       "pr_time:"
589 @ MSG_CNOTE_T_PR_TSTAMP     "pr_tstamp:"
590 @ MSG_CNOTE_T_PR_TTYDEV     "pr_ttydev:"
591 @ MSG_CNOTE_T_PR_UID        "pr_uid:"
592 @ MSG_CNOTE_T_PR_USTACK     "pr_ustack:"
593 @ MSG_CNOTE_T_PR_UTIME      "pr_utime:"
594 @ MSG_CNOTE_T_PR_WCHAN      "pr_wchan:"
595 @ MSG_CNOTE_T_PR_WHAT       "pr_what:"
596 @ MSG_CNOTE_T_PR_WHO       "pr_who:"
597 @ MSG_CNOTE_T_PR_WHY        "pr_why:"
598 @ MSG_CNOTE_T_PR_WSTAT      "pr_wstat:"
599 @ MSG_CNOTE_T_PR_ZOMB       "pr_zomb:"
600 @ MSG_CNOTE_T_PR_ZONEID     "pr_zoneid:"
601 @ MSG_CNOTE_T_PR_EFFECTIVE  "pr_effective:"
602 @ MSG_CNOTE_T_PR_INHERIT    "pr_inherit:"
603 @ MSG_CNOTE_T_PR_LOWER     "pr_lower:"
604 @ MSG_CNOTE_T_PR_UPPER     "pr_upper:"
605 @ MSG_CNOTE_T_PR_VERSION    "pr_version:"
606 #endif /* !codereview */
607 @ MSG_CNOTE_T_SA_FLAGS      "sa_flags:"
608 @ MSG_CNOTE_T_SA_HANDLER    "sa_handler:"
609 @ MSG_CNOTE_T_SA_MASK       "sa_mask:"
610 @ MSG_CNOTE_T_SA_SIGACTION  "sa_sigaction:"
611 @ MSG_CNOTE_T_SIVAL_INT     "sival_int:"
612 @ MSG_CNOTE_T_SIVAL_PTR     "sival_ptr:"
613 @ MSG_CNOTE_T_SI_ADDR      "si_addr:"
614 @ MSG_CNOTE_T_SI_BAND      "si_band:"
615 @ MSG_CNOTE_T_SI_CODE      "si_code:"
616 @ MSG_CNOTE_T_SI_CTIID     "si_ctid:"
617 @ MSG_CNOTE_T_SI_ENTITY     "si_entity:"
618 @ MSG_CNOTE_T_SI_ERRNO     "si_errno:"
619 @ MSG_CNOTE_T_SI_PID       "si_pid:"
620 @ MSG_CNOTE_T_SI_SIGNO     "si_signo:"
621 @ MSG_CNOTE_T_SI_STATUS    "si_status:"
622 @ MSG_CNOTE_T_SI_UID       "si_uid:"
623 @ MSG_CNOTE_T_SI_VALUE     "si_value:"
624 @ MSG_CNOTE_T_SI_ZONEID    "si_zoneid:"
625 @ MSG_CNOTE_T_SS_FLAGS     "ss_flags:"
626 @ MSG_CNOTE_T_SS_SIZE     "ss_size:"
627 @ MSG_CNOTE_T_SS_SP       "ss_sp:"
628 @ MSG_CNOTE_T_TV_NSEC     "tv_nsec:"
629 @ MSG_CNOTE_T_TV_SEC      "tv_sec:"
630 @ MSG_CNOTE_T_UTS_MACHINE  "machine:"
631 @ MSG_CNOTE_T_UTS_NODENAME "nodename:"
632 @ MSG_CNOTE_T_UTS_RELEASE  "release:"
633 @ MSG_CNOTE_T_UTS_SYSNAME  "sysname:"
634 @ MSG_CNOTE_T_UTS_VERSION  "version:"
635 @ MSG_CNOTE_T_PR_FD        "pr_fd:"
636 @ MSG_CNOTE_T_PR_MODE     "pr_mode:"
637 @ MSG_CNOTE_T_PR_PATH     "pr_path:"
638 @ MSG_CNOTE_T_PR_MAJOR    "pr_major:"
639 @ MSG_CNOTE_T_PR_MINOR    "pr_minor:"
640 @ MSG_CNOTE_T_PR_RMAJOR    "pr_rmajor:"
641 @ MSG_CNOTE_T_PR_RMINOR    "pr_rminor:"
642 @ MSG_CNOTE_T_PR_OFFSET   "pr_offset:"
643 @ MSG_CNOTE_T_PR_INO      "pr_ino:"
644 @ MSG_CNOTE_T_PR_FILEFLAGS "pr_fileflags:"
645 @ MSG_CNOTE_T_PR_FD_FLAGS "pr_fdflags:"

648 # Names of fake sections generated from program header data
649 @ MSG_PHDRNAM_CAP           ".SUNW_cap(phdr)"
650 @ MSG_PHDRNAM_CAPINFO      ".SUNW_capinfo(phdr)"
651 @ MSG_PHDRNAM_CAPCHAIN     ".SUNW_capchain(phdr)"

```



```
652 @ MSG_PHDRNAM_DYN          ".dynamic(phdr)"
653 @ MSG_PHDRNAM_DYNSTR       ".dynstr(phdr)"
654 @ MSG_PHDRNAM_DYNSYM       ".dynsym(phdr)"
655 @ MSG_PHDRNAM_FINIARR      ".fini_array(phdr)"
656 @ MSG_PHDRNAM_HASH         ".hash(phdr)"
657 @ MSG_PHDRNAM_INITARR      ".init_array(phdr)"
658 @ MSG_PHDRNAM_INTERP       ".interp(phdr)"
659 @ MSG_PHDRNAM_LDYSYM       ".SUNW_ldynsym(phdr)"
660 @ MSG_PHDRNAM_MOVE         ".move(phdr)"
661 @ MSG_PHDRNAM_NOTE         ".note(phdr)"
662 @ MSG_PHDRNAM_PREINITARR   ".preinit_array(phdr)"
663 @ MSG_PHDRNAM_REL          ".rel(phdr)"
664 @ MSG_PHDRNAM_RELA         ".rela(phdr)"
665 @ MSG_PHDRNAM_SYMINFO      ".syminfo(phdr)"
666 @ MSG_PHDRNAM_SYMSORT      ".SUNW_symsort(phdr)"
667 @ MSG_PHDRNAM_TLSSORT      ".SUNW_tlssort(phdr)"
668 @ MSG_PHDRNAM_UNWIND       ".eh_frame_hdr(phdr)"
669 @ MSG_PHDRNAM_VER          ".SUNW_version(phdr)"
```

new/usr/src/cmd/sgs/elfdump/common/gen_layout_obj.c

1

1406 Thu Jun 30 22:01:18 2016

new/usr/src/cmd/sgs/elfdump/common/gen_layout_obj.c

7029 want per-process exploit mitigation features (secflags)

7030 want basic address space layout randomization (aslr)

7031 noexec_user_stack should be a secflag

7032 want a means to forbid mappings around NULL.

```
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 2015 Nexenta Systems, Inc. All rights reserved.
14 */
```

```
16 /*
17  * A little program who's only purpose is to get all the
18  * CTF type information we want into an object.
19 */
```

```
21 #include <sys/types.h>
22 #include <sys/stat.h>
23 #include <sys/sysmacros.h>
24 #include <sys/corect1.h>
25 #define _STRUCTURED_PROC 1
26 #include <sys/procfs.h>
27 #include <sys/auxv.h>
28 #include <sys/old_procfs.h>
29 #include <sys/utsname.h>
30 #include <sys/secflags.h>
31 #endif /* ! codereview */
```

```
33 /* prgregset_t is a define on intel */
34 #ifndef prgregset_t
35 typedef prgregset_t
36 #undef prgregset_t
37 prgregset_t;
38 #endif
```

```
40 /* instantiate the types for CTF */
41 auxv_t auxv;
42 prgregset_t prgregset;
43 lwpstatus_t lwpstatus;
44 pstatus_t pstatus;
45 prstatus_t prstatus;
46 psinfo_t psinfo;
47 prpsinfo_t prpsinfo;
48 lwpsinfo_t lwpsinfo;
49 prcred_t prcred;
50 prpriv_t prpriv;
51 priv_impl_info_t priv_impl;
52 fltset_t fltset;
53 siginfo_t siginfo;
54 sigset_t sigset;
55 struct sigaction sigact;
56 stack_t stack;
57 sysset_t sysset;
58 timestruc_t ts;
```

new/usr/src/cmd/sgs/elfdump/common/gen_layout_obj.c

2

```
59 struct utsname uts;
60 prfdinfo_t ptfld;
61 prsecflags_t psf;
62 #endif /* ! codereview */
```

```

*****
25074 Thu Jun 30 22:01:19 2016
new/usr/src/cmd/sgs/elfdump/common/gen_struct_layout.c
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
_____unchanged_portion_omitted_____

590 static void
591 gen_prsecflags(void)
592 {
593     START(prsecflags, prsecflags_t);
594     SCALAR_FIELD(prsecflags_t, pr_version, 0);
595     SCALAR_FIELD(prsecflags_t, pr_effective, 0);
596     SCALAR_FIELD(prsecflags_t, pr_inherit, 0);
597     SCALAR_FIELD(prsecflags_t, pr_lower, 0);
598     SCALAR_FIELD(prsecflags_t, pr_upper, 0);
599     END;
600 }
601 #endif /* ! codereview */

603 /*ARGSUSED*/
604 int
605 main(int argc, char *argv[])
606 {
607     const char *fmt = "\t&#x2026;_layout,\n";

609     /* get obj file for input */
610     if (argc < 3) {
611         (void) fprintf(stderr,
612             "usage: %s {object_file} {MACH}\n", argv[0]);
613         exit(1);
614     }

616     objfile = argv[1];
617     machname = argv[2];

619     get_ctf_file(objfile);

621     (void) printf("#include <struct_layout.h>\n");

623     gen_auxv();
624     gen_prgregset();
625     gen_lwpstatus();
626     gen_pstatus();
627     gen_prstatus();
628     gen_psinfo();
629     gen_prpsinfo();
630     gen_lwpsinfo();
631     gen_prcred();
632     gen_prpriv();
633     gen_priv_impl_info();
634     genfltset();
635     gensiginfo();
636     gensigset();
637     gensigaction();
638     genstack();
639     gensysset();
640     gentimestruc();
641     genutsname();
642     genprfdinfo();
643     genprsecflags();

```

```

645     /*
646      * Generate the full arch_layout description
647      */
648     (void) printf(
649         "\n\n\nstatic const sl_arch_layout_t layout_%s = {\n",
650         machname);
651     (void) printf(fmt, "auxv");
652     (void) printf(fmt, "fltset");
653     (void) printf(fmt, "lwpsinfo");
654     (void) printf(fmt, "lwpstatus");
655     (void) printf(fmt, "prcred");
656     (void) printf(fmt, "priv_impl_info");
657     (void) printf(fmt, "prpriv");
658     (void) printf(fmt, "psinfo");
659     (void) printf(fmt, "pstatus");
660     (void) printf(fmt, "prgregset");
661     (void) printf(fmt, "prpsinfo");
662     (void) printf(fmt, "prstatus");
663     (void) printf(fmt, "sigaction");
664     (void) printf(fmt, "siginfo");
665     (void) printf(fmt, "sigset");
666     (void) printf(fmt, "stack");
667     (void) printf(fmt, "sysset");
668     (void) printf(fmt, "timestruc");
669     (void) printf(fmt, "utsname");
670     (void) printf(fmt, "prfdinfo");
671     (void) printf(fmt, "prsecflags");
672 #endif /* ! codereview */
673     (void) printf("};\n");

675     /*
676      * A public function, to make the information available
677      */
678     (void) printf("\n\nconst sl_arch_layout_t *\n");
679     (void) printf("struct_layout_%s(void)\n", machname);
680     (void) printf("{\n\treturn (&layout_%s);\n}\n", machname);

682     return (0);
683 }

685 /*
686  * Helper functions using the CTF library to get type info.
687  */

689 static void
690 get_ctf_file(char *fname)
691 {
692     int ctferr;

694     objfile = fname;
695     if ((ctf = ctf_open(objfile, &ctferr)) == NULL) {
696         errx(1, "Couldn't open object file %s: %s\n", objfile,
697             ctf_errmsg(ctferr));
698     }
699 }

701 static void
702 print_row(int boff, int eltlen, int nelts, int issigned, char *comment)
703 {
704     (void) printf("\t{ %d,\t%d,\t%d,\t%d },\t\t/* %s */\n",
705         boff, eltlen, nelts, issigned, comment);
706 }

708 static void
709 do_start(char *sname, char *tname)
710 {

```

```

711     do_start_name(sname);
712     do_start_sizeof(tname, NULL);
713 }

715 static void
716 do_start_name(char *sname)
717 {
718     (void) printf("\n\nstatic const sl_%s_layout_t %s_layout = {\n",
719                 sname, sname);
720 }

722 static void
723 do_end(void)
724 {
725     (void) printf("};\n");
726 }

728 static void
729 do_start_sizeof(char *tname, char *rtname)
730 {
731     char comment[100];
732     ctf_id_t stype;
733     int sz;

735     if (rtname == NULL)
736         rtname = tname;

738     if ((stype = ctf_lookup_by_name(ctf, rtname)) == CTF_ERR)
739         errx(1, "Couldn't find type %s", rtname);
740     if ((stype = ctf_type_resolve(ctf, stype)) == CTF_ERR)
741         errx(1, "Couldn't resolve type %s", tname);

743     if ((sz = (int)ctf_type_size(ctf, stype)) < 0) {
744         errx(1, "Couldn't get size for type %s", tname);
745     } else if (sz == 0) {
746         errx(1, "Invalid type size 0 for %s", tname);
747     }

749     (void) snprintf(comment, sizeof (comment), "sizeof (%s)", tname);
750     print_row(0, sz, 0, 0, comment);
751 }

753 static void
754 do_scalar_field(char *tname, char *fname, int _sign, char *dotfield)
755 {
756     int rc, off, sz, ftype;

758     rc = get_field_info(tname, fname, dotfield, &off, &ftype);
759     if (rc < 0)
760         errx(1, "Can't get field info for %s->%s", tname, fname);

762     if ((ftype = ctf_type_resolve(ctf, ftype)) == CTF_ERR)
763         errx(1, "Couldn't resolve type of %s->%s", tname, fname);

765     if ((sz = (int)ctf_type_size(ctf, ftype)) < 0) {
766         errx(1, "Couldn't get size for type ID %d", ftype);
767     } else if (sz == 0) {
768         errx(1, "Invalid type size 0 for type ID %d", ftype);
769     }

771     print_row(off, sz, 0, _sign, fname);
772 }

774 static void
775 do_array_field(char *tname, char *fname,
776               int _sign, char *dotfield)

```

```

777 {
778     char comment[100];
779     ctf_arinfo_t ai;
780     int typekind;
781     int esz, rc, off, ftype;

783     rc = get_field_info(tname, fname, dotfield, &off, &ftype);
784     if (rc < 0)
785         errx(1, "Can't get field info for %s->%s", tname, fname);

787     if ((ftype = ctf_type_resolve(ctf, ftype)) == CTF_ERR)
788         errx(1, "Couldn't resolve type of %s->%s", tname, fname);

790     typekind = ctf_type_kind(ctf, ftype);
791     if (typekind != CTF_K_ARRAY)
792         errx(1, "Wrong type for %s->%s", tname, fname);

794     rc = ctf_array_info(ctf, ftype, &ai);
795     if (rc != 0)
796         errx(1, "Can't get array info for %s->%s\n", tname, fname);
797     esz = ctf_type_size(ctf, ai.ctr_contents);
798     if (esz < 0)
799         errx(1, "Can't get element size for %s->%s\n", tname, fname);

801     (void) snprintf(comment, sizeof (comment), "%s[]", fname);
802     print_row(off, esz, ai.ctr_nelems, _sign, comment);
803 }

805 static void
806 do_array_type(char *tname, char *fname, int _sign)
807 {
808     ctf_arinfo_t ai;
809     int stype, typekind;
810     int esz, rc;

812     if ((stype = ctf_lookup_by_name(ctf, tname)) == CTF_ERR)
813         errx(1, "Couldn't find type %s", tname);
814     if ((stype = ctf_type_resolve(ctf, stype)) == CTF_ERR)
815         errx(1, "Couldn't resolve type %s", tname);

817     typekind = ctf_type_kind(ctf, stype);
818     if (typekind != CTF_K_ARRAY)
819         errx(1, "Wrong type for %s->%s", tname, fname);

821     rc = ctf_array_info(ctf, stype, &ai);
822     if (rc != 0)
823         errx(1, "Can't get array info for %s->%s\n", tname, fname);
824     esz = ctf_type_size(ctf, ai.ctr_contents);
825     if (esz < 0)
826         errx(1, "Can't get element size for %s->%s\n", tname, fname);

828     print_row(0, esz, ai.ctr_nelems, _sign, fname);
829 }

832 struct gfinfo {
833     char *tname;      /* top type name, i.e. the struct */
834     char *fname;     /* field name */
835     char *dotname;   /* full field name with dots (optional) */
836     char *prefix;    /* current field search prefix */
837     int base_off;
838     int fld_off;
839     int fld_type;
840 };

842 static int gfi_iter(const char *fname, ctf_id_t mbrtid,

```

```

843     ulong_t off, void *varg);
844
845 /*
846  * Lookup field "fname" in type "tname".  If "dotname" is non-NULL,
847  * that's the full field name with dots, i.e. a un.un.foo, which
848  * we must search for by walking the struct CTF recursively.
849  */
850 static int
851 get_field_info(char *tname, char *fname, char *dotname,
852               int *offp, int *tidp)
853 {
854     struct gfinfo gfi;
855     ctf_id_t stype;
856     int typekind;
857     int rc;
858
859     if ((stype = ctf_lookup_by_name(ctf, tname)) == CTF_ERR)
860         errx(1, "Couldn't find type %s", tname);
861     if ((stype = ctf_type_resolve(ctf, stype)) == CTF_ERR)
862         errx(1, "Couldn't resolve type %s", tname);
863
864     /* If fname has a dot, use it as dotname too. */
865     if (dotname == NULL && strchr(fname, '.') != NULL)
866         dotname = fname;
867
868     gfi.tname = tname;
869     gfi.fname = fname;
870     gfi.dotname = dotname;
871     gfi.prefix = "";
872     gfi.base_off = 0;
873     gfi.fld_off = 0;
874     gfi.fld_type = 0;
875
876     typekind = ctf_type_kind(ctf, stype);
877     switch (typekind) {
878     case CTF_K_STRUCT:
879     case CTF_K_UNION:
880         rc = ctf_member_iter(ctf, stype, gfi_iter, &gfi);
881         break;
882     default:
883         errx(1, "Unexpected top-level type for %s", tname);
884         break;
885     }
886
887     if (rc < 0)
888         errx(1, "Error getting info for %s.%s", stype, fname);
889     if (rc == 0)
890         errx(1, "Did not find %s.%s", tname, fname);
891
892     *offp = gfi.fld_off;
893     *tidp = gfi.fld_type;
894
895     return (0);
896 }
897
898 /*
899  * Iteration callback for ctf_member_iter
900  * Return <0 on error, 0 to keep looking, >0 for found.
901  */
902 *
903 * If no dotname, simple search for fieldname.
904 * If we're asked to search with dotname, we need to do a full
905 * recursive walk of the types under the dotname.
906 */
907 int
908 
```

```

909 gfi_iter(const char *fieldname, ctf_id_t mbrtid, ulong_t off, void *varg)
910 {
911     char namebuf[100];
912     struct gfinfo *gfi = varg;
913     char *saveprefix;
914     int saveoff;
915     int typekind;
916     int byteoff;
917     int len, rc;
918
919     byteoff = gfi->base_off + (int)(off >> 3);
920
921     /* Easy cases first: no dotname */
922     if (gfi->dotname == NULL) {
923         if (strcmp(gfi->fname, fieldname) == 0) {
924             gfi->fld_off = byteoff;
925             gfi->fld_type = mbrtid;
926             return (1);
927         }
928         return (0);
929     }
930
931     /* Exact match on the dotname? */
932     (void) snprintf(namebuf, sizeof(namebuf), "%s%s",
933                   gfi->prefix, fieldname);
934     if (strcmp(gfi->dotname, namebuf) == 0) {
935         gfi->fld_off = byteoff;
936         gfi->fld_type = mbrtid;
937         return (1);
938     }
939
940     /*
941      * May need to recurse under this field, but
942      * only if there's a match through '.'
943      */
944     (void) strlcat(namebuf, ".", sizeof(namebuf));
945     len = strlen(namebuf);
946     if (strncmp(gfi->dotname, namebuf, len) != 0)
947         return (0);
948
949     typekind = ctf_type_kind(ctf, mbrtid);
950     switch (typekind) {
951     case CTF_K_STRUCT:
952     case CTF_K_UNION:
953         break;
954     default:
955         return (0);
956     }
957
958     /* Recursively walk members */
959     saveprefix = gfi->prefix;
960     saveoff = gfi->base_off;
961     gfi->prefix = namebuf;
962     gfi->base_off = byteoff;
963     rc = ctf_member_iter(ctf, mbrtid, gfi_iter, gfi);
964     gfi->prefix = saveprefix;
965     gfi->base_off = saveoff;
966
967     return (rc);
968 }

```

new/usr/src/cmd/sgs/elfdump/common/struct_layout.c

1

```
*****
7287 Thu Jun 30 22:01:20 2016
new/usr/src/cmd/sgs/elfdump/common/struct_layout.c
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */
22 /*
23  * Copyright 2008 Sun Microsystems, Inc. All rights reserved.
24  * Use is subject to license terms.
25 */
26 #pragma ident "%Z%M% %I% %E% SMI"
27 #include <stdlib.h>
28 #include <stdio.h>
29 #include <string.h>
30 #include <msg.h>
31 #include <_elfdump.h>
32 #include <struct_layout.h>
33 #include <conv.h>
36 /*
37  * Functions for extracting and formatting numeric values from
38  * structure data.
39 */
44 /*
45  * Extract the integral field into the value union given and
46  * perform any necessary byte swapping to make the result readable
47  * on the elfdump host.
48 */
49 void
50 sl_extract_num_field(const char *data, int do_swap, const sl_field_t *fdesc,
51 sl_data_t *field_data)
52 {
53     /* Copy the value bytes into our union */
54     (void) memcpy(field_data, data + fdesc->slf_offset,
55 fdesc->slf_eltlen);
57     /* Do byte swapping as necessary */
```

new/usr/src/cmd/sgs/elfdump/common/struct_layout.c

2

```
58     if (do_swap) {
59         switch (fdesc->slf_eltlen) {
60             case 2:
61                 field_data->sld_ui16 = BSWAP_HALF(field_data->sld_ui16);
62                 break;
64             case 4:
65                 field_data->sld_ui32 = BSWAP_WORD(field_data->sld_ui32);
66                 break;
68             case 8:
69                 field_data->sld_ui64 =
70                     BSWAP_LWORD(field_data->sld_ui64);
71                 break;
72             }
73     }
74 }
_____ unchanged_portion_omitted _____
120 /*
121  * Extract the given integer field, and return its value, cast
122  * to Lword. Note that this operation must not be used on values
123  * to Word. Note that this operation must not be used on values
124  * that can be negative, as information can be lost.
125 */
126 sl_extract_as_lword(const char *data, int do_swap, const sl_field_t *fdesc)
127 {
128     sl_data_t    v;
130     /* Extract the value from the raw data */
131     sl_extract_num_field(data, do_swap, fdesc, &v);
133     if (fdesc->slf_sign) {
134         switch (fdesc->slf_eltlen) {
135             case 1:
136                 return ((Lword) v.sld_i8);
137             case 2:
138                 return ((Lword) v.sld_i16);
139             case 4:
140                 return ((Lword) v.sld_i32);
141             case 8:
142                 return ((Lword) v.sld_i64);
143         }
144     } else {
145         switch (fdesc->slf_eltlen) {
146             case 1:
147                 return ((Lword) v.sld_ui8);
148             case 2:
149                 return ((Lword) v.sld_ui16);
150             case 4:
151                 return ((Lword) v.sld_ui32);
152             case 8:
153                 return ((Lword) v.sld_ui64);
154         }
155     }
157     /* This should not be reached */
158     assert(0);
159     return (0);
160 }
_____ unchanged_portion_omitted _____
```

```

*****
15651 Thu Jun 30 22:01:21 2016
new/usr/src/cmd/sgs/elfdump/common/struct_layout.h
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
_____unchanged_portion_omitted_

529 typedef struct {
530     sl_field_t          sizeof_struct;
531     sl_field_t          pr_version;
532     sl_field_t          pr_effective;
533     sl_field_t          pr_inherit;
534     sl_field_t          pr_lower;
535     sl_field_t          pr_upper;
536 } sl_prsecflags_layout_t;

538 #endif /* ! codereview */
539 /*
540  * This type collects all of the layout definitions for
541  * a given architecture.
542  */
543 typedef struct {
544     const sl_auxv_layout_t      *auxv;          /* auxv_t */
545     const slfltset_layout_t     *fltset;        /* fltset_t */
546     const sllwpsinfo_layout_t   *lwpsinfo;     /* lwpsinfo_t */
547     const sl_lwpstatus_layout_t *lwpstatus;    /* lwpstatus_t */
548     const sl_prcred_layout_t    *prcred;       /* prcred_t */
549     const sl_priv_impl_info_layout_t *priv_impl_info; /* priv_impl_info_t */
550     const sl_prpriv_layout_t    *prpriv;       /* prpriv_t */
551     const sl_psinfo_layout_t    *psinfo;      /* psinfo_t */
552     const sl_pstatus_layout_t   *pstatus;     /* pstatus_t */
553     const sl_prgregset_layout_t *prgregset;   /* prgregset_t */
554     const sl_prpsinfo_layout_t  *prpsinfo;    /* prpsinfo_t */
555     const sl_prstatus_layout_t  *prstatus;    /* prstatus_t */
556     const sl_sigaction_layout_t *sigaction;   /* struct sigaction */
557     const sl_siginfo_layout_t   *siginfo;     /* siginfo_t */
558     const sl_sigset_layout_t    *sigset;      /* sigset_t */
559     const sl_stack_layout_t     *stack;       /* stack_t */
560     const sl_sysset_layout_t    *sysset;      /* sysset_t */
561     const sl_timestruc_layout_t *timestruc;   /* timestruc_t */
562     const sl_utsname_layout_t   *utsname;     /* struct utsname */
563     const sl_prfdinfo_layout_t  *prfdinfo;   /* prfdinfo_t */
564     const sl_prsecflags_layout_t *prsecflags; /* prsecflags_t */
565 #endif /* ! codereview */
566 } sl_arch_layout_t;

570 extern void          sl_extract_num_field(const char *data, int do_swap,
571     const sl_field_t *fdesc, sl_data_t *field_data);
572 extern Word          sl_extract_as_word(const char *data, int do_swap,
573     const sl_field_t *fdesc);
574 extern Lword         sl_extract_as_lword(const char *data, int do_swap,
575     const sl_field_t *fdesc);
576 extern Sword         sl_extract_as_sword(const char *data, int do_swap,
577     const sl_field_t *fdesc);
578 extern const char    *sl_fmt_num(const char *data, int do_swap,
579     const sl_field_t *fdesc, sl_fmt_num_t fmt_type,
580     sl_fmtbuf_t buf);

583 extern const sl_arch_layout_t *sl_mach(Half);
584 extern const sl_arch_layout_t *struct_layout_i386(void);

```

```

585 extern const sl_arch_layout_t *struct_layout_amd64(void);
586 extern const sl_arch_layout_t *struct_layout_sparc(void);
587 extern const sl_arch_layout_t *struct_layout_sparcv9(void);

591 #ifdef __cplusplus
592 }
593 #endif

595 #endif /* _STRUCT_LAYOUT_H */

```

new/usr/src/cmd/sgs/elfdump/common/struct_layout_amd64.c

1

```
*****
12547 Thu Jun 30 22:01:21 2016
new/usr/src/cmd/sgs/elfdump/common/struct_layout_amd64.c
Code review comments from jeffpc
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
_____unchanged_portion_omitted_____
```

```
380 static const sl_prsecflags_layout_t prsecflags_layout = {
381     { 0, 40, 0, 0 }, /* sizeof (prsecflags_t) */
382     { 0, 4, 0, 0 }, /* pr_version */
383     { 8, 8, 0, 0 }, /* pr_effective */
384     { 16, 8, 0, 0 }, /* pr_inherit */
385     { 24, 8, 0, 0 }, /* pr_lower */
386     { 32, 8, 0, 0 }, /* pr_upper */
387 };
```

```
390 #endif /* ! codereview */
```

```
393 static const sl_arch_layout_t layout_amd64 = {
394     &auxv_layout,
395     &fltset_layout,
396     &lwpsinfo_layout,
397     &lwpsstatus_layout,
398     &prcred_layout,
399     &priv_impl_info_layout,
400     &prpriv_layout,
401     &psinfo_layout,
402     &pstatus_layout,
403     &prgregset_layout,
404     &prpsinfo_layout,
405     &prstatus_layout,
406     &sigaction_layout,
407     &siginfo_layout,
408     &sigset_layout,
409     &stack_layout,
410     &sysset_layout,
411     &timestruc_layout,
412     &utsname_layout,
413     &prfdinfo_layout,
414     &prsecflags_layout,
415 #endif /* ! codereview */
416 };
```

```
419 const sl_arch_layout_t *
420 struct_layout_amd64(void)
421 {
422     return (&layout_amd64);
423 }
```


new/usr/src/cmd/sgs/elfdump/common/struct_layout_i386.c

1

```
*****
12501 Thu Jun 30 22:01:22 2016
new/usr/src/cmd/sgs/elfdump/common/struct_layout_i386.c
Code review comments from jeffpc
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
_____unchanged_portion_omitted_____
```

```
380 static const sl_prsecflags_layout_t prsecflags_layout = {
381     { 0, 40, 0, 0 }, /* sizeof (prsecflags_t) */
382     { 0, 4, 0, 0 }, /* pr_version */
383     { 8, 8, 0, 0 }, /* pr_effective */
384     { 16, 8, 0, 0 }, /* pr_inherit */
385     { 24, 8, 0, 0 }, /* pr_lower */
386     { 32, 8, 0, 0 }, /* pr_upper */
387 };
```

```
390 #endif /* ! codereview */
```

```
393 static const sl_arch_layout_t layout_i386 = {
394     &auxv_layout,
395     &fltset_layout,
396     &lwpsinfo_layout,
397     &lwpsstatus_layout,
398     &prcred_layout,
399     &priv_impl_info_layout,
400     &prpriv_layout,
401     &psinfo_layout,
402     &pstatus_layout,
403     &prgregset_layout,
404     &prpsinfo_layout,
405     &prststatus_layout,
406     &sigaction_layout,
407     &siginfo_layout,
408     &sigset_layout,
409     &stack_layout,
410     &sysset_layout,
411     &timestruc_layout,
412     &utsname_layout,
413     &prfdinfo_layout,
414     &prsecflags_layout,
415 #endif /* ! codereview */
416 };
```

```
419 const sl_arch_layout_t *
420 struct_layout_i386(void)
421 {
422     return (&layout_i386);
423 }
```

```
*****
12676 Thu Jun 30 22:01:23 2016
new/usr/src/cmd/sgs/elfdump/common/struct_layout_sparc.c
Code review comments from pmooney (sundry), and igork (screwups in zonecfg refac
*****
_____unchanged_portion_omitted_____
```

```
379 static const sl_prsecflags_layout_t prsecflags_layout = {
380     { 0, 20, 0, 0 }, /* sizeof (prsecflags_t) */
381     { 0, 4, 0, 0 }, /* pr_version */
382     { 4, 4, 0, 0 }, /* pr_effective */
383     { 8, 4, 0, 0 }, /* pr_inherit */
384     { 12, 4, 0, 0 }, /* pr_lower */
385     { 16, 4, 0, 0 }, /* pr_upper */
386 };
```

```
389 #endif /* ! codereview */
```

```
392 static const sl_arch_layout_t layout_sparc = {
393     &auxv_layout,
394     &fltset_layout,
395     &lwpsinfo_layout,
396     &lwpsstatus_layout,
397     &prcred_layout,
398     &priv_impl_info_layout,
399     &prpriv_layout,
400     &psinfo_layout,
401     &pstatus_layout,
402     &prgregset_layout,
403     &prpsinfo_layout,
404     &prstatus_layout,
405     &sigaction_layout,
406     &siginfo_layout,
407     &sigset_layout,
408     &stack_layout,
409     &sysset_layout,
410     &timestruc_layout,
411     &utsname_layout,
412     &prfdinfo_layout,
413     &prsecflags_layout,
414 #endif /* ! codereview */
415 };
```

```
418 const sl_arch_layout_t *
419 struct_layout_sparc(void)
420 {
421     return (&layout_sparc);
422 }
```

```
*****  
12722 Thu Jun 30 22:01:24 2016  
new/usr/src/cmd/sgs/elfdump/common/struct_layout_sparcv9.c  
Code review comments from pmoooney (sundry), and igork (screwups in zonecfg refac  
*****  
_____unchanged_portion_omitted_____
```

```
380 static const sl_prsecflags_layout_t prsecflags_layout = {  
381     { 0, 20, 0, 0 }, /* sizeof (prsecflags_t) */  
382     { 0, 4, 0, 0 }, /* pr_version */  
383     { 4, 4, 0, 0 }, /* pr_effective */  
384     { 8, 4, 0, 0 }, /* pr_inherit */  
385     { 12, 4, 0, 0 }, /* pr_lower */  
386     { 16, 4, 0, 0 }, /* pr_upper */  
387 };
```

```
390 #endif /* ! codereview */
```

```
393 static const sl_arch_layout_t layout_sparcv9 = {  
394     &auxv_layout,  
395     &fltset_layout,  
396     &lwpsinfo_layout,  
397     &lwpstatus_layout,  
398     &prcred_layout,  
399     &priv_impl_info_layout,  
400     &prpriv_layout,  
401     &psinfo_layout,  
402     &pstatus_layout,  
403     &prgregset_layout,  
404     &prpsinfo_layout,  
405     &prstatus_layout,  
406     &sigaction_layout,  
407     &siginfo_layout,  
408     &sigset_layout,  
409     &stack_layout,  
410     &sysset_layout,  
411     &timestruc_layout,  
412     &utsname_layout,  
413     &prfdinfo_layout,  
414     &prsecflags_layout,  
415 #endif /* ! codereview */  
416 };  
417 const sl_arch_layout_t *
```

```
382 const sl_arch_layout_t *  
418 struct_layout_sparcv9(void)  
419 {  
420     return (&layout_sparcv9);  
421 }
```

```
_____unchanged_portion_omitted_____
```

new/usr/src/cmd/sgs/include/conv.h

1

```
*****
41201 Thu Jun 30 22:01:25 2016
new/usr/src/cmd/sgs/include/conv.h
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */
22 /*
23  * Copyright (c) 1988 AT&T
24  * All Rights Reserved
25  *
26  * Copyright (c) 1992, 2010, Oracle and/or its affiliates. All rights reserved.
27  * Copyright 2012 DEY Storage Systems, Inc. All rights reserved.
28  */
30 #ifndef _CONV_H
31 #define _CONV_H
33 /*
34  * Global include file for conversion library.
35  */
37 #include <stdlib.h>
38 #include <libelf.h>
39 #include <dlfcn.h>
40 #include <libld.h>
41 #include <sgs.h>
42 #include <sgsmsg.h>
44 #include <sys/secflags.h>
46 #endif /* ! codereview */
47 #ifdef __cplusplus
48 extern "C" {
49 #endif
51 /*
52  * Configuration features available - maintained here (instead of debug.h)
53  * to save libconv from having to include debug.h which results in numerous
54  * "declared but not used or defined" lint errors.
55  */
56 #define CONF_EDLIBPATH 0x000100 /* ELF default library path */
57 #define CONF_ESLIBPATH 0x000200 /* ELF secure library path */
58 #define CONF_ADLIBPATH 0x000400 /* AOUT default library path */
```

new/usr/src/cmd/sgs/include/conv.h

2

```
59 #define CONF_ASLIBPATH 0x000800 /* AOUT secure library path */
60 #define CONF_DIRCFG 0x001000 /* directory configuration available */
61 #define CONF_OBJALT 0x002000 /* object alternatives available */
62 #define CONF_MEMRESV 0x004000 /* memory reservation required */
63 #define CONF_ENVS 0x008000 /* environment variables available */
64 #define CONF_FLTR 0x010000 /* filter information available */
65 #define CONF_FEATMSK 0xffff00
68 /*
69  * Valid flags for conv_strproc_extract_value().
70  */
71 #define CONV_SPEXV_F_NOTRIM 0x0001 /* Do not trim whitespace around '=' */
72 #define CONV_SPEXV_F_UCASE 0x0002 /* Convert value to uppercase */
73 #define CONV_SPEXV_F_NULLOK 0x0004 /* Empty ("") value is OK */
75 /*
76  * Buffer types:
77  *
78  * Many of the routines in this module require the user to supply a
79  * buffer into which the desired strings may be written. These are
80  * all arrays of characters, and might be defined as simple arrays
81  * of char. The problem with that approach is that when such an array
82  * is passed to a function, the C language considers it to have the
83  * type (char *), without any regard to its length. Not all of our
84  * buffers have the same length, and we want to ensure that the compiler
85  * will refuse to compile code that passes the wrong type of buffer to
86  * a given routine. The solution is to define the buffers as unions
87  * that contain the needed array, and then to pass the given union
88  * by address. The compiler will catch attempts to pass the wrong type
89  * of pointer, and the size of a structure/union is implicit in its type.
90  *
91  * A nice side effect of this approach is that we can use a union with
92  * multiple buffers to handle the cases where a given routine needs
93  * more than one type of buffer. The end result is a single buffer large
94  * enough to handle any of the subcases, but no larger.
95  */
97 /*
98  * Size of buffer used by conv_invalid_val():
99  *
100 * Various values that can't be matched to a symbolic definition are converted
101 * to a numeric string.
102 *
103 * The buffer size reflects the maximum number of digits needed to
104 * display an integer as text, plus a trailing null, and with room for
105 * a leading "0x" if hexadecimal display is selected.
106 *
107 * The 32-bit version of this requires 12 characters, and the 64-bit version
108 * needs 22. By using the larger value for both, we can have a single
109 * definition, which is necessary for code that is ELFCLASS independent. A
110 * nice side benefit is that it lets us dispense with a large number of 32/64
111 * buffer size definitions that build off CONV_INV_BUFSIZE, and the macros
112 * that would then be needed.
113  */
114 #define CONV_INV_BUFSIZE 22
115 typedef union {
116     char buf[CONV_INV_BUFSIZE];
117 } Conv_inv_buf_t;
119 /* conv_ehdr_flags() */
120 #define CONV_EHDR_FLAGS_BUFSIZE 91
121 typedef union {
122     Conv_inv_buf_t inv_buf;
123     char buf[CONV_EHDR_FLAGS_BUFSIZE];
124 } Conv_ehdr_flags_buf_t;
```

```

126 /* conv_reject_desc() */
127 typedef union {
128     Conv_inv_buf_t          inv_buf;
129     Conv_ehdr_flags_buf_t  flags_buf;
130 } Conv_reject_desc_buf_t;

132 /*
133  * conv_la_bind()
134  */
135 #define CONV_LA_BIND_BUFSIZE      56
136 typedef union {
137     Conv_inv_buf_t          inv_buf;
138     char                    buf[CONV_LA_BIND_BUFSIZE];
139 } Conv_la_bind_buf_t;

141 /*
142  * conv_la_search()
143  */
144 #define CONV_LA_SEARCH_BUFSIZE    111
145 typedef union {
146     Conv_inv_buf_t          inv_buf;
147     char                    buf[CONV_LA_SEARCH_BUFSIZE];
148 } Conv_la_search_buf_t;

150 /*
151  * conv_la_symbind()
152  */
153 #define CONV_LA_SYMBIND_BUFSIZE   113
154 typedef union {
155     Conv_inv_buf_t          inv_buf;
156     char                    buf[CONV_LA_SYMBIND_BUFSIZE];
157 } Conv_la_symbind_buf_t;

159 /*
160  * conv_cap_val_hw/sf()
161  */
162 * These sizes are based on the maximum number of capabilities that exist.
163 * See common/elfcap.
164 */
165 #define CONV_CAP_VAL_HW1_BUFSIZE  195
166 typedef union {
167     Conv_inv_buf_t          inv_buf;
168     char                    buf[CONV_CAP_VAL_HW1_BUFSIZE];
169 } Conv_cap_val_hw1_buf_t;

171 #define CONV_CAP_VAL_HW2_BUFSIZE  CONV_INV_BUFSIZE      /* for now */
172 typedef union {
173     Conv_inv_buf_t          inv_buf;
174     char                    buf[CONV_CAP_VAL_HW2_BUFSIZE];
175 } Conv_cap_val_hw2_buf_t;

177 #define CONV_CAP_VAL_SF1_BUFSIZE  45
178 typedef union {
179     Conv_inv_buf_t          inv_buf;
180     char                    buf[CONV_CAP_VAL_SF1_BUFSIZE];
181 } Conv_cap_val_sf1_buf_t;

183 /* conv_cap_val_buf() */
184 typedef union {
185     Conv_inv_buf_t          inv_buf;
186     Conv_cap_val_hw1_buf_t  cap_val_hw1_buf;
187     Conv_cap_val_sf1_buf_t  cap_val_sf1_buf;
188     Conv_cap_val_hw2_buf_t  cap_val_hw2_buf;
189 } Conv_cap_val_buf_t;

```

```

191 /* conv_config_feat() */
192 #define CONV_CONFIG_FEAT_BUFSIZE  204
193 typedef union {
194     Conv_inv_buf_t          inv_buf;
195     char                    buf[CONV_CONFIG_FEAT_BUFSIZE];
196 } Conv_config_feat_buf_t;

198 /* conv_config_obj() */
199 #define CONV_CONFIG_OBJ_BUFSIZE    164
200 typedef union {
201     Conv_inv_buf_t          inv_buf;
202     char                    buf[CONV_CONFIG_OBJ_BUFSIZE];
203 } Conv_config_obj_buf_t;

205 /* conv_dl_mode() */
206 #define CONV_DL_MODE_BUFSIZE       132
207 typedef union {
208     Conv_inv_buf_t          inv_buf;
209     char                    buf[CONV_DL_MODE_BUFSIZE];
210 } Conv_dl_mode_buf_t;

212 /* conv_dl_flag() */
213 #define CONV_DL_FLAG_BUFSIZE       185
214 typedef union {
215     Conv_inv_buf_t          inv_buf;
216     char                    buf[CONV_DL_FLAG_BUFSIZE];
217 } Conv_dl_flag_buf_t;

219 /* conv_grphdl_flags() */
220 #define CONV_GRPDDL_FLAGS_BUFSIZE  78
221 typedef union {
222     Conv_inv_buf_t          inv_buf;
223     char                    buf[CONV_GRPDDL_FLAGS_BUFSIZE];
224 } Conv_grphdl_flags_buf_t;

226 /* conv_grpdesc_flags() */
227 #define CONV_GRPDESC_FLAGS_BUFSIZE 91
228 typedef union {
229     Conv_inv_buf_t          inv_buf;
230     char                    buf[CONV_GRPDESC_FLAGS_BUFSIZE];
231 } Conv_grpdesc_flags_buf_t;

233 /* conv_seg_flags() */
234 #define CONV_SEG_FLAGS_BUFSIZE     241
235 typedef union {
236     Conv_inv_buf_t          inv_buf;
237     char                    buf[CONV_SEG_FLAGS_BUFSIZE];
238 } Conv_seg_flags_buf_t;

240 /* conv_dyn_posflag1() */
241 #define CONV_DYN_POSFLAG1_BUFSIZE  72
242 typedef union {
243     Conv_inv_buf_t          inv_buf;
244     char                    buf[CONV_DYN_POSFLAG1_BUFSIZE];
245 } Conv_dyn_posflag1_buf_t;

247 /* conv_dyn_flag() */
248 #define CONV_DYN_FLAG_BUFSIZE      85
249 typedef union {
250     Conv_inv_buf_t          inv_buf;
251     char                    buf[CONV_DYN_FLAG_BUFSIZE];
252 } Conv_dyn_flag_buf_t;

254 /* conv_dyn_flag1() */
255 #define CONV_DYN_FLAG1_BUFSIZE     361
256 typedef union {

```

```

257     Conv_inv_buf_t      inv_buf;
258     char                 buf[CONV_DYN_FLAG1_BUFSIZE];
259 } Conv_dyn_flag1_buf_t;

261 /* conv_dyn_feature1() */
262 #define CONV_DYN_FEATURE1_BUFSIZE 54
263 typedef union {
264     Conv_inv_buf_t      inv_buf;
265     char                 buf[CONV_DYN_FEATURE1_BUFSIZE];
266 } Conv_dyn_feature1_buf_t;

268 /* conv_bnd_type() */
269 #define CONV_BND_TYPE_BUFSIZE 51
270 typedef union {
271     Conv_inv_buf_t      inv_buf;
272     char                 buf[CONV_BND_TYPE_BUFSIZE];
273 } Conv_bnd_type_buf_t;

275 /* conv_bnd_obj() */
276 #define CONV_BND_OBJ_BUFSIZE 60
277 typedef union {
278     Conv_inv_buf_t      inv_buf;
279     char                 buf[CONV_BND_OBJ_BUFSIZE];
280 } Conv_bnd_obj_buf_t;

282 /* conv_phdr_flags() */
283 #define CONV_PHDR_FLAGS_BUFSIZE 88
284 typedef union {
285     Conv_inv_buf_t      inv_buf;
286     char                 buf[CONV_PHDR_FLAGS_BUFSIZE];
287 } Conv_phdr_flags_buf_t;

289 /* conv_sec_flags() */
290 #define CONV_SEC_FLAGS_BUFSIZE 190
291 typedef union {
292     Conv_inv_buf_t      inv_buf;
293     char                 buf[CONV_SEC_FLAGS_BUFSIZE];
294 } Conv_sec_flags_buf_t;

296 /* conv_dwarf_ehe() */
297 #define CONV_DWARF_EHE_BUFSIZE 43
298 typedef union {
299     Conv_inv_buf_t      inv_buf;
300     char                 buf[CONV_DWARF_EHE_BUFSIZE];
301 } Conv_dwarf_ehe_buf_t;

303 /* conv_syminfo_flags() */
304 #define CONV_SYMINFO_FLAGS_BUFSIZE 230
305 typedef union {
306     Conv_inv_buf_t      inv_buf;
307     char                 buf[CONV_SYMINFO_FLAGS_BUFSIZE];
308 } Conv_syminfo_flags_buf_t;

310 /* conv_cnote_pr_flags() */
311 #define CONV_CNOTE_PR_FLAGS_BUFSIZE 254
312 typedef union {
313     Conv_inv_buf_t      inv_buf;
314     char                 buf[CONV_CNOTE_PR_FLAGS_BUFSIZE];
315 } Conv_cnote_pr_flags_buf_t;

317 /* conv_cnote_old_pr_flags() */
318 #define CONV_CNOTE_OLD_PR_FLAGS_BUFSIZE 174
319 typedef union {
320     Conv_inv_buf_t      inv_buf;
321     char                 buf[CONV_CNOTE_OLD_PR_FLAGS_BUFSIZE];
322 } Conv_cnote_old_pr_flags_buf_t;

```

```

324 /* conv_cnote_proc_flag() */
325 #define CONV_CNOTE_PROC_FLAG_BUFSIZE 39
326 typedef union {
327     Conv_inv_buf_t      inv_buf;
328     char                 buf[CONV_CNOTE_PROC_FLAG_BUFSIZE];
329 } Conv_cnote_proc_flag_buf_t;

331 /* conv_prsecflags() */
332 #define CONV_PRSECFLAGS_BUFSIZE 57
333 typedef union {
334     Conv_inv_buf_t      inv_buf;
335     char                 buf[CONV_PRSECFLAGS_BUFSIZE];
336 } Conv_secflags_buf_t;
337 #endif /* ! codereview */

339 /* conv_cnote_sigset() */
340 #define CONV_CNOTE_SIGSET_BUFSIZE 639
341 typedef union {
342     Conv_inv_buf_t      inv_buf;
343     char                 buf[CONV_CNOTE_SIGSET_BUFSIZE];
344 } Conv_cnote_sigset_buf_t;

346 /* conv_cnotefltset() */
347 #define CONV_CNOTE_FLTSET_BUFSIZE 511
348 typedef union {
349     Conv_inv_buf_t      inv_buf;
350     char                 buf[CONV_CNOTE_FLTSET_BUFSIZE];
351 } Conv_cnotefltset_buf_t;

353 /* conv_cnote_sysset() */
354 #define CONV_CNOTE_SYSET_BUFSIZE 3195
355 typedef union {
356     Conv_inv_buf_t      inv_buf;
357     char                 buf[CONV_CNOTE_SYSET_BUFSIZE];
358 } Conv_cnote_sysset_buf_t;

360 /* conv_cnote_sa_flags() */
361 #define CONV_CNOTE_SA_FLAGS_BUFSIZE 109
362 typedef union {
363     Conv_inv_buf_t      inv_buf;
364     char                 buf[CONV_CNOTE_SA_FLAGS_BUFSIZE];
365 } Conv_cnote_sa_flags_buf_t;

367 /* conv_cnote_ss_flags() */
368 #define CONV_CNOTE_SS_FLAGS_BUFSIZE 48
369 typedef union {
370     Conv_inv_buf_t      inv_buf;
371     char                 buf[CONV_CNOTE_SS_FLAGS_BUFSIZE];
372 } Conv_cnote_ss_flags_buf_t;

374 /* conv_cnote_cc_content() */
375 #define CONV_CNOTE_CC_CONTENT_BUFSIZE 97
376 typedef union {
377     Conv_inv_buf_t      inv_buf;
378     char                 buf[CONV_CNOTE_CC_CONTENT_BUFSIZE];
379 } Conv_cnote_cc_content_buf_t;

381 /* conv_cnote_auxv_af() */
382 #define CONV_CNOTE_AUXV_AF_BUFSIZE 73
383 typedef union {
384     Conv_inv_buf_t      inv_buf;
385     char                 buf[CONV_CNOTE_AUXV_AF_BUFSIZE];
386 } Conv_cnote_auxv_af_buf_t;

388 /* conv_ver_flags() */

```

```

389 #define CONV_VER_FLAGS_BUFSIZE 41
390 typedef union {
391     Conv_inv_buf_t inv_buf;
392     char buf[CONV_VER_FLAGS_BUFSIZE];
393 } Conv_ver_flags_buf_t;

395 /* conv_ent_flags() */
396 #define CONV_ENT_FLAGS_BUFSIZE 69
397 typedef union {
398     Conv_inv_buf_t inv_buf;
399     char buf[CONV_ENT_FLAGS_BUFSIZE];
400 } Conv_ent_flags_buf_t;

402 /* conv_ent_files_flags() */
403 #define CONV_ENT_FILES_FLAGS_BUFSIZE 89
404 typedef union {
405     Conv_inv_buf_t inv_buf;
406     char buf[CONV_ENT_FILES_FLAGS_BUFSIZE];
407 } Conv_ent_files_flags_buf_t;

409 /*
410 * conv_time()
411 *
412 * This size is based on the maximum "hour.min.sec.fraction: " time that
413 * would be expected of ld().
414 */
415 #define CONV_TIME_BUFSIZE 18
416 typedef union {
417     char buf[CONV_TIME_BUFSIZE];
418 } Conv_time_buf_t;

420 /*
421 * Many conversion routines accept a fmt_flags argument of this type
422 * to allow the caller to modify the output. There are two parts to
423 * this value:
424 *
425 * (1) Format requests (decimal vs hex, etc...)
426 * (2) The low order bits specified by CONV_MASK_FMT_ALT
427 * and retrieved by CONV_TYPE_FMT_ALT are integer
428 * values that specify that an alternate set of
429 * strings should be used.
430 *
431 * The fmt_flags value is designed such that a caller can always
432 * supply a 0 in order to receive default behavior.
433 */
434 typedef int Conv_fmt_flags_t;

436 /*
437 * Type used to represent ELF constants within libconv. This relies on
438 * the fact that there are no ELF constants that need more than 32-bits,
439 * nor are there any signed values.
440 */
441 typedef uint32_t Conv_elfvalue_t;

443 /*
444 * Most conversion routines are able to provide strings in one of
445 * several alternative styles. The bottom 8 bits of Conv_fmt_flags_t
446 * are used to specify which strings should be used for a given call
447 * to a conversion routine:
448 *
449 * DEFAULT
450 * The default string style used by a given conversion routine is
451 * an independent choice made by that routine. Different routines
452 * make different choices, based largely on historical usage and
453 * the perceived common case. It may be an alias for one of the
454 * specific styles listed below, or it may be unique.

```

```

455 *
456 * DUMP
457 * Style of strings used by dump(1).
458 *
459 * FILE
460 * Style of strings used by file(1).
461 *
462 * CRLE
463 * Style of strings used by crle(1).
464 *
465 * CF
466 * Canonical Form: The string is exactly the same as the name
467 * of the #define macro that defines it in the public header files.
468 * (e.g. STB_LOCAL, not LOCL, LOCAL, LOC, or any other variation).
469 *
470 * CFNP
471 * No Prefix Canonical Form: The same strings supplied by CF,
472 * but without their standard prefix. (e.g. LOCAL, instead of STT_LOCAL).
473 *
474 * NF
475 * Natural Form: The form of the strings that might typically be entered
476 * via a keyboard by an interactive user. These are usually the strings
477 * from CFNP, converted to lowercase, although in some cases they may
478 * take some other "natural" form. In command completion applications,
479 * lowercase strings appear less formal, and are easier on the eye.
480 *
481 * Every routine is required to have a default style. The others are optional,
482 * and may not be provided if not needed. If a given conversion routine does
483 * not support alternative strings for a given CONV_FMT_ALT type, it silently
484 * ignores the request and supplies the default set. This means that a utility
485 * like dump(1) is free to specify a style like DUMP to every conversion
486 * routine. It will receive its special strings if there are any, and
487 * the defaults otherwise.
488 */
489 #define CONV_MASK_FMT_ALT 0xff
490 #define CONV_TYPE_FMT_ALT(fmt_flags) (fmt_flags & CONV_MASK_FMT_ALT)

492 #define CONV_FMT_ALT_DEFAULT 0 /* "Standard" strings */
493 #define CONV_FMT_ALT_DUMP 1 /* dump(1) */
494 #define CONV_FMT_ALT_FILE 2 /* file(1) */
495 #define CONV_FMT_ALT_CRLE 3 /* crle(1) */
496 #define CONV_FMT_ALT_CF 4 /* Canonical Form */
497 #define CONV_FMT_ALT_CFNP 5 /* No Prefix Canonical Form */
498 #define CONV_FMT_ALT_NF 6 /* Natural Form */

500 /*
501 * Flags that alter standard formatting for conversion routines.
502 * These bits start after the range occupied by CONV_MASK_FMT_ALT.
503 */
504 #define CONV_FMT_DECIMAL 0x0100 /* conv_invalid_val() should print */
505 /* integer print as decimal */
506 /* (default is hex) */
507 #define CONV_FMT_SPACE 0x0200 /* conv_invalid_val() should append */
508 /* a space after the number. */
509 #define CONV_FMT_NOBKT 0x0400 /* conv_expn_field() should omit */
510 /* prefix and suffix strings */

512 /*
513 * A Val_desc structure is used to associate an ELF constant and
514 * the message code (Msg) for the string that corresponds to it.
515 *
516 * Val_desc2 adds v_osabi and v_mach fields to Val_desc, which allows
517 * for non-generic mappings that apply only to a specific OSABI/machine.
518 * Setting v_osabi to 0 (ELFOSABI_NONE) specifies that any OSABI matches.
519 * Similarly, setting v_mach to 0 (EM_MACH) matches any machine. Hence,
520 * setting v_osabi and v_mach to 0 in a Val_desc2 results in a generic item,

```

```

521 * and is equivalent to simply using a Val_desc.
522 *
523 * These structs are used in two different contexts:
524 *
525 * 1) To expand bit-field data items, using conv_expn_field() to
526 * process a NULL terminated array of Val_desc, or conv_expn_field2()
527 * to process a null terminated array of Val_desc2.
528 *
529 * 2) To represent sparse ranges of non-bitfield values, referenced via
530 * conv_ds_vd_t or conv_ds_vd2_t descriptors, as described below.
531 */
532 typedef struct {
533     Conv_elfvalue_t v_val;          /* expansion value */
534     Msg             v_msg;          /* associated message string code */
535 } Val_desc;
536 typedef struct {
537     Conv_elfvalue_t v_val;          /* expansion value */
538     uchar_t         v_osabi;        /* OSABI to which entry applies */
539     Half            v_mach;         /* Machine to which entry applies */
540     Msg             v_msg;          /* associated message string code */
541 } Val_desc2;
542
543 /*
544 * The conv_ds_XXX_t structs are used to pull together the information used
545 * to map non-bitfield values to strings. They are a variant family, sharing
546 * the same initial fields, with a generic "header" definition that can be
547 * used to read those common fields and determine which subcase is being
548 * seen. We do this instead of using a single struct containing a type code
549 * and a union in order to allow for static compile-time initialization.
550 *
551 * conv_ds_t is the base type, containing the initial fields common to all
552 * the variants. Variables of type conv_ds_t are never instantiated. This
553 * type exists only to provide a common pointer type that can reference
554 * any of the variants safely. In C++, it would be a virtual base class.
555 * The fields common to all the variants are:
556 *
557 *     ds_type: Identifies the variant
558 *     ds_baseval/ds_topval: The lower and upper bound of the range
559 *                       of values represented by this conv_ds_XXX_t descriptor.
560 *
561 * There are three different variants:
562 * conv_ds_msg_t (ds_type == CONV_DS_MSGARR)
563 * This structure references an array of message codes corresponding
564 * to consecutive ELF values. The first item in the array is the Msg
565 * code for the value given by ds_baseval. Consecutive strings follow
566 * in consecutive order. The final item corresponds to the value given
567 * by ds_topval. Zero (0) Msg values can be used to represent missing
568 * values. Entries with a 0 are quietly ignored.
569 *
570 * conv_ds_vd_t (ds_type == CONV_DS_VD)
571 * This structure employs a NULL terminated array of Val_desc structs.
572 * Each Val_desc supplies a mapping from a value in the range
573 * (ds_baseval <= value <= ds_topval). The values described need not
574 * be consecutive, and can be sparse. ds_baseval does not need to
575 * correspond to the first item, and ds_topval need not correspond to
576 * the final item.
577 *
578 * conv_ds_vd2_t (ds_type == CONV_DS_VD2)
579 * This structure employs a NULL terminated array of Val_desc2 structs,
580 * rather than Val_desc, adding the ability to specify OSABI and machine
581 * as part of the value/string mapping. It is otherwise the same thing
582 * as CONV_DS_VD.
583 */
584 typedef enum {
585     CONV_DS_MSGARR = 0,          /* Array of Msg */
586     CONV_DS_VD = 1,             /* Null terminated array of Val_desc */

```

```

587     CONV_DS_VD2 = 2,           /* Null terminated array of Val_desc2 */
588 } conv_ds_type_t;
589
590 #define CONV_DS_COMMON_FIELDS \
591     conv_ds_type_t ds_type;    /* Type of data structure used */ \
592     uint32_t       ds_baseval; /* Value of first item */ \
593     uint32_t       ds_topval;  /* Value of last item */
594
595 typedef struct {                /* Virtual base type --- do not instantiate */
596     CONV_DS_COMMON_FIELDS;
597 } conv_ds_t;
598 typedef struct {
599     CONV_DS_COMMON_FIELDS;
600     const Msg *ds_msg;
601 } conv_ds_msg_t;
602 typedef struct {
603     CONV_DS_COMMON_FIELDS;
604     const Val_desc *ds_vd;
605 } conv_ds_vd_t;
606 typedef struct {
607     CONV_DS_COMMON_FIELDS;
608     const Val_desc2 *ds_vd2;
609 } conv_ds_vd2_t;
610
611 /*
612 * The initialization of conv_ds_msg_t can be completely derived from
613 * its base value and the array of Msg codes. CONV_DS_MSG_INIT() is used
614 * to do that.
615 */
616 #define CONV_DS_MSG_INIT(_baseval, _arr) \
617     CONV_DS_MSGARR, \
618     _baseval + (sizeof (_arr) / sizeof (_arr[0])) - 1, _arr
619
620 /*
621 * Null terminated arrays of pointers to conv_ds_XXX_t structs are processed
622 * by conv_map_ds() to convert ELF constants to their symbolic names, and by
623 * conv_iter_ds() to iterate over all the available value/name combinations.
624 *
625 * These pointers are formed by casting the address of the specific
626 * variant types (described above) to generic base type pointer.
627 * CONV_DS_ADDR() is a convenience macro to take the address of
628 * one of these variants and turn it into a generic pointer.
629 */
630 #define CONV_DS_ADDR(_item) ((conv_ds_t *)&(_item))
631
632 /*
633 * Type used by libconv to represent osabi values passed to iteration
634 * functions. The type in the ELF header is uchar_t. However, every possible
635 * value 0-255 has a valid meaning, leaving us no extra value to assign
636 * to mean "ALL". Using Half for osabi leaves us the top byte to use for
637 * out of bound values.
638 *
639 * Non-iteration functions, and any code that does not need to use
640 * CONV_OSABI_ALL, should use uchar_t for osabi.
641 */
642 typedef Half conv_iter_osabi_t;
643
644 /*
645 * Many of the iteration functions accept an osabi or mach argument,
646 * used to specify the type of object being processed. The following
647 * values can be used to specify a wildcard that matches any item. Their
648 * values are carefully chosen to ensure that they cannot be interpreted
649 * as an otherwise valid osabi or machine.
650 */
651 #define CONV_OSABI_ALL 1024 /* Larger than can be represented by uchar_t */
652 #define CONV_MACH_ALL EM_NUM /* Never a valid machine type */

```



```

654 /*
655 * We compare Val_Desc2 descriptors with a specified osabi and machine
656 * to determine whether to use it or not. This macro encapsulates that logic.
657 *
658 * We consider an osabi to match when any of the following things hold:
659 *
660 * - The descriptor osabi is ELFOSABI_NONE.
661 * - The supplied osabi and the descriptor osabi match
662 * - The supplied osabi is ELFOSABI_NONE, and the descriptor osabi is
663 *   ELFOSABI_SOLARIS. Many operating systems, Solaris included,
664 *   produce or have produced ELFOSABI_NONE native objects, if only
665 *   because OSABI ranges are not an original ELF feature. We
666 *   give our own objects the home field advantage.
667 * - Iteration Only: An osabi value of CONV_OSABI_ALL is specified.
668 *
669 * We consider a machine to match when any of the following things hold:
670 *
671 * - The descriptor mach is EM_NONE.
672 * - The supplied mach and the descriptor mach match
673 * - Iteration Only: A mach value of CONV_MACH_ALL is specified.
674 *
675 * The special extra _ALL case for iteration is handled by defining a separate
676 * macro with the extra CONV_XXX_ALL tests.
677 */
678 #define CONV_VD2_SKIP_OSABI(_osabi, _vdp) \
679 (( _vdp->v_osabi != ELFOSABI_NONE) && ( _vdp->v_osabi != osabi) && \
680 (( _osabi != ELFOSABI_NONE) || ( _vdp->v_osabi != ELFOSABI_SOLARIS)))
681
682 #define CONV_VD2_SKIP_MACH(_mach, _vdp) \
683 (( _vdp->v_mach != EM_NONE) && ( _vdp->v_mach != _mach))
684
685 #define CONV_VD2_SKIP(_osabi, _mach, _vdp) \
686 (CONV_VD2_SKIP_OSABI(_osabi, _vdp) || CONV_VD2_SKIP_MACH(_mach, _vdp))
687
688 #define CONV_ITER_VD2_SKIP(_osabi, _mach, _vdp) \
689 ((CONV_VD2_SKIP_OSABI(_osabi, _vdp) && (_osabi != CONV_OSABI_ALL)) || \
690 (CONV_VD2_SKIP_MACH(_mach, _vdp) && (_mach != CONV_MACH_ALL)))
691
692
693 /*
694 * Possible return values from iteration functions.
695 */
696 typedef enum {
697     CONV_ITER_DONE,          /* Stop: No more iterations are desired */
698     CONV_ITER_CONT         /* Continue with following iterations */
699 } conv_iter_ret_t;
700
701 /*
702 * Prototype for caller supplied callback function to iteration functions.
703 */
704 typedef conv_iter_ret_t (* conv_iter_cb_t)(const char *str,
705     Conv_elfvalue_t value, void *uvalue);
706
707 /*
708 * User value block employed by conv_iter_strtol()
709 */
710 typedef struct {
711     const char    *csl_str;          /* String to search for */
712     size_t        csl_strlen;       /* # chars in csl_str to examine */
713     int           csl_found;        /* Init to 0, set to 1 if item found */
714     Conv_elfvalue_t csl_value;      /* If csl_found, resulting value */
715 } conv_strtol_uvalue_t;
716
717 /*
718 * conv_expn_field() is willing to supply default strings for the

```

```

719 * prefix, separator, and suffix arguments, if they are passed as NULL.
720 * The caller needs to know how much room to allow for these items.
721 * These values supply those sizes.
722 */
723 #define CONV_EXPN_FIELD_DEF_PREFIX_SIZE 2    /* Default is "[ " */
724 #define CONV_EXPN_FIELD_DEF_SEP_SIZE 1     /* Default is " " */
725 #define CONV_EXPN_FIELD_DEF_SUFFIX_SIZE 2  /* Default is "]" */
726
727 /*
728 * conv_expn_field() requires a large number of inputs, many of which
729 * can be NULL to accept default behavior. An argument of the following
730 * type is used to supply them.
731 */
732 typedef struct {
733     char *buf;                /* Buffer to receive generated string */
734     size_t bufsize;          /* sizeof(buf) */
735     const char **lead_str;   /* NULL, or array of pointers to strings to */
736                             /* be output at the head of the list. */
737                             /* Last entry must be NULL. */
738     Xword oflags;           /* Bits for which output strings are desired */
739     Xword rflags;           /* Bits for which a numeric value should be */
740                             /* output if vdp does not provide str. */
741                             /* Must be a proper subset of oflags */
742     const char *prefix;     /* NULL, or string to prefix output with */
743                             /* If NULL, "[ " is used. */
744     const char *sep;        /* NULL, or string to separate output items */
745                             /* with. If NULL, " " is used. */
746     const char *suffix;     /* NULL, or string to suffix output with */
747                             /* If NULL, "]" is used. */
748 } CONV_EXPN_FIELD_ARG;
749
750 /*
751 * Callback function for conv_str_to_c_literal(). A user supplied function
752 * of this type is called by conv_str_to_c_literal() in order to dispatch
753 * the translated output characters.
754 */
755 * buf - Pointer to output text
756 * n - # of characters to output
757 * uvalue - User value argument to conv_str_to_c_literal(),
758 *         passed through without interpretation.
759 */
760 typedef void Conv_str_to_c_literal_func_t(const void *ptr,
761     size_t size, void *uvalue);
762
763 /*
764 * Generic miscellaneous interfaces
765 */
766 extern uchar_t conv_check_native(char **, char **);
767 extern const char *conv_lddstub(int);
768 extern int conv_strproc_isspace(int);
769 extern char *conv_strproc_trim(char *);
770 extern Boolean conv_strproc_extract_value(char *, size_t, int,
771     const char **);
772 extern int conv_sys_eclass(void);
773 extern int conv_translate_c_esc(char **);
774
775 /*
776 * Generic core formatting and iteration functionality
777 */
778 extern conv_iter_ret_t conv_iter_ds(conv_iter_osabi_t, Half,
779     const conv_ds_t **, conv_iter_cb_t, void *,
780     const char *);
781 extern conv_iter_ret_t conv_iter_ds_msg(const conv_ds_msg_t *,
782     conv_iter_cb_t, void *, const char *);
783 extern conv_iter_ret_t conv_iter_vd(const Val_desc *, conv_iter_cb_t,
784     void *, const char *);

```

```

785 extern conv_iter_ret_t _conv_iter_vd2(conv_iter_osabi_t, Half,
786     const Val_desc2 *, conv_iter_cb_t, void *,
787     const char *);
788 extern int conv_iter_strtol_init(const char *,
789     conv_strtol_uvalue_t *);
790 extern conv_iter_ret_t conv_iter_strtol(const char *, Conv_elfvalue_t, void *);
791 extern const char *_conv_map_ds(uchar_t, Half, Conv_elfvalue_t,
792     const conv_ds_t **, Conv_fmt_flags_t,
793     Conv_inv_buf_t *, const char *);

796 /*
797  * Generic formatting interfaces.
798  */
799 extern const char *conv_bnd_obj(uint_t, Conv_bnd_obj_buf_t *);
800 extern const char *conv_bnd_type(uint_t, Conv_bnd_type_buf_t *);
801 extern const char *conv_config_feat(int, Conv_config_feat_buf_t *);
802 extern const char *conv_config_obj(ushort_t, Conv_config_obj_buf_t *);
803 extern const char *conv_config_upm(const char *, const char *,
804     const char *, size_t);
805 extern const char *conv_cnote_auxv_af(Word, Conv_fmt_flags_t,
806     Conv_cnote_auxv_af_buf_t *);
807 extern const char *conv_cnote_auxv_type(Word, Conv_fmt_flags_t,
808     Conv_inv_buf_t *);
809 extern const char *conv_cnote_cc_content(Lword, Conv_fmt_flags_t,
810     Conv_cnote_cc_content_buf_t *);
811 extern const char *conv_cnote_errno(int, Conv_fmt_flags_t,
812     Conv_inv_buf_t *);
813 extern const char *conv_cnote_fault(Word, Conv_fmt_flags_t,
814     Conv_inv_buf_t *);
815 extern const char *conv_cnotefltset(uint32_t *, int,
816     Conv_fmt_flags_t, Conv_cnotefltset_buf_t *);
817 extern const char *conv_cnote_old_pr_flags(int, Conv_fmt_flags_t,
818     Conv_cnote_old_pr_flags_buf_t *);
819 extern const char *conv_cnote_pr_dmodel(Word, Conv_fmt_flags_t,
820     Conv_inv_buf_t *);
821 extern const char *conv_cnote_pr_flags(int, Conv_fmt_flags_t,
822     Conv_cnote_pr_flags_buf_t *);
823 extern const char *conv_cnote_proc_flag(int, Conv_fmt_flags_t,
824     Conv_cnote_proc_flag_buf_t *);
825 extern const char *conv_cnote_pr_regname(Half, int, Conv_fmt_flags_t,
826     Conv_inv_buf_t *inv_buf);
827 extern const char *conv_cnote_pr_stype(Word, Conv_fmt_flags_t,
828     Conv_inv_buf_t *);
829 extern const char *conv_cnote_pr_what(short, short, Conv_fmt_flags_t,
830     Conv_inv_buf_t *);
831 extern const char *conv_cnote_pr_why(short, Conv_fmt_flags_t,
832     Conv_inv_buf_t *);
833 extern const char *conv_cnote_priv(int, Conv_fmt_flags_t,
834     Conv_inv_buf_t *);
835 extern const char *conv_prsecflags(secflagset_t, Conv_fmt_flags_t,
836     Conv_secflags_buf_t *);
837 #endif /* ! codereview */
838 extern const char *conv_cnote_psetid(int, Conv_fmt_flags_t,
839     Conv_inv_buf_t *);
840 extern const char *conv_cnote_sa_flags(int, Conv_fmt_flags_t,
841     Conv_cnote_sa_flags_buf_t *);
842 extern const char *conv_cnote_signal(Word, Conv_fmt_flags_t,
843     Conv_inv_buf_t *);
844 extern const char *conv_cnote_si_code(Half, int, int, Conv_fmt_flags_t,
845     Conv_inv_buf_t *);
846 extern const char *conv_cnote_sigset(uint32_t *, int,
847     Conv_fmt_flags_t, Conv_cnote_sigset_buf_t *);
848 extern const char *conv_cnote_ss_flags(int, Conv_fmt_flags_t,
849     Conv_cnote_ss_flags_buf_t *);
850 extern const char *conv_cnote_syscall(Word, Conv_fmt_flags_t,

```

```

851     Conv_inv_buf_t *);
852 extern const char *conv_cnote_sysset(uint32_t *, int,
853     Conv_fmt_flags_t, Conv_cnote_sysset_buf_t *);
854 extern const char *conv_cnote_fileflags(uint32_t, Conv_fmt_flags_t,
855     char *, size_t);
856 extern const char *conv_cnote_filemode(uint32_t, Conv_fmt_flags_t,
857     char *, size_t);
858 extern const char *conv_cnote_type(Word, Conv_fmt_flags_t,
859     Conv_inv_buf_t *);
860 extern const char *conv_def_tag(Symref, Conv_inv_buf_t *);
861 extern const char *conv_demangle_name(const char *);
862 extern const char *conv_dl_flag(int, Conv_fmt_flags_t,
863     Conv_dl_flag_buf_t *);
864 extern const char *conv_dl_info(int);
865 extern const char *conv_dl_mode(int, int, Conv_dl_mode_buf_t *);
866 extern const char *conv_dwarf_cfa(uchar_t, Word, Conv_fmt_flags_t,
867     Conv_inv_buf_t *);
868 extern const char *conv_dwarf_ehe(uint_t, Conv_dwarf_ehe_buf_t *);
869 extern const char *conv_dwarf_regname(Half, Word, Conv_fmt_flags_t,
870     int *, Conv_inv_buf_t *);
871 extern const char *conv_ehdr_abivers(uchar_t, Word, Conv_fmt_flags_t,
872     Conv_inv_buf_t *);
873 extern const char *conv_ehdr_class(uchar_t, Conv_fmt_flags_t,
874     Conv_inv_buf_t *);
875 extern const char *conv_ehdr_data(uchar_t, Conv_fmt_flags_t,
876     Conv_inv_buf_t *);
877 extern const char *conv_ehdr_flags(Half, Word, Conv_fmt_flags_t,
878     Conv_ehdr_flags_buf_t *);
879 extern const char *conv_ehdr_mach(Half, Conv_fmt_flags_t,
880     Conv_inv_buf_t *);
881 extern const char *conv_ehdr_osabi(uchar_t, Conv_fmt_flags_t,
882     Conv_inv_buf_t *);
883 extern const char *conv_ehdr_type(uchar_t, Half, Conv_fmt_flags_t,
884     Conv_inv_buf_t *);
885 extern const char *conv_ehdr_vers(Word, Conv_fmt_flags_t,
886     Conv_inv_buf_t *);
887 extern const char *conv_elfdata_type(Elf_Type, Conv_inv_buf_t *);
888 extern const char *conv_ent_flags(ec_flags_t, Conv_ent_flags_buf_t *);
889 extern const char *conv_ent_files_flags(Word, Conv_fmt_flags_t fmt_flags,
890     Conv_ent_files_flags_buf_t *);
891 extern const char *conv_la_activity(uint_t, Conv_fmt_flags_t,
892     Conv_inv_buf_t *);
893 extern const char *conv_la_bind(uint_t, Conv_la_bind_buf_t *);
894 extern const char *conv_la_search(uint_t, Conv_la_search_buf_t *);
895 extern const char *conv_la_symbind(uint_t, Conv_la_symbind_buf_t *);
896 extern const char *conv_grphdl_flags(uint_t, Conv_grphdl_flags_buf_t *);
897 extern const char *conv_grpdesc_flags(uint_t, Conv_grpdesc_flags_buf_t *);
898 extern const char *conv_isalist(void);
899 extern const char *conv_mapfile_version(Word, Conv_fmt_flags_t,
900     Conv_inv_buf_t *);
901 extern const char *conv_phdr_flags(uchar_t, Word, Conv_fmt_flags_t,
902     Conv_phdr_flags_buf_t *);
903 extern const char *conv_phdr_type(uchar_t, Half, Word, Conv_fmt_flags_t,
904     Conv_inv_buf_t *);
905 extern const char *conv_reject_desc(Rej_desc *, Conv_reject_desc_buf_t *,
906     Half mach);
907 extern const char *conv_reloc_type(Half, Word, Conv_fmt_flags_t,
908     Conv_inv_buf_t *);
909 extern const char *conv_reloc_type_static(Half, Word, Conv_fmt_flags_t);
910 extern const char *conv_reloc_386_type(Word, Conv_fmt_flags_t,
911     Conv_inv_buf_t *);
912 extern const char *conv_reloc_amd64_type(Word, Conv_fmt_flags_t,
913     Conv_inv_buf_t *);
914 extern const char *conv_reloc_SPARC_type(Word, Conv_fmt_flags_t,
915     Conv_inv_buf_t *);
916 extern const char *conv_sec_type(uchar_t, Half, Word, Conv_fmt_flags_t,

```

```

917     Conv_inv_buf_t *);
918 extern const char *conv_seg_flags(sg_flags_t, Conv_seg_flags_buf_t *);
919 extern void conv_str_to_c_literal(const char *buf, size_t n,
920     Conv_str_to_c_literal_func_t *cb_func,
921     void *uvalue);
922 extern const char *conv_sym_info_bind(uchar_t, Conv_fmt_flags_t,
923     Conv_inv_buf_t *);
924 extern const char *conv_sym_info_type(Half, uchar_t, Conv_fmt_flags_t,
925     Conv_inv_buf_t *);
926 extern const char *conv_sym_shndx(uchar_t, Half, Half, Conv_fmt_flags_t,
927     Conv_inv_buf_t *);
928 extern const char *conv_sym_other(uchar_t, Conv_inv_buf_t *);
929 extern const char *conv_sym_other_vis(uchar_t, Conv_fmt_flags_t,
930     Conv_inv_buf_t *);
931 extern const char *conv_syminfo_boundto(Half, Conv_fmt_flags_t,
932     Conv_inv_buf_t *);
933 extern const char *conv_syminfo_flags(Half, Conv_fmt_flags_t,
934     Conv_syminfo_flags_buf_t *);
935 extern const char *conv_time(struct timeval *, struct timeval *,
936     Conv_time_buf_t *);
937 extern Uts_desc *conv_uts(void);
938 extern const char *conv_ver_flags(Half, Conv_fmt_flags_t,
939     Conv_ver_flags_buf_t *);
940 extern const char *conv_ver_index(Versym, int, Conv_inv_buf_t *);

943 /*
944  * Generic iteration interfaces.
945  */
946 extern conv_iter_ret_t conv_iter_cap_tags(Conv_fmt_flags_t, conv_iter_cb_t,
947     void *);
948 extern conv_iter_ret_t conv_iter_cap_val_hwl(Half, Conv_fmt_flags_t,
949     conv_iter_cb_t, void *);
950 extern conv_iter_ret_t conv_iter_cap_val_hw2(Half, Conv_fmt_flags_t,
951     conv_iter_cb_t, void *);
952 extern conv_iter_ret_t conv_iter_cap_val_sf1(Conv_fmt_flags_t, conv_iter_cb_t,
953     void *);

955 extern conv_iter_ret_t conv_iter_dyn_feature1(Conv_fmt_flags_t, conv_iter_cb_t,
956     void *);
957 extern conv_iter_ret_t conv_iter_dyn_flag(Conv_fmt_flags_t, conv_iter_cb_t,
958     void *);
959 extern conv_iter_ret_t conv_iter_dyn_flag1(Conv_fmt_flags_t, conv_iter_cb_t,
960     void *);
961 extern conv_iter_ret_t conv_iter_dyn_posflag1(Conv_fmt_flags_t, conv_iter_cb_t,
962     void *);
963 extern conv_iter_ret_t conv_iter_dyn_tag(conv_iter_osabi_t, Half,
964     Conv_fmt_flags_t, conv_iter_cb_t, void *);

966 extern conv_iter_ret_t conv_iter_ehdr_abivers(conv_iter_osabi_t,
967     Conv_fmt_flags_t, conv_iter_cb_t, void *);
968 extern conv_iter_ret_t conv_iter_ehdr_class(Conv_fmt_flags_t, conv_iter_cb_t,
969     void *);
970 extern conv_iter_ret_t conv_iter_ehdr_data(Conv_fmt_flags_t, conv_iter_cb_t,
971     void *);
972 extern conv_iter_ret_t conv_iter_ehdr_eident(Conv_fmt_flags_t, conv_iter_cb_t,
973     void *);
974 extern conv_iter_ret_t conv_iter_ehdr_flags(Half, Conv_fmt_flags_t,
975     conv_iter_cb_t, void *);
976 extern conv_iter_ret_t conv_iter_ehdr_mach(Conv_fmt_flags_t, conv_iter_cb_t,
977     void *);
978 extern conv_iter_ret_t conv_iter_ehdr_osabi(Conv_fmt_flags_t, conv_iter_cb_t,
979     void *);
980 extern conv_iter_ret_t conv_iter_ehdr_type(conv_iter_osabi_t, Conv_fmt_flags_t,
981     conv_iter_cb_t, void *);
982 extern conv_iter_ret_t conv_iter_ehdr_vers(Conv_fmt_flags_t, conv_iter_cb_t,

```

```

983     void *);

985 extern conv_iter_ret_t conv_iter_phdr_flags(conv_iter_osabi_t,
986     Conv_fmt_flags_t, conv_iter_cb_t, void *);
987 extern conv_iter_ret_t conv_iter_phdr_type(conv_iter_osabi_t, Conv_fmt_flags_t,
988     conv_iter_cb_t, void *);

990 extern conv_iter_ret_t conv_iter_sec_flags(conv_iter_osabi_t, Half,
991     Conv_fmt_flags_t, conv_iter_cb_t, void *);
992 extern conv_iter_ret_t conv_iter_sec_syntab(conv_iter_osabi_t,
993     Conv_fmt_flags_t, conv_iter_cb_t, void *);
994 extern conv_iter_ret_t conv_iter_sec_type(conv_iter_osabi_t, Half,
995     Conv_fmt_flags_t, conv_iter_cb_t, void *);

997 extern conv_iter_ret_t conv_iter_sym_info_bind(Conv_fmt_flags_t,
998     conv_iter_cb_t, void *);
999 extern conv_iter_ret_t conv_iter_sym_other_vis(Conv_fmt_flags_t,
1000     conv_iter_cb_t, void *);
1001 extern conv_iter_ret_t conv_iter_sym_shndx(conv_iter_osabi_t, Half,
1002     Conv_fmt_flags_t, conv_iter_cb_t, void *);
1003 extern conv_iter_ret_t conv_iter_sym_info_type(Half, Conv_fmt_flags_t,
1004     conv_iter_cb_t, void *);

1006 extern conv_iter_ret_t conv_iter_syminfo_boundto(Conv_fmt_flags_t,
1007     conv_iter_cb_t, void *);
1008 extern conv_iter_ret_t conv_iter_syminfo_flags(Conv_fmt_flags_t,
1009     conv_iter_cb_t, void *);

1011 /*
1012  * Define all class specific routines.
1013  */
1014 #if defined(_ELF64)
1015 #define conv_cap_tag conv64_cap_tag
1016 #define conv_cap_val conv64_cap_val
1017 #define conv_cap_val_hwl conv64_cap_val_hwl
1018 #define conv_cap_val_hw2 conv64_cap_val_hw2
1019 #define conv_cap_val_sf1 conv64_cap_val_sf1
1020 #define conv_dyn_feature1 conv64_dyn_feature1
1021 #define conv_dyn_flag1 conv64_dyn_flag1
1022 #define conv_dyn_flag conv64_dyn_flag
1023 #define conv_dyn_posflag1 conv64_dyn_posflag1
1024 #define conv_dyn_tag conv64_dyn_tag
1025 #define _conv_expn_field _conv64_expn_field
1026 #define _conv_expn_field2 _conv64_expn_field2
1027 #define conv_invalid_val conv64_invalid_val
1028 #define conv_sec_flags conv64_sec_flags
1029 #define conv_sec_linkinfo conv64_sec_linkinfo
1030 #define conv_sym_value conv64_sym_value
1031 #define conv_sym_SPARC_value conv64_sym_SPARC_value
1032 #else
1033 #define conv_cap_tag conv32_cap_tag
1034 #define conv_cap_val conv32_cap_val
1035 #define conv_cap_val_hwl conv32_cap_val_hwl
1036 #define conv_cap_val_hw2 conv32_cap_val_hw2
1037 #define conv_cap_val_sf1 conv32_cap_val_sf1
1038 #define conv_dyn_feature1 conv32_dyn_feature1
1039 #define conv_dyn_flag1 conv32_dyn_flag1
1040 #define conv_dyn_flag conv32_dyn_flag
1041 #define conv_dyn_posflag1 conv32_dyn_posflag1
1042 #define conv_dyn_tag conv32_dyn_tag
1043 #define _conv_expn_field _conv32_expn_field
1044 #define _conv_expn_field2 conv32_expn_field2
1045 #define conv_invalid_val conv32_invalid_val
1046 #define conv_sec_flags conv32_sec_flags
1047 #define conv_sec_linkinfo conv32_sec_linkinfo
1048 #define conv_sym_value conv32_sym_value

```

```

1049 #define conv_sym_SPARC_value    conv32_sym_SPARC_value
1050 #endif

1052 /*
1053  * ELFCLASS-specific core formatting functionality
1054  */
1055 extern int          _conv_expn_field(CONV_EXPN_FIELD_ARG *,
1056                                     const Val_desc *, Conv_fmt_flags_t, const char *);
1057 extern int          _conv_expn_field2(CONV_EXPN_FIELD_ARG *, uchar_t,
1058                                     Half, const Val_desc2 *, Conv_fmt_flags_t,
1059                                     const char *);
1060 extern const char   *conv_invalid_val(Conv_inv_buf_t *, Xword,
1061                                     Conv_fmt_flags_t);

1063 /*
1064  * ELFCLASS-specific formatting interfaces.
1065  */
1066 extern const char   *conv_cap_tag(Xword, Conv_fmt_flags_t,
1067                                   Conv_inv_buf_t *);
1068 extern const char   *conv_cap_val(Xword, Xword, Half, Conv_fmt_flags_t,
1069                                   Conv_cap_val_buf_t *);
1070 extern const char   *conv_cap_val_hwl(Xword, Half, Conv_fmt_flags_t,
1071                                       Conv_cap_val_hwl_buf_t *);
1072 extern const char   *conv_cap_val_hw2(Xword, Half, Conv_fmt_flags_t,
1073                                       Conv_cap_val_hw2_buf_t *);
1074 extern const char   *conv_cap_val_sfl(Xword, Half, Conv_fmt_flags_t,
1075                                       Conv_cap_val_sfl_buf_t *);
1076 extern const char   *conv_dyn_flag1(Xword, Conv_fmt_flags_t,
1077                                     Conv_dyn_flag1_buf_t *);
1078 extern const char   *conv_dyn_flag(Xword, Conv_fmt_flags_t,
1079                                     Conv_dyn_flag_buf_t *);
1080 extern const char   *conv_dyn_posflag1(Xword, Conv_fmt_flags_t,
1081                                       Conv_dyn_posflag1_buf_t *);
1082 extern const char   *conv_dyn_tag(Xword, uchar_t, Half, Conv_fmt_flags_t,
1083                                   Conv_inv_buf_t *);
1084 extern const char   *conv_dyn_feature1(Xword, Conv_fmt_flags_t,
1085                                       Conv_dyn_feature1_buf_t *);
1086 extern const char   *conv_sec_flags(uchar_t osabi, Half mach, Xword,
1087                                     Conv_fmt_flags_t, Conv_sec_flags_buf_t *);
1088 extern const char   *conv_sec_linkinfo(Word, Xword, Conv_inv_buf_t *);
1089 extern const char   *conv_sym_value(Half, uchar_t, Addr, Conv_inv_buf_t *);
1090 extern const char   *conv_sym_SPARC_value(Addr, Conv_fmt_flags_t,
1091                                           Conv_inv_buf_t *);

1093 /*
1094  * Define macros for _conv_XXX() routines that accept local_sgs_msg as the
1095  * final argument. The macros hide that argument from the caller's view and
1096  * supply the SGS message array for the file from which the macro is used
1097  * in its place. This trick is used to allow these functions to access the
1098  * message strings from any source file they are called from.
1099  */
1100 #define conv_expn_field(arg, _vdp, _fmt_flags) \
1101   _conv_expn_field(arg, _vdp, _fmt_flags, MSG_SGS_LOCAL_ARRAY)

1103 #define conv_expn_field2(arg, _osabi, _mach, _vdp, _fmt_flags) \
1104   _conv_expn_field2(arg, _osabi, _mach, _vdp, _fmt_flags, \
1105   MSG_SGS_LOCAL_ARRAY)

1107 #define conv_iter_ds(_osabi, _mach, _dsp, _func, _uvalue) \
1108   _conv_iter_ds(_osabi, _mach, _dsp, _func, _uvalue, MSG_SGS_LOCAL_ARRAY)

1110 #define conv_iter_vd(_vdp, _func, _uvalue) \
1111   _conv_iter_vd(_vdp, _func, _uvalue, MSG_SGS_LOCAL_ARRAY)

1113 #define conv_iter_vd2(_osabi, _mach, _vdp, _func, _uvalue) \
1114   _conv_iter_vd2(_osabi, _mach, _vdp, _func, _uvalue, MSG_SGS_LOCAL_ARRAY)

```

```

1116 #define conv_map_ds(_osabi, _mach, _value, _dsp, _fmt_flags, _inv_buf) \
1117   _conv_map_ds(_osabi, _mach, _value, _dsp, _fmt_flags, _inv_buf, \
1118   MSG_SGS_LOCAL_ARRAY)

1121 #ifdef __cplusplus
1122 }
1123 #endif

1125 #endif /* _CONV_H */

```

```

*****
66593 Thu Jun 30 22:01:26 2016
new/usr/src/cmd/sgs/include/libld.h
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
_____unchanged_portion_omitted_____

241 /*
242  * Output file processing structure
243  */
244 typedef Lword  ofl_flag_t;
245 typedef Word   ofl_guidedeflag_t;
246 struct ofl_desc {
247     char          *ofl_sgssid; /* link-editor identification */
248     const char    *ofl_name;  /* full file name */
249     Elf           *ofl_elf;   /* elf_memory() elf descriptor */
250     Elf           *ofl_welf;  /* ELF_C_WRITE elf descriptor */
251     Ehdr         *ofl_dehdr; /* default elf header, and new elf */
252     Ehdr         *ofl_nehdr; /* header describing this file */
253     Phdr         *ofl_phdr;  /* program header descriptor */
254     Phdr         *ofl_tlsphdr; /* TLS phdr */
255     int          ofl_fd;     /* file descriptor */
256     size_t       ofl_size;   /* image size */
257     APlist       *ofl_maps;  /* list of input mapfiles */
258     APlist       *ofl_segs;  /* list of segments */
259     APlist       *ofl_segs_order; /* SEGMENT_ORDER segments */
260     avl_tree_t   ofl_segs_avl; /* O(log N) access to named segments */
261     APlist       *ofl_ents;  /* list of entrance descriptors */
262     avl_tree_t   ofl_ents_avl; /* O(log N) access to named ent. desc */
263     APlist       *ofl_objjs; /* relocatable object file list */
264     Word         *ofl_objjscnt; /* and count */
265     APlist       *ofl_ars;   /* archive library list */
266     Word         *ofl_arscnt; /* and count */
267     int          ofl_ars_gsndx; /* archive group argv index. 0 means */
268     /* no current group, < 0 means */
269     /* error reported. >0 is cur ndx */
270     Word         ofl_ars_gsndx; /* current -zrescan-start ofl_ars ndx */
271     APlist       *ofl_sos;   /* shared object list */
272     Word         *ofl_soscnt; /* and count */
273     APlist       *ofl_soneed; /* list of implicitly required .so's */
274     APlist       *ofl_socnt;  /* list of .so control definitions */
275     Rel_cache    ofl_outrels; /* list of output relocations */
276     Rel_cache    ofl_actrels; /* list of relocations to perform */
277     APlist       *ofl_relaux; /* Rel_aux cache for outrels/actrels */
278     Word         *ofl_entrelscnt; /* no of relocations entered */
279     AList        *ofl_copyrels; /* list of copy relocations */
280     APlist       *ofl_ordered; /* list of shf_ordered sections */
281     APlist       *ofl_symdntent; /* list of syminfo symbols that need */
282     /* to reference .dynamic entries */
283     APlist       *ofl_ismove; /* list of .SUNW_move sections */
284     APlist       *ofl_ismoverel; /* list of relocation input section */
285     /* targeting to expanded area */
286     APlist       *ofl_parsyms; /* list of partially initialized */
287     /* symbols (ie. move symbols) */
288     APlist       *ofl_extrarels; /* relocation sections which have */
289     /* a NULL sh_info */
290     avl_tree_t   *ofl_groups; /* pointer to head of Groups AVL tree */
291     APlist       *ofl_initarray; /* list of init array func names */
292     APlist       *ofl_finiarray; /* list of fini array func names */
293     APlist       *ofl_preiarray; /* list of preinit array func names */
294     APlist       *ofl_rtldinfo; /* list of rtldinfo syms */
295     APlist       *ofl_osgroups; /* list of output GROUP sections */

```

```

296     APlist       *ofl_ostlsseg; /* pointer to sections in TLS segment */
297     APlist       *ofl_unwind; /* list of unwind output sections */
298     Os_desc     *ofl_unwindhdr; /* Unwind hdr */
299     avl_tree_t   ofl_symavl; /* pointer to head of Syms AVL tree */
300     Sym_desc     *ofl_regssyms; /* array of potential register */
301     Word        *ofl_regssymsno; /* symbols and array count */
302     Word        *ofl_regssymcnt; /* no. of output register symbols */
303     Word        *ofl_lregssymcnt; /* no. of local register symbols */
304     Sym_desc     *ofl_dtracesym; /* ld -zdrace= */
305     ofl_flag_t  *ofl_flags; /* various state bits, args etc. */
306     ofl_flag_t  *ofl_flags1; /* more flags */
307     void        *ofl_entry; /* entry point (-e and Sym_desc *) */
308     char        *ofl_filtees; /* shared objects we are a filter for */
309     const char   *ofl_soname; /* (-h option) output file name for */
310     /* dynamic structure */
311     const char   *ofl_interp; /* interpreter name used by exec() */
312     char         *ofl_rpath; /* run path to store in .dynamic */
313     char         *ofl_config; /* config path to store in .dynamic */
314     APlist       *ofl_ulibdirs; /* user supplied library search list */
315     APlist       *ofl_dlibdirs; /* default library search list */
316     Word        *ofl_vercnt; /* number of versions to generate */
317     APlist       *ofl_verdesc; /* list of version descriptors */
318     size_t       *ofl_verdefsz; /* size of version definition section */
319     size_t       *ofl_verneedsz; /* size of version needed section */
320     Word        *ofl_entercnt; /* no. of global symbols entered */
321     Word        *ofl_globcnt; /* no. of global symbols to output */
322     Word        *ofl_scopecnt; /* no. of scoped symbols to output */
323     Word        *ofl_dynscopecnt; /* no. scoped syms in .SUNW_ldynsym */
324     Word        *ofl_elimcnt; /* no. of eliminated symbols */
325     Word        *ofl_locscnt; /* no. of local symbols in .symtab */
326     Word        *ofl_dynlocscnt; /* no. local symbols in .SUNW_ldynsym */
327     Word        *ofl_dynsymssortcnt; /* no. ndx in .SUNW_dynsymssort */
328     Word        *ofl_dyntlssortcnt; /* no. ndx in .SUNW_dyntlssort */
329     Word        *ofl_dynshdrscnt; /* no. of output section in .dynsym */
330     Word        *ofl_shdrscnt; /* no. of output sections */
331     Word        *ofl_caplocscnt; /* no. of local capabilities symbols */
332     Word        *ofl_capsymcnt; /* no. of symbol capabilities entries */
333     /* required */
334     Word        *ofl_capchaincnt; /* no. of Capchain symbols */
335     APlist       *ofl_capgroups; /* list of capabilities groups */
336     avl_tree_t   *ofl_capfamilies; /* capability family AVL tree */
337     Str_tbl      *ofl_shdrstrtab; /* Str_tbl for shdr strtab */
338     Str_tbl      *ofl_strtab; /* Str_tbl for symtab strtab */
339     Str_tbl      *ofl_dynstrtab; /* Str_tbl for dynsym strtab */
340     Gotndx       *ofl_tlsldgotndx; /* index to LD TLS_index structure */
341     Xword        *ofl_relocsz; /* size of output relocations */
342     Xword        *ofl_relocgotsz; /* size of .got relocations */
343     Xword        *ofl_relocpltsz; /* size of .plt relocations */
344     Xword        *ofl_relocbssz; /* size of .bss (copy) relocations */
345     Xword        *ofl_relocrelsz; /* size of .rel[a] relocations */
346     Word         *ofl_relocincnt; /* no. of input relocations */
347     Word         *ofl_relococnt; /* tot number of output relocations */
348     Word         *ofl_relococntsub; /* tot numb of output relocations to */
349     /* skip (-zignore) */
350     Word         *ofl_relocrelcnt; /* tot number of relative */
351     /* relocations */
352     Word         *ofl_gotcnt; /* no. of .got entries */
353     Word         *ofl_pltcnt; /* no. of .plt entries */
354     Word         *ofl_pltpad; /* no. of .plt padd entries */
355     Word         *ofl_hashbkts; /* no. of hash buckets required */
356     Is_desc     *ofl_isbss; /* .bss input section (globals) */
357     Is_desc     *ofl_lsbss; /* .lbss input section (globals) */
358     Is_desc     *ofl_istlsbss; /* .tlsbss input section (globals) */
359     Is_desc     *ofl_isplayexpn; /* -z nopartial .data input section */
360     Os_desc     *ofl_osdynamic; /* .dynamic output section */
361     Os_desc     *ofl_osdynsym; /* .dynsym output section */

```

```

362 Os_desc *ofl_osldynsym; /* .SUNW_ldynsym output section */
363 Os_desc *ofl_dynstr; /* .dynstr output section */
364 Os_desc *ofl_osdynsym; /* .SUNW_dynsym output section */
365 Os_desc *ofl_osdntlssort; /* .SUNW_dyntlssort output section */
366 Os_desc *ofl_osgot; /* .got output section */
367 Os_desc *ofl_oshash; /* .hash output section */
368 Os_desc *ofl_osinitarray; /* .init_array output section */
369 Os_desc *ofl_osfiniarray; /* .fini_array output section */
370 Os_desc *ofl_ospreinitarray; /* .preinit_array output section */
371 Os_desc *ofl_osinterp; /* .interp output section */
372 Os_desc *ofl_oscapi; /* .SUNW_cap output section */
373 Os_desc *ofl_oscapiinfo; /* .SUNW_capinfo output section */
374 Os_desc *ofl_oscapichain; /* .SUNW_capchain output section */
375 Os_desc *ofl_osplt; /* .plt output section */
376 Os_desc *ofl_osmove; /* .SUNW_move output section */
377 Os_desc *ofl_osrelhead; /* first relocation section */
378 Os_desc *ofl_osrel; /* .rel[a] relocation section */
379 Os_desc *ofl_osshstrtab; /* .shstrtab output section */
380 Os_desc *ofl_osstrtab; /* .strtab output section */
381 Os_desc *ofl_ossymtab; /* .symtab output section */
382 Os_desc *ofl_ossymshndx; /* .symtab_shndx output section */
383 Os_desc *ofl_osdynshndx; /* .dynsym_shndx output section */
384 Os_desc *ofl_osldynshndx; /* .SUNW_ldynsym_shndx output sec */
385 Os_desc *ofl_osverdef; /* .version definition output section */
386 Os_desc *ofl_osverneed; /* .version needed output section */
387 Os_desc *ofl_osversym; /* .version symbol ndx output section */
388 Word ofl_dtflags_l; /* DT_FLAGS_1 entries */
389 Word ofl_dtflags; /* DT_FLAGS entries */
390 Os_desc *ofl_ossyminfo; /* .SUNW_syminfo output section */
391 Half ofl_parexpndx; /* -z nopartial section index */
392 /* Ref. at perform_outreloc() in */
393 /* libld/{mach}/machrel.c */
394 Xword *ofl_checksum; /* DT_CHECKSUM value address */
395 char *ofl_depaudit; /* dependency auditing required (-P) */
396 char *ofl_audit; /* object auditing required (-p) */
397 Alist *ofl_symfilters; /* per-symbol filters and their */
398 Alist *ofl_dtsfilters; /* associated .dynamic/.dynstrs */
399 Objcapset *ofl_objcapset; /* object capabilities */
400 Lm_list *ofl_lm; /* runtime link-map list */
401 Gottable *ofl_gottable; /* debugging got information */
402 Rlxrel_cache ofl_sr_cache; /* Cache last result from */
403 /* sloppy_comdat_reloc() */
404 Aplist *ofl_maptext; /* mapfile added text sections */
405 Aplist *ofl_mapdata; /* mapfile added data sections */
406 avl_tree_t *ofl_wrap; /* -z wrap symbols */
407 ofl_guideflag_t *ofl_guideflags; /* -z guide flags */
408 Aplist *ofl_asrdeflib; /* -z assert-deflib exceptions */
409 int ofl_aslr; /* -z aslr, -1 disable, 1 enable */
410 #endif /* ! codereview */
411 };

413 #define FLG_OF_DYNAMIC 0x00000001 /* generate dynamic output module */
414 #define FLG_OF_STATIC 0x00000002 /* generate static output module */
415 #define FLG_OF_EXEC 0x00000004 /* generate an executable */
416 #define FLG_OF_RELOBJ 0x00000008 /* generate a relocatable object */
417 #define FLG_OF_SHAROBJ 0x00000010 /* generate a shared object */
418 #define FLG_OF_BFLAG 0x00000020 /* do no special plt building: -b */
419 #define FLG_OF_IGNENV 0x00000040 /* ignore LD_LIBRARY_PATH: -i */
420 #define FLG_OF_STRIP 0x00000080 /* strip output: -s */
421 #define FLG_OF_NOWARN 0x00000100 /* disable symbol warnings: -t */
422 #define FLG_OF_NOUNDEF 0x00000200 /* allow no undefined symbols: -zdefs */
423 #define FLG_OF_PURETXT 0x00000400 /* allow no text relocations: -ztext */
424 #define FLG_OF_GENMAP 0x00000800 /* generate a memory map: -m */
425 #define FLG_OF_DYNLIBS 0x00001000 /* dynamic input allowed: -Bdynamic */
426 #define FLG_OF_SYMBOLIC 0x00002000 /* bind global symbols: -Bsymbolic */
427 #define FLG_OF_ADDVERS 0x00004000 /* add version stamp: -Qy */

```

```

428 #define FLG_OF_NOLDYNSYM 0x00008000 /* -znoldynsym set */
429 #define FLG_OF_IS_ORDER 0x00010000 /* input section ordering within a */
430 /* segment is required */
431 #define FLG_OF_EC_FILES 0x00020000 /* Ent_desc exist w/non-NULL ec_files */
432 #define FLG_OF_TEXTREL 0x00040000 /* text relocations have been found */
433 #define FLG_OF_MULDEFS 0x00080000 /* multiple symbols are allowed */
434 #define FLG_OF_TLSPHDR 0x00100000 /* a TLS program header is required */
435 #define FLG_OF_BLDGOT 0x00200000 /* build GOT table */
436 #define FLG_OF_VERDEF 0x00400000 /* record version definitions */
437 #define FLG_OF_VERNEED 0x00800000 /* record version dependencies */
438 #define FLG_OF_NOVERSEC 0x01000000 /* don't record version sections */
439 #define FLG_OF_KEY 0x02000000 /* file requires sort keys */
440 #define FLG_OF_PROCREL 0x04000000 /* process any symbol reductions by */
441 /* effecting the symbol table */
442 /* output and relocations */
443 #define FLG_OF_SYMINFO 0x08000000 /* create a syminfo section */
444 #define FLG_OF_AUX 0x10000000 /* ofl_filter is an auxiliary filter */
445 #define FLG_OF_FATAL 0x20000000 /* fatal error during input */
446 #define FLG_OF_WARN 0x40000000 /* warning during input processing. */
447 #define FLG_OF_VERBOSE 0x80000000 /* -z verbose flag set */

449 #define FLG_OF_MAPSYMB 0x000100000000 /* symbolic scope definition seen */
450 #define FLG_OF_MAPGLOB 0x000200000000 /* global scope definition seen */
451 #define FLG_OF_COMREL 0x000400000000 /* -z combreloc set, which enables */
452 /* DT_RELACNT tracking, */
453 #define FLG_OF_NOCOMREL 0x000800000000 /* -z nocombreloc set */
454 #define FLG_OF_AUTOLCL 0x001000000000 /* automatically reduce unspecified */
455 /* global symbols to locals */
456 #define FLG_OF_AUTOELM 0x002000000000 /* automatically eliminate */
457 /* unspecified global symbols */
458 #define FLG_OF_REDLSYM 0x004000000000 /* reduce local symbols */
459 #define FLG_OF_OS_ORDER 0x008000000000 /* output section ordering required */
460 #define FLG_OF_OSABI 0x010000000000 /* tag object as ELFOSABI_SOLARIS */
461 #define FLG_OF_ADJOSCNT 0x020000000000 /* adjust ofl_shdrct to accommodate */
462 /* discarded sections */
463 #define FLG_OF_OTOSCAP 0x040000000000 /* convert object capabilities to */
464 /* symbol capabilities */
465 #define FLG_OF_PTCAP 0x080000000000 /* PT_SUNWCAP required */
466 #define FLG_OF_CAPSTRS 0x100000000000 /* capability strings are required */
467 #define FLG_OF_EHFRAME 0x200000000000 /* output contains .eh_frame section */
468 #define FLG_OF_FATWARN 0x400000000000 /* make warnings fatal */
469 #define FLG_OF_ADEFLTB 0x800000000000 /* no libraries in default path */

471 /*
472 * In the flags1 arena, establish any options that are applicable to archive
473 * extraction first, and associate a mask. These values are recorded with any
474 * archive descriptor so that they may be reset should the archive require a
475 * rescan to try and resolve undefined symbols.
476 */
477 #define FLG_OF1_ALLEXRT 0x0000000001 /* extract all members from an */
478 /* archive file */
479 #define FLG_OF1_WEAKEXT 0x0000000002 /* allow archive extraction to */
480 /* resolve weak references */
481 #define MSK_OF1_ARCHIVE 0x0000000003 /* archive flags mask */

483 #define FLG_OF1_NOINTRP 0x0000000008 /* -z nointerp flag set */
484 #define FLG_OF1_ZDIRECT 0x0000000010 /* -z direct flag set */
485 #define FLG_OF1_NDIRECT 0x0000000020 /* no-direct bindings specified */
486 #define FLG_OF1_DEFERRED 0x0000000040 /* deferred dependency recording */

488 #define FLG_OF1_RELDYN 0x0000000100 /* process .dynamic in rel obj */
489 #define FLG_OF1_NRLXREL 0x0000000200 /* -z norelaxreloc flag set */
490 #define FLG_OF1_RLXREL 0x0000000400 /* -z relaxreloc flag set */
491 #define FLG_OF1_IGNORE 0x0000000800 /* ignore unused dependencies */
492 #define FLG_OF1_NOSGHND 0x0000001000 /* -z nosighandler flag set */
493 #define FLG_OF1_TEXTTOFF 0x0000002000 /* text relocations are ok */

```

```

494 #define FLG_OF1_ABSEEXEC 0x0000004000 /* -zabsexec set */
495 #define FLG_OF1_LAZYLD 0x0000008000 /* lazy loading of objects enabled */
496 #define FLG_OF1_GRPPRM 0x0000010000 /* dependencies are to have */
497 /* GROUPEM enabled */

499 #define FLG_OF1_NOPARTI 0x0000040000 /* -znopartial set */
500 #define FLG_OF1_BSSOREL 0x0000080000 /* output relocation against bss */
501 /* section */
502 #define FLG_OF1_TLSOREL 0x0000100000 /* output relocation against .tlbss */
503 /* section */
504 #define FLG_OF1_MEMORY 0x0000200000 /* produce a memory model */
505 #define FLG_OF1_NGLBDIR 0x0000400000 /* no DT_1_DIRECT flag allowed */
506 #define FLG_OF1_ENCDIFF 0x0000800000 /* host running linker has different */
507 /* byte order than output object */
508 #define FLG_OF1_VADDR 0x0001000000 /* a segment defines explicit vaddr */
509 #define FLG_OF1_EXTRACT 0x0002000000 /* archive member has been extracted */
510 #define FLG_OF1_RESCAN 0x0004000000 /* any archives should be rescanned */
511 #define FLG_OF1_IGNPRC 0x0008000000 /* ignore processing required */
512 #define FLG_OF1_NCSTTAB 0x0010000000 /* -znoompstrtab set */
513 #define FLG_OF1_DONE 0x0020000000 /* link-editor processing complete */
514 #define FLG_OF1_NONREG 0x0040000000 /* non-regular file specified as */
515 /* the output file */
516 #define FLG_OF1_ALNODIR 0x0080000000 /* establish NODIRECT for all */
517 /* exported interfaces. */
518 #define FLG_OF1_OVHWCAP1 0x0100000000 /* override CA_SUNW_HW_1 capabilities */
519 #define FLG_OF1_OVSFCAP1 0x0200000000 /* override CA_SUNW_SF_1 capabilities */
520 #define FLG_OF1_OVHWCAP2 0x0400000000 /* override CA_SUNW_HW_2 capabilities */
521 #define FLG_OF1_OVMACHCAP 0x0800000000 /* override CA_SUNW_MACH capability */
522 #define FLG_OF1_OVPLATCAP 0x1000000000 /* override CA_SUNW_PLAT capability */
523 #define FLG_OF1_OVIDCAP 0x2000000000 /* override CA_SUNW_ID capability */

525 /*
526 * Guidance flags. The flags with the FLG_OFG_NO_ prefix are used to suppress
527 * messages for a given category, and use the lower 28 bits of the word,
528 * The upper nibble is reserved for other guidance status.
529 */
530 #define FLG_OFG_ENABLE 0x10000000 /* -z guidance option active */
531 #define FLG_OFG_ISSUED 0x20000000 /* -z guidance message issued */

533 #define FLG_OFG_NO_ALL 0x0fffffff /* disable all guidance */
534 #define FLG_OFG_NO_DEFS 0x00000001 /* specify all dependencies */
535 #define FLG_OFG_NO_DB 0x00000002 /* use direct bindings */
536 #define FLG_OFG_NO_LAZY 0x00000004 /* be explicit about lazyload */
537 #define FLG_OFG_NO_MF 0x00000008 /* use v2 mapfile syntax */
538 #define FLG_OFG_NO_TEXT 0x00000010 /* verify pure text segment */
539 #define FLG_OFG_NO_UNUSED 0x00000020 /* remove unused dependency */

541 /*
542 * Test to see if a guidance should be given for a given category
543 * or not. _no_flag is one of the FLG_OFG_NO_xxx flags. Returns TRUE
544 * if the guidance should be issued, and FALSE to remain silent.
545 */
546 #define OF1_GUIDANCE(_of1, _no_flag) (((_of1)->of1_guideflags & \
547 (FLG_OFG_ENABLE | (_no_flag))) == FLG_OFG_ENABLE)

549 /*
550 * Test to see if the output file would allow the presence of
551 * a .dynsym section.
552 */
553 #define OF1_ALLOW_DYNSYM(_of1) (((_of1)->of1_flags & \
554 (FLG_OF_DYNAMIC | FLG_OF_RELOBJ)) == FLG_OF_DYNAMIC)

556 /*
557 * Test to see if the output file would allow the presence of
558 * a .SUNW_ldynsym section. The requirements are that a .dynsym
559 * is allowed, and -znoldynsym has not been specified. Note that

```

```

560 * even if the answer is True (1), we will only generate one if there
561 * are local symbols that require it.
562 */
563 #define OF1_ALLOW_LDYSYM(_of1) (((_of1)->of1_flags & \
564 (FLG_OF_DYNAMIC | FLG_OF_RELOBJ | FLG_OF_NOLDYNSYM)) == FLG_OF_DYNAMIC)

566 /*
567 * Test to see if relocation processing should be done. This is normally
568 * true, but can be disabled via the '-z noreloc' option. Note that
569 * relocatable objects are still relocated even if '-z noreloc' is present.
570 */
571 #define OF1_DO_RELOC(_of1) (((_of1)->of1_flags & FLG_OF_RELOBJ) || \
572 !((_of1)->of1_dtflags_1 & DF_1_NORELOC))

574 /*
575 * Determine whether a static executable is being built.
576 */
577 #define OF1_IS_STATIC_EXEC(_of1) (((_of1)->of1_flags & \
578 (FLG_OF_STATIC | FLG_OF_EXEC)) == (FLG_OF_STATIC | FLG_OF_EXEC))

580 /*
581 * Determine whether a static object is being built. This macro is used
582 * to select the appropriate string table, and symbol table that other
583 * sections need to reference.
584 */
585 #define OF1_IS_STATIC_OBJ(_of1) (((_of1)->of1_flags & \
586 (FLG_OF_RELOBJ | FLG_OF_STATIC))

588 /*
589 * Macros for counting symbol table entries. These are used to size symbol
590 * tables and associated sections (.syminfo, SUNW_capinfo, .hash, etc.) and
591 * set required sh_info entries (the offset to the first global symbol).
592 */
593 #define SYMTAB_LOC_CNT(_of1) /* local .symtab entries */ \
594 (2 + /* NULL and ST_FILE */ \
595 (_of1)->of1_shdrct + /* section symbol */ \
596 (_of1)->of1_caplocnt + /* local capabilities */ \
597 (_of1)->of1_scopecnt + /* scoped symbols */ \
598 (_of1)->of1_locscnt) /* standard locals */
599 #define SYMTAB_ALL_CNT(_of1) /* all .symtab entries */ \
600 (SYMTAB_LOC_CNT(_of1) + /* .symtab locals */ \
601 (_of1)->of1_globcnt) /* standard globals */

603 #define DYNSYM_LOC_CNT(_of1) /* local .dynsym entries */ \
604 (1 + /* NULL */ \
605 (_of1)->of1_dynshdrct + /* section symbols */ \
606 (_of1)->of1_caplocnt + /* local capabilities */ \
607 (_of1)->of1_lregsymcnt) /* local register symbols */
608 #define DYNSYM_ALL_CNT(_of1) /* all .dynsym entries */ \
609 (DYNSYM_LOC_CNT(_of1) + /* .dynsym locals */ \
610 (_of1)->of1_globcnt) /* standard globals */

612 /*
613 * Define a move descriptor used within relocation structures.
614 */
615 typedef struct {
616 Move *mr_move;
617 Sym_desc *mr_sym;
618 } Mv_reloc;

620 /*
621 * Relocation (active & output) processing structure - transparent to common
622 * code. There can be millions of these structures in a large link, so it
623 * is important to keep it small. You should only add new items to Rel_desc
624 * if they are critical, apply to most relocations, and cannot be easily
625 * computed from the other information.

```

```

626 *
627 * Items that can be derived should be implemented as a function that accepts
628 * a Rel_desc argument, and returns the desired data. ld_reloc_sym_name() is
629 * an example of this.
630 *
631 * Lesser used relocation data is kept in an auxiliary block, Rel_aux,
632 * that is only allocated as necessary. In exchange for adding one pointer
633 * of overhead to Rel_desc (rel_aux), most relocations are reduced in size
634 * by the size of Rel_aux. This strategy relies on the data in Rel_aux
635 * being rarely needed --- otherwise it will backfire badly.
636 *
637 * Note that rel_raddend is primarily only of interest to RELA relocations,
638 * and is set to 0 for REL. However, there is an exception: If FLG_REL_NADDEND
639 * is set, then rel_raddend contains a replacement value for the implicit
640 * addend found in the relocation target.
641 *
642 * Fields should be ordered from largest to smallest, to minimize packing
643 * holes in the struct layout.
644 */
645 struct rel_desc {
646     Is_desc      *rel_isdesc; /* input section reloc is against */
647     Sym_desc     *rel_sym;    /* sym relocation is against */
648     Rel_aux      *rel_aux;    /* NULL, or auxiliary data */
649     Xword        rel_offset; /* relocation offset */
650     Sxword       rel_raddend; /* addend from input relocation */
651     Word         rel_flags;   /* misc. flags for relocations */
652     Word         rel_rtype;   /* relocation type */
653 };
654
655 /*
656 * Data that would be kept in Rel_desc if the size of that structure was
657 * not an issue. This auxiliary block is only allocated as needed,
658 * and must only contain rarely needed items. The goal is for the vast
659 * majority of Rel_desc structs to not have an auxiliary block.
660 *
661 * When a Rel_desc does not have an auxiliary block, a default value
662 * is assumed for each auxiliary item:
663 *
664 * - ra_osdesc:
665 *   Output section to which relocation applies. The default
666 *   value for this is the output section associated with the
667 *   input section (rel_isdesc->is_osdesc), or NULL if there
668 *   is no associated input section.
669 *
670 * - ra_usym:
671 *   If the symbol associated with a relocation is part of a weak/strong
672 *   pair, then ra_usym contains the strong symbol and rel_sym the weak.
673 *   Otherwise, the default value is the same value as rel_sym.
674 *
675 * - ra_move:
676 *   Move table data. The default value is NULL.
677 *
678 * - ra_typedata:
679 *   ELF_R_TYPE_DATA(info). This value applies only to a small
680 *   subset of 64-bit sparc relocations, and is otherwise 0. The
681 *   default value is 0.
682 *
683 * If any value in Rel_aux is non-default, then an auxiliary block is
684 * necessary, and each field contains its actual value. If all the auxiliary
685 * values are default, no Rel_aux is needed, and the RELAUX_GET_xxx()
686 * macros below are able to supply the proper default.
687 *
688 * To set a Rel_aux value, use the ld_reloc_set_aux_xxx() functions.
689 * These functions are written to avoid unnecessary auxiliary allocations,
690 * and know the rules for each item.
691 */

```

```

692 struct rel_aux {
693     Os_desc      *ra_osdesc; /* output section reloc is against */
694     Sym_desc     *ra_usym;   /* strong sym if this is a weak pair */
695     Mv_reloc     *ra_move;   /* move table information */
696     Word         ra_typedata; /* ELF_R_TYPE_DATA(info) */
697 };
698
699 /*
700 * Test a given auxiliary value to determine if it has the default value
701 * for that item, as described above. If all the auxiliary items have
702 * their default values, no auxiliary place is necessary to represent them.
703 * If any one of them is non-default, the auxiliary block is needed.
704 */
705 #define RELAUX_ISDEFAULT_MOVE(_rdesc, _mv) (_mv == NULL)
706 #define RELAUX_ISDEFAULT_USYM(_rdesc, _usym) ((_rdesc)->rel_sym == _usym)
707 #define RELAUX_ISDEFAULT_OSDESC(_rdesc, _osdesc) \
708     (((_rdesc)->rel_isdesc == NULL) && (_osdesc == NULL)) || \
709     ((_rdesc)->rel_isdesc && ((_rdesc)->rel_isdesc->is_osdesc == _osdesc))
710 #define RELAUX_ISDEFAULT_TYPEDATA(_rdesc, _typedata) (_typedata == 0)
711
712 /*
713 * Retrieve the value of an auxiliary relocation item, preserving the illusion
714 * that every relocation descriptor has an auxiliary block attached. The
715 * real implementation is that an auxiliary block is only present if one or
716 * more auxiliary items have non-default values. These macros return the true
717 * value if an auxiliary block is present, and the default value for the
718 * item otherwise.
719 */
720 #define RELAUX_GET_MOVE(_rdesc) \
721     ((_rdesc)->rel_aux ? (_rdesc)->rel_aux->ra_move : NULL)
722 #define RELAUX_GET_USYM(_rdesc) \
723     ((_rdesc)->rel_aux ? (_rdesc)->rel_aux->ra_usym : (_rdesc)->rel_sym)
724 #define RELAUX_GET_OSDESC(_rdesc) \
725     ((_rdesc)->rel_aux ? (_rdesc)->rel_aux->ra_osdesc : \
726     ((_rdesc)->rel_isdesc ? (_rdesc)->rel_isdesc->is_osdesc : NULL))
727 #define RELAUX_GET_TYPEDATA(_rdesc) \
728     ((_rdesc)->rel_aux ? (_rdesc)->rel_aux->ra_typedata : 0)
729
730 /*
731 * common flags used on the Rel_desc structure (defined in machrel.h).
732 */
733 #define FLG_REL_GOT      0x00000001 /* relocation against GOT */
734 #define FLG_REL_PLT     0x00000002 /* relocation against PLT */
735 #define FLG_REL_BSS     0x00000004 /* relocation against BSS */
736 #define FLG_REL_LOAD    0x00000008 /* section loadable */
737 #define FLG_REL_SCNNDX  0x00000010 /* use section index for symbol ndx */
738 #define FLG_REL_CLVAL   0x00000020 /* clear VALUE for active relocation */
739 #define FLG_REL_ADVAL   0x00000040 /* add VALUE for output relocation, */
740 /* only relevant to SPARC and */
741 /* R_SPARC_RELATIVE */
742 #define FLG_REL_GOTCL   0x00000080 /* clear the GOT entry. This is */
743 /* relevant to RELA relocations, */
744 /* not REL (i386) relocations */
745 #define FLG_REL_MOVETAB 0x00000100 /* Relocation against .SUNW.move */
746 /* adjustments required before */
747 /* actual relocation */
748 #define FLG_REL_NOINFO  0x00000200 /* Relocation comes from a section */
749 /* with a null sh_info field */
750 #define FLG_REL_REG     0x00000400 /* Relocation target is reg sym */
751 #define FLG_REL_FPTR   0x00000800 /* relocation against func. desc. */
752 #define FLG_REL_RFPTR1  0x00001000 /* Relative relocation against */
753 /* 1st part of FD */
754 #define FLG_REL_RFPTR2  0x00002000 /* Relative relocation against */
755 /* 2nd part of FD */
756 #define FLG_REL_DISP    0x00004000 /* *disp* relocation */
757 #define FLG_REL_STLS    0x00008000 /* IE TLS reference to */

```



```

758 /* static TLS GOT index */
759 #define FLG_REL_DTLS 0x00010000 /* GD TLS reference relative to */
760 /* dynamic TLS GOT index */
761 #define FLG_REL_MTLS 0x00020000 /* LD TLS reference against GOT */
762 #define FLG_REL_STTLS 0x00040000 /* LE TLS reference directly */
763 /* to static tls index */
764 #define FLG_REL_TLSTFIX 0x00080000 /* relocation points to TLS instr. */
765 /* which needs updating */
766 #define FLG_REL_RELA 0x00100000 /* descriptor captures a Rela */
767 #define FLG_REL_GOTFIX 0x00200000 /* relocation points to GOTOP instr. */
768 /* which needs updating */
769 #define FLG_REL_NADDEND 0x00400000 /* Replace implicit addend in dest */
770 /* with value in rel_raddend */
771 /* Relevant to REL (i386) */
772 /* relocations, not to RELA. */

774 /*
775 * We often need the name of the symbol contained in a relocation descriptor
776 * for diagnostic or error output. This is usually the symbol name, but
777 * we substitute a constructed name in some cases. Hence, the name is
778 * generated on the fly by a private function within libld. This is the
779 * prototype for that function.
780 */
781 typedef const char *(* rel_desc_sname_func_t)(Rel_desc *);

783 /*
784 * Header for a relocation descriptor cache buffer.
785 */
786 struct rel_cachebuf {
787     Rel_desc *rc_end;
788     Rel_desc *rc_free;
789     Rel_desc rc_arr[1];
790 };

792 /*
793 * Header for a relocation auxiliary descriptor cache buffer.
794 */
795 struct rel_aux_cachebuf {
796     Rel_aux *rac_end;
797     Rel_aux *rac_free;
798     Rel_aux rac_arr[1];
799 };

801 /*
802 * Convenience macro for traversing every relocation descriptor found within
803 * a given relocation cache, transparently handling the cache buffers and
804 * skipping any unallocated descriptors within the buffers.
805 *
806 * entry:
807 *   _rel_cache - Relocate descriptor cache (Rel_cache) to traverse
808 *   _idx - Aliste index variable for use by the macro
809 *   _rcbp - Cache buffer pointer, for use by the macro
810 *   _orsp - Rel_desc pointer, which will take on the value of a different
811 *   relocation descriptor in the cache in each iteration.
812 *
813 * The caller must not assign new values to _idx, _rcbp, or _orsp within
814 * the scope of REL_CACHE_TRAVERSE.
815 */
816 #define REL_CACHE_TRAVERSE(_rel_cache, _idx, _rcbp, _orsp) \
817     for (APLIST_TRAVERSE((_rel_cache)->rc_list, _idx, _rcbp)) \
818         for (_orsp = _rcbp->rc_arr; _orsp < _rcbp->rc_free; _orsp++)

820 /*
821 * Symbol value descriptor. For relocatable objects, each symbols value is
822 * its offset within its associated section. Therefore, to uniquely define
823 * each symbol within a relocatable object, record and sort the sh_offset and

```

```

824 * symbol value. This information is used to search for displacement
825 * relocations as part of copy relocation validation.
826 */
827 typedef struct {
828     Addr ssv_value;
829     Sym_desc *ssv_sdp;
830 } Ssv_desc;

832 /*
833 * Input file processing structures.
834 */
835 struct ifl_desc { /* input file descriptor */
836     const char *ifl_name; /* full file name */
837     const char *ifl_soname; /* shared object name */
838     dev_t ifl_stdev; /* device id and inode number for .so */
839     ino_t ifl_stino; /* multiple inclusion checks */
840     Ehdr *ifl_ehdr; /* elf header describing this file */
841     Elf *ifl_elf; /* elf descriptor for this file */
842     Sym_desc *ifl_oldndx; /* original symbol table indices */
843     Sym_desc *ifl_locs; /* symbol desc version of locals */
844     Ssv_desc *ifl_sortsyms; /* sorted list of symbols by value */
845     Word ifl_locscnt; /* no. of local symbols to process */
846     Word ifl_symscnt; /* total no. of symbols to process */
847     Word ifl_sortcnt; /* no. of sorted symbols to process */
848     Word ifl_shnum; /* number of sections in file */
849     Word ifl_shstrndx; /* index to .shstrtab */
850     Word ifl_vercnt; /* number of versions in file */
851     Half ifl_neededndx; /* index to NEEDED in .dyn section */
852     Word ifl_flags; /* explicit/implicit reference */
853     Is_desc *ifl_isdesc; /* isdesc[scn ndx] = Is_desc ptr */
854     Sdf_desc *ifl_sdfdesc; /* control definition */
855     Versym *ifl_versym; /* version symbol table array */
856     Ver_index *ifl_verndx; /* verndx[ver ndx] = Ver_index */
857     Aplist *ifl_verdesc; /* version descriptor list */
858     Aplist *ifl_relsect; /* relocation section list */
859     Alist *ifl_groups; /* SHT_GROUP section list */
860     Cap_desc *ifl_caps; /* capabilities descriptor */
861 };

863 #define FLG_IF_CMDLINE 0x00000001 /* full filename specified from the */
864 /* command line (no -l) */
865 #define FLG_IF_NEEDED 0x00000002 /* shared object should be recorded */
866 #define FLG_IF_DIRECT 0x00000004 /* establish direct bindings to this */
867 /* object */
868 #define FLG_IF_EXTRACT 0x00000008 /* file extracted from an archive */
869 #define FLG_IF_VERNEED 0x00000010 /* version dependency information is */
870 /* required */
871 #define FLG_IF_DEPREQD 0x00000020 /* dependency is required to satisfy */
872 /* symbol references */
873 #define FLG_IF_NEEDSTR 0x00000040 /* dependency specified by -Nn */
874 /* flag */
875 #define FLG_IF_IGNORE 0x00000080 /* ignore unused dependencies */
876 #define FLG_IF_NODIRECT 0x00000100 /* object contains symbols that */
877 /* cannot be directly bound to */
878 #define FLG_IF_LAZYLD 0x00000200 /* dependency should be lazy loaded */
879 #define FLG_IF_GRPERR 0x00000400 /* dependency establishes a group */
880 #define FLG_IF_DISPPEND 0x00000800 /* displacement relocation done */
881 /* in the ld time. */
882 #define FLG_IF_DISPDONE 0x00001000 /* displacement relocation done */
883 /* at the run time */
884 #define FLG_IF_MAPFILE 0x00002000 /* file is a mapfile */
885 #define FLG_IF_HSTRTAB 0x00004000 /* file has a string section */
886 #define FLG_IF_FILEREF 0x00008000 /* file contains a section which */
887 /* is included in the output */
888 /* allocatable image */
889 #define FLG_IF_GNUVER 0x00010000 /* file used GNU-style versioning */

```

```

890 #define FLG_IF_ORDERED 0x00020000 /* ordered section processing */
891 /* required */
892 #define FLG_IF_OTOSCAP 0x00040000 /* convert object capabilities to */
893 /* symbol capabilities */
894 #define FLG_IF_DEFERRED 0x00080000 /* dependency is deferred */
895 #define FLG_IF_RTLDINF 0x00100000 /* dependency has DT_SUNW_RTLINF set */
896 #define FLG_IF_GROUPS 0x00200000 /* input file has groups to process */

898 /*
899 * Symbol states that require the generation of a DT_POSFLAG_1 .dynamic entry.
900 */
901 #define MSK_IF_POSFLAG1 (FLG_IF_LAZYLD | FLG_IF_GRPPRM | FLG_IF_DEFERRED)

903 /*
904 * Symbol states that require an associated Syminfo entry.
905 */
906 #define MSK_IF_SYMINFO (FLG_IF_LAZYLD | FLG_IF_DIRECT | FLG_IF_DEFERRED)

909 struct is_desc {
910     const char    *is_name;      /* original section name */
911     const char    *is_sym_name;  /* NULL, or name string to use for */
912     /* related STT_SECTION symbols */
913     Shdr          *is_shdr;      /* the elf section header */
914     Ifl_desc      *is_file;      /* infile desc for this section */
915     Os_desc       *is_osdesc;    /* new output section for this */
916     /* input section */
917     Elf_Data      *is_indata;    /* input sections raw data */
918     Is_desc       *is_symshndx;  /* related SHT_SYM_SHNDX section */
919     Is_desc       *is_comdatkeep; /* If COMDAT section is discarded, */
920     /* this is section that was kept */
921     Word          is_scnndx;     /* original section index in file */
922     Word          is_ordndx;     /* index for section. Used to decide */
923     /* where to insert section when */
924     /* reordering sections */
925     Word          is_keyident;   /* key for SHF_{ORDERED|LINK_ORDER} */
926     /* processing and ident used for */
927     /* placing/ordering sections */
928     Word          is_flags;     /* Various flags */
929 };

931 #define FLG_IS_ORDERED 0x0001 /* this is a SHF_ORDERED section */
932 #define FLG_IS_KEY 0x0002 /* section requires sort keys */
933 #define FLG_IS_DISCARD 0x0004 /* section is to be discarded */
934 #define FLG_IS_RELUPD 0x0008 /* symbol defined here may have moved */
935 #define FLG_IS_SECTREF 0x0010 /* section has been referenced */
936 #define FLG_IS_GDATADEF 0x0020 /* section contains global data sym */
937 #define FLG_IS_EXTERNAL 0x0040 /* isp from a user file */
938 #define FLG_IS_INSTRMRG 0x0080 /* Usable SHF_MERGE|SHF_STRINGS sec */
939 #define FLG_IS_GNSTRMRG 0x0100 /* Generated mergeable string section */

941 #define FLG_IS_PLACE 0x0400 /* section requires to be placed */
942 #define FLG_IS_COMDAT 0x0800 /* section is COMDAT */
943 #define FLG_IS_EHFRAME 0x1000 /* section is .eh_frame */

945 /*
946 * Output sections contain lists of input sections that are assigned to them.
947 * These items fall into 4 categories:
948 * BEFORE - Ordered sections that specify SHN_BEFORE, in input order.
949 * ORDERED - Ordered sections that are sorted using unsorted sections
950 * as the sort key.
951 * DEFAULT - Sections that are placed into the output section
952 * in input order.
953 * AFTER - Ordered sections that specify SHN_AFTER, in input order.
954 */
955 #define OS_ISD_BEFORE 0

```

```

956 #define OS_ISD_ORDERED 1
957 #define OS_ISD_DEFAULT 2
958 #define OS_ISD_AFTER 3
959 #define OS_ISD_NUM 4
960 typedef APLIST *os_isdecs_arr[OS_ISD_NUM];

962 /*
963 * Convenience macro for traversing every input section associated
964 * with a given output section. The primary benefit of this macro
965 * is that it preserves a precious level of code indentation in the
966 * code that uses it.
967 */
968 #define OS_ISDESCS_TRAVERSE(_list_idx, _osp, _idx, _isp) \
969     for (_list_idx = 0; _list_idx < OS_ISD_NUM; _list_idx++) \
970         for (APLIST_TRAVERSE(_osp->os_isdecs[_list_idx], _idx, _isp))

973 /*
974 * Map file and output file processing structures
975 */
976 struct os_desc {
977     const char    *os_name;      /* Output section descriptor */
978     /* the section name */
979     Elf_Scn       *os_scn;       /* the elf section descriptor */
980     Shdr          *os_shdr;     /* the elf section header */
981     Os_desc       *os_relo_desc; /* the output relocation section */
982     APLIST        *os_relisdecs; /* reloc input section descriptors */
983     /* for this output section */
984     os_isdecs_arr os_isdecs;    /* lists of input sections in output */
985     APLIST        *os_mstrisdecs; /* FLG_IS_INSTRMRG input sections */
986     Sg_desc       *os_sgdesc;   /* segment os_desc is placed on */
987     Elf_Data      *os_outdata;  /* output sections raw data */
988     avl_tree_t    *os_comdats;  /* AVL tree of COMDAT input sections */
989     /* associated to output section */
990     Word          os_identndx;   /* section identifier for input */
991     /* section processing, followed */
992     /* by section symbol index */
993     Word          os_ordndx;    /* index for section. Used to decide */
994     /* where to insert section when */
995     /* reordering sections */
996     Xword         os_szoutrels; /* size of output relocation section */
997     uint_t        os_namehash; /* hash on section name */
998     uchar_t       os_flags;    /* various flags */
999 };

1000 #define FLG_OS_KEY 0x01 /* section requires sort keys */
1001 #define FLG_OS_OUTREL 0x02 /* output rel against this section */
1002 #define FLG_OS_SECTREF 0x04 /* isps are not affected by -zignore */
1003 #define FLG_OS_EHFRAME 0x08 /* section is .eh_frame */

1005 /*
1006 * The sg_id field of the segment descriptor is used to establish the default
1007 * order for program headers and segments in the output object. Segments are
1008 * ordered according to the following SGID values that classify them based on
1009 * their attributes. The initial set of built in segments are in this order,
1010 * and new mapfile defined segments are inserted into these groups. Within a
1011 * given SGID group, the position of new segments depends on the syntax
1012 * version of the mapfile that creates them. Version 1 (original sysv)
1013 * mapfiles place the new segment at the head of their group (reverse creation
1014 * order). The newer syntax places them at the end, following the others
1015 * (creation order).
1016 *
1017 * Note that any new segments must always be added after PT_PHDR and
1018 * PT_INTERP (refer Generic ABI, Page 5-4).
1019 */
1020 #define SGID_PHDR 0 /* PT_PHDR */
1021 #define SGID_INTERP 1 /* PT_INTERP */

```

```

1022 #define SGID_SUNWCAP 2 /* PT_SUNWCAP */
1023 #define SGID_TEXT 3 /* PT_LOAD */
1024 #define SGID_DATA 4 /* PT_LOAD */
1025 #define SGID_BSS 5 /* PT_LOAD */
1026 #if defined(ELF64)
1027 #define SGID_LRODATA 6 /* PT_LOAD (amd64-only) */
1028 #define SGID_LDATA 7 /* PT_LOAD (amd64-only) */
1029 #endif
1030 #define SGID_TEXT_EMPTY 8 /* PT_LOAD, reserved (?E in version 1 syntax) */
1031 #define SGID_NULL_EMPTY 9 /* PT_NULL, reserved (?E in version 1 syntax) */
1032 #define SGID_DYN 10 /* PT_DYNAMIC */
1033 #define SGID_DTRACE 11 /* PT_SUNWDTRACE */
1034 #define SGID_TLS 12 /* PT_TLS */
1035 #define SGID_UNWIND 13 /* PT_SUNWUNWIND */
1036 #define SGID_SUNWSTACK 14 /* PT_SUNWSTACK */
1037 #define SGID_NOTE 15 /* PT_NOTE */
1038 #define SGID_NULL 16 /* PT_NULL, mapfile defined empty phdr slots */
1039 /* for use by post processors */
1040 #define SGID_EXTRA 17 /* PT_NULL (final catchall) */

1042 typedef Half sg_flags_t;
1043 struct sg_desc {
1044     Word sg_id; /* output segment descriptor */
1045     Phdr sg_phdr; /* segment identifier (for sorting) */
1046     const char *sg_name; /* segment header for output file */
1047     /* segment name for PT_LOAD, PT_NOTE, */
1048     /* and PT_NULL, otherwise NULL */
1049     Xword sg_round; /* data rounding required (mapfile) */
1050     Xword sg_length; /* maximum segment length; if 0 */
1051     /* segment is not specified */
1052     Aplist *sg_osdescs; /* list of output section descriptors */
1053     Aplist *sg_is_order; /* list of entry criteria */
1054     /* giving input section order */
1055     Alist *sg_os_order; /* list specifying output section */
1056     /* ordering for the segment */
1057     sg_flags_t sg_flags;
1058     Aplist *sg_sizesym; /* size symbols for this segment */
1059     Xword sg_align; /* LCM of sh_addralign */
1060     Elf_Scn *sg_fscn; /* the SCN of the first section. */
1061     avl_node_t sg_avlnode; /* AVL book-keeping */
1062 };

1063 #define FLG_SG_P_VADDR 0x0001 /* p_vaddr segment attribute set */
1064 #define FLG_SG_P_PADDR 0x0002 /* p_paddr segment attribute set */
1065 #define FLG_SG_LENGTH 0x0004 /* length segment attribute set */
1066 #define FLG_SG_P_ALIGN 0x0008 /* p_align segment attribute set */
1067 #define FLG_SG_ROUND 0x0010 /* round segment attribute set */
1068 #define FLG_SG_P_FLAGS 0x0020 /* p_flags segment attribute set */
1069 #define FLG_SG_P_TYPE 0x0040 /* p_type segment attribute set */
1070 #define FLG_SG_IS_ORDER 0x0080 /* input section ordering is required */
1071 /* for this segment. */
1072 #define FLG_SG_NOHDR 0x0100 /* don't map ELF or phdrs into */
1073 /* this segment */
1074 #define FLG_SG_EMPTY 0x0200 /* an empty segment specification */
1075 /* no input sections will be */
1076 /* associated to this section */
1077 #define FLG_SG_KEY 0x0400 /* segment requires sort keys */
1078 #define FLG_SG_NODISABLE 0x0800 /* FLG_SG_DISABLED is not allowed on */
1079 /* this segment */
1080 #define FLG_SG_DISABLED 0x1000 /* this segment is disabled */
1081 #define FLG_SG_PHREQ 0x2000 /* this segment requires a program */
1082 /* header */
1083 #define FLG_SG_ORDERED 0x4000 /* SEGMENT_ORDER segment */

1085 struct sec_order {
1086     const char *sco_secname; /* section name to be ordered */
1087     Half sco_flags;

```

```

1088 };
1090 #define FLG_SGO_USED 0x0001 /* was ordering used? */

1092 typedef Half ec_flags_t;
1093 struct ent_desc {
1094     const char *ec_name; /* input section entrance criteria */
1095     Alist *ec_files; /* entrance criteria name, or NULL */
1096     /* files from which to accept */
1097     /* sections */
1098     const char *ec_is_name; /* input section name to match */
1099     /* (NULL if none) */
1100     Word ec_type; /* section type */
1101     Word ec_attrmask; /* section attribute mask (AWX) */
1102     Word ec_attrbits; /* sections attribute bits */
1103     Sg_desc *ec_segment; /* output segment to enter if matched */
1104     Word ec_ordndx; /* index to determine where section */
1105     /* meeting this criteria should */
1106     /* be inserted. Used for reordering */
1107     /* of sections. */
1108     ec_flags_t ec_flags;
1109     avl_node_t ec_avlnode; /* AVL book-keeping */
1110 };

1111 #define FLG_EC_BUILTIN 0x0001 /* built in descriptor */
1112 #define FLG_EC_USED 0x0002 /* entrance criteria met? */
1113 #define FLG_EC_CATCHALL 0x0004 /* Catches any section */

1115 /*
1116 * Ent_desc_file is the type of element maintained in the ec_files Alist
1117 * of an entrance criteria descriptor. Each item maintains one file
1118 * path, and a set of flags that specify the type of comparison it implies,
1119 * and other information about it. The comparison type is maintained in
1120 * the bottom byte of the flags.
1121 */
1122 #define TYP_ECF_MASK 0x00ff /* Comparison type mask */
1123 #define TYP_ECF_PATH 0 /* Compare to file path */
1124 #define TYP_ECF_BASENAME 1 /* Compare to file basename */
1125 #define TYP_ECF_OBJNAME 2 /* Compare to regular file basename, */
1126 /* or to archive member name */
1127 #define TYP_ECF_NUM 3

1129 #define FLG_ECF_ARMEMBER 0x0100 /* name includes archive member */

1131 typedef struct {
1132     Word edf_flags; /* Type of comparison */
1133     const char *edf_name; /* String to compare to */
1134     size_t edf_name_len; /* strlen(edf_name) */
1135 } Ent_desc_file;

1137 /*
1138 * One structure is allocated for a move entry, and associated to the symbol
1139 * against which a move is targeted.
1140 */
1141 typedef struct {
1142     Move *md_move; /* original Move entry */
1143     Xword md_start; /* start position */
1144     Xword md_len; /* length of initialization */
1145     Word md_oidx; /* output Move entry index */
1146 } Mv_desc;

1148 /*
1149 * Symbol descriptor.
1150 */
1151 typedef Lword sd_flag_t;
1152 struct sym_desc {
1153     Alist *sd_GOTndx; /* list of associated GOT entries */

```

```

1154     Sym      *sd_sym;      /* pointer to symbol table entry */
1155     Sym      *sd_osym;     /* copy of the original symbol entry */
1156     /*      used only for local partial */
1157     Alist    *sd_move;     /* move information associated with a */
1158     /*      partially initialized symbol */
1159     const char *sd_name;   /* symbols name */
1160     Ifl_desc *sd_file;    /* file where symbol is taken */
1161     Is_desc  *sd_isc;     /* input section of symbol definition */
1162     Sym_aux  *sd_aux;     /* auxiliary global symbol info. */
1163     Word     sd_symndx;   /* index in output symbol table */
1164     Word     sd_shndx;    /* sect. index sym is associated w/ */
1165     sd_flag_t sd_flags;   /* state flags */
1166     Half     sd_ref;     /* reference definition of symbol */
1167 };

1169 /*
1170 * The auxiliary symbol descriptor contains the additional information (beyond
1171 * the symbol descriptor) required to process global symbols. These symbols are
1172 * accessed via an internal symbol hash table where locality of reference is
1173 * important for performance.
1174 */
1175 struct sym_aux {
1176     Aplist   *sa_dfiles;  /* files where symbol is defined */
1177     Sym      sa_sym;     /* copy of symtab entry */
1178     const char *sa_vfile; /* first unavailable definition */
1179     const char *sa_rfile; /* file with first symbol referenced */
1180     Word      sa_hash;   /* the pure hash value of symbol */
1181     Word      sa_PLTndx;  /* index into PLT for symbol */
1182     Word      sa_PLTGOTndx; /* GOT entry indx for PLT indirection */
1183     Word      sa_linkndx; /* index of associated symbol from */
1184     /*      ET_DYN file */
1185     Half      sa_symspec; /* special symbol ids */
1186     Half      sa_overndx; /* output file versioning index */
1187     Half      sa_dverndx; /* dependency versioning index */
1188 };

1190 /*
1191 * Nodes used to track symbols in the global AVL symbol dictionary.
1192 */
1193 struct sym_avlnode {
1194     avl_node_t sav_node; /* AVL node */
1195     Word       sav_hash; /* symbol hash value */
1196     const char *sav_name; /* symbol name */
1197     Sym_desc   sav_sdp;  /* symbol descriptor */
1198 };

1200 /*
1201 * These are the ids for processing of 'Special symbols'. They are used
1202 * to set the sym->sd_aux->sa_symspec field.
1203 */
1204 #define SDAUX_ID_ETEXT 1 /* etext && _etext symbol */
1205 #define SDAUX_ID_EDATA 2 /* edata && _edata symbol */
1206 #define SDAUX_ID_END 3 /* end, _end, && _END symbol */
1207 #define SDAUX_ID_DYN 4 /* DYNAMIC && _DYNAMIC symbol */
1208 #define SDAUX_ID_PLT 5 /* _PROCEDURE LINKAGE TABLE symbol */
1209 #define SDAUX_ID_GOT 6 /* _GLOBAL OFFSET TABLE symbol */
1210 #define SDAUX_ID_START 7 /* START_ && _START_ symbol */

1212 /*
1213 * Flags for sym_desc.sd_flags
1214 */
1215 #define FLG_SY_MVTOCOMM 0x00000001 /* assign symbol to common (.bss) */
1216 /* this is a result of a */
1217 /* copy reloc against sym */
1218 #define FLG_SY_GLOBREF 0x00000002 /* a global reference has been seen */
1219 #define FLG_SY_WEAKDEF 0x00000004 /* a weak definition has been used */

```

```

1220 #define FLG_SY_CLEAN 0x00000008 /* 'Sym' entry points to original */
1221 /* input file (read-only). */
1222 #define FLG_SY_UPREQD 0x00000010 /* symbol value update is required, */
1223 /* either it's used as an entry */
1224 /* point or for relocation, but */
1225 /* it must be updated even if */
1226 /* the -s flag is in effect */
1227 #define FLG_SY_NOTAVAIL 0x00000020 /* symbol is not available to the */
1228 /* application either because it */
1229 /* originates from an implicitly */
1230 /* referenced shared object, or */
1231 /* because it is not part of a */
1232 /* specified version. */
1233 #define FLG_SY_REDUCED 0x00000040 /* a global is reduced to local */
1234 #define FLG_SY_VERSPROM 0x00000080 /* version definition has been */
1235 /* promoted to output file */
1236 #define FLG_SY_PROT 0x00000100 /* stv_protected visibility seen */
1237 #define FLG_SY_MAPREF 0x00000200 /* symbol reference generated by user */
1238 /* from mapfile */
1239 #define FLG_SY_REFRSD 0x00000400 /* symbols sd_ref has been raised */
1240 /* due to a copy-relocs */
1241 /* weak-strong pairing */
1242 #define FLG_SY_INTPOSE 0x00000800 /* symbol defines an interposer */
1243 #define FLG_SY_INVALID 0x00001000 /* unwanted/erroneous symbol */
1244 #define FLG_SY_SMGOT 0x00002000 /* small got index assigned to symbol */
1245 /* sparc only */
1246 #define FLG_SY_PARENT 0x00004000 /* symbol to be found in parent */
1247 /* only used with direct bindings */
1248 #define FLG_SY_LAZYLD 0x00008000 /* symbol to cause lazyloading of */
1249 /* parent object */
1250 #define FLG_SY_ISDISC 0x00010000 /* symbol is a member of a DISCARDED */
1251 /* section (COMDAT) */
1252 #define FLG_SY_PAREXPN 0x00020000 /* partially init. symbol to be */
1253 /* expanded */
1254 #define FLG_SY_PLTPAD 0x00040000 /* pltpadding has been allocated for */
1255 /* this symbol */
1256 #define FLG_SY_REGSYM 0x00080000 /* REGISTER symbol (sparc only) */
1257 #define FLG_SY_SOFOUND 0x00100000 /* compared against an SO definition */
1258 #define FLG_SY_EXTERN 0x00200000 /* symbol is external, allows -zdefs */
1259 /* error suppression */
1260 #define FLG_SY_MAPUSED 0x00400000 /* mapfile symbol used (occurred */
1261 /* within a relocatable object) */
1262 #define FLG_SY_COMMEXP 0x00800000 /* COMMON symbol which has been */
1263 /* allocated */
1264 #define FLG_SY_CMDREF 0x01000000 /* symbol was referenced from the */
1265 /* command line. (ld -u <>, */
1266 /* ld -zrtldinfo=<>, ...) */
1267 #define FLG_SY_SPECSEC 0x02000000 /* section index is reserved value */
1268 /* ABS, COMMON, ... */
1269 #define FLG_SY_TENTSYM 0x04000000 /* tentative symbol */
1270 #define FLG_SY_VISIBLE 0x08000000 /* symbols visibility determined */
1271 /* symbol is a standard filter */
1272 #define FLG_SY_AUXFLTR 0x20000000 /* symbol is an auxiliary filter */
1273 #define FLG_SY_DYNSORT 0x40000000 /* req. in dyn[sym|tls]sort section */
1274 #define FLG_SY_NODYNSORT 0x80000000 /* excluded from dyn[sym|tls]sort sec */

1276 #define FLG_SY_DEFAULT 0x000010000000 /* global symbol, default */
1277 #define FLG_SY_SINGLE 0x000020000000 /* global symbol, singleton defined */
1278 #define FLG_SY_PROTECT 0x000040000000 /* global symbol, protected defined */
1279 #define FLG_SY_EXPORT 0x000080000000 /* global symbol, exported defined */

1281 #define MSK_SY_GLOBAL \
1282     (FLG_SY_DEFAULT | FLG_SY_SINGLE | FLG_SY_PROTECT | FLG_SY_EXPORT)
1283 /* this mask indicates that the */
1284 /* symbol has been explicitly */
1285 /* defined within a mapfile */

```

```

1286          /* definition, and is a candidate */
1287          /* for versioning */

1289 #define FLG_SY_HIDDEN 0x000100000000 /* global symbol, reduce to local */
1290 #define FLG_SY_ELIM 0x000200000000 /* global symbol, eliminate */
1291 #define FLG_SY_IGNORE 0x000400000000 /* global symbol, ignored */

1293 #define MSK_SY_LOCAL (FLG_SY_HIDDEN | FLG_SY_ELIM | FLG_SY_IGNORE)
1294 /* this mask allows all local state */
1295 /* flags to be removed when the */
1296 /* symbol is copy relocated */

1298 #define FLG_SY_EXPDEF 0x000800000000 /* symbol visibility defined */
1299 /* explicitly */

1301 #define MSK_SY_NOAUTO (FLG_SY_SINGLE | FLG_SY_EXPORT | FLG_SY_EXPDEF)
1302 /* this mask indicates that the */
1303 /* symbol is not a candidate for */
1304 /* auto-reduction/elimination */

1306 #define FLG_SY_MAPFILE 0x001000000000 /* symbol attribute defined in a */
1307 /* mapfile */
1308 #define FLG_SY_DIR 0x002000000000 /* global symbol, direct bindings */
1309 #define FLG_SY_NDIR 0x004000000000 /* global symbol, nondirect bindings */
1310 #define FLG_SY_OVERLAP 0x008000000000 /* move entry overlap detected */
1311 #define FLG_SY_CAP 0x010000000000 /* symbol is associated with */
1312 /* capabilities */
1313 #define FLG_SY_DEFERRED 0x020000000000 /* symbol should not be bound to */
1314 /* during BIND_NOW relocations */

1316 /*
1317 * A symbol can only be truly hidden if it is not a capabilities symbol.
1318 */
1319 #define SYM_IS_HIDDEN(_sdp) \
1320     (((_sdp)->sdf_flags & (FLG_SY_HIDDEN | FLG_SY_CAP)) == FLG_SY_HIDDEN)

1322 /*
1323 * Create a mask for (sym.st_other & visibility) since the gABI does not yet
1324 * define a ELF*_ST_OTHER macro.
1325 */
1326 #define MSK_SYM_VISIBILITY 0x7

1328 /*
1329 * Structure to manage the shared object definition lists. There are two lists
1330 * that use this structure:
1331 *
1332 * - ofl_soneed; maintain the list of implicitly required dependencies
1333 * (ie. shared objects needed by other shared objects). These definitions
1334 * may include RPATH's required to locate the dependencies, and any
1335 * version requirements.
1336 *
1337 * - ofl_socnt1; maintains the shared object control definitions. These are
1338 * provided by the user (via a mapfile) and are used to indicate any
1339 * version control requirements.
1340 */
1341 struct sdf_desc {
1342     const char *sdf_name; /* the shared objects file name */
1343     char *sdf_rpath; /* library search path DT_RPATH */
1344     const char *sdf_rfile; /* referencing file for diagnostics */
1345     Ifl_desc *sdf_file; /* the final input file descriptor */
1346     Alist *sdf_vers; /* list of versions that are required */
1347     /* from this object */
1348     Alist *sdf_verneed; /* list of VERNEEDS to create for */
1349     /* object via mapfile ADDVERS */
1350     Word sdf_flags;
1351 };

```

```

1353 #define FLG_SDF_SELECT 0x01 /* version control selection required */
1354 #define FLG_SDF_VERIFY 0x02 /* version definition verification */
1355 /* required */
1356 #define FLG_SDF_ADDVER 0x04 /* add VERNEED references */

1358 /*
1359 * Structure to manage shared object version usage requirements.
1360 */
1361 struct sdv_desc {
1362     const char *sdv_name; /* version name */
1363     const char *sdv_ref; /* versions reference */
1364     Word sdv_flags; /* flags */
1365 };

1367 #define FLG_SDV_MATCHED 0x01 /* VERDEF found and matched */

1369 /*
1370 * Structures to manage versioning information. Two versioning structures are
1371 * defined:
1372 *
1373 * - a version descriptor maintains a linked list of versions and their
1374 * associated dependencies. This is used to build the version definitions
1375 * for an image being created (see map_symbol), and to determine the
1376 * version dependency graph for any input files that are versioned.
1377 *
1378 * - a version index array contains each version of an input file that is
1379 * being processed. It informs us which versions are available for
1380 * binding, and is used to generate any version dependency information.
1381 */
1382 struct ver_desc {
1383     const char *vd_name; /* version name */
1384     Ifl_desc *vd_file; /* file that defined version */
1385     Word vd_hash; /* hash value of name */
1386     Half vd_ndx; /* coordinates with symbol index */
1387     Half vd_flags; /* version information */
1388     Aplist *vd_deps; /* version dependencies */
1389     Ver_desc *vd_ref; /* dependency's first reference */
1390 };

1392 struct ver_index {
1393     const char *vi_name; /* dependency version name */
1394     Half vi_flags; /* communicates availability */
1395     Half vi_overndx; /* index assigned to this version in */
1396     /* output object Verneed section */
1397     Ver_desc *vi_desc; /* cross reference to descriptor */
1398 };

1400 /*
1401 * Define any internal version descriptor flags ([vd|vi]_flags). Note that the
1402 * first byte is reserved for user visible flags (refer VER_FLG's in link.h).
1403 */
1404 #define MSK_VER_USER 0x0f /* mask for user visible flags */

1406 #define FLG_VER_AVAIL 0x10 /* version is available for binding */
1407 #define FLG_VER_REFERER 0x20 /* version has been referenced */
1408 #define FLG_VER_CYCLIC 0x40 /* a member of cyclic dependency */

1410 /*
1411 * isalist(1) descriptor - used to break an isalist string into its component
1412 * options.
1413 */
1414 struct isa_opt {
1415     char *isa_name; /* individual isa option name */
1416     size_t isa_namesz; /* and associated size */
1417 };

```

```

1419 struct isa_desc {
1420     char      *isa_list;      /* sysinfo(SI_ISALIST) list */
1421     size_t    isa_listsz;    /* and associated size */
1422     Isa_opt   *isa_opt;      /* table of individual isa options */
1423     size_t    isa_optno;     /* and associated number */
1424 };

1426 /*
1427 * uname(2) descriptor - used to break a utsname structure into its component
1428 * options (at least those that we're interested in).
1429 */
1430 struct uts_desc {
1431     char      *uts_osname;    /* operating system name */
1432     size_t    uts_osnamesz;  /* and associated size */
1433     char      *uts_osrel;    /* operating system release */
1434     size_t    uts_osrelsz;  /* and associated size */
1435 };

1437 /*
1438 * SHT_GROUP descriptor - used to track group sections at the global
1439 * level to resolve conflicts and determine which to keep.
1440 */
1441 struct group_desc {
1442     Is_desc   *gd_isc;       /* input section descriptor */
1443     Is_desc   *gd_oisc;     /* overriding input section */
1444     /* descriptor when discarded */
1445     const char *gd_name;     /* group name (signature symbol) */
1446     Word      *gd_data;     /* data for group section */
1447     size_t    gd_cnt;       /* number of entries in group data */
1448 };

1450 /*
1451 * Indexes into the ld_support_funcs[] table.
1452 */
1453 typedef enum {
1454     LDS_VERSION = 0,        /* Must be first and have value 0 */
1455     LDS_INPUT_DONE,
1456     LDS_START,
1457     LDS_ATEXIT,
1458     LDS_OPEN,
1459     LDS_FILE,
1460     LDS_INSEC,
1461     LDS_SEC,
1462     LDS_NUM
1463 } Support_ndx;

1465 /*
1466 * Structure to manage archive member caching. Each archive has an archive
1467 * descriptor (Ar_desc) associated with it. This contains pointers to the
1468 * archive symbol table (obtained by elf_getarsyms(3e)) and an auxiliary
1469 * structure (Ar_uax[]) that parallels this symbol table. The member element
1470 * of this auxiliary table indicates whether the archive member associated with
1471 * the symbol offset has already been extracted (AREXTRACTED) or partially
1472 * processed (refer process_member()).
1473 */
1474 typedef struct ar_mem {
1475     Elf      *am_elf;        /* elf descriptor for this member */
1476     const char *am_name;     /* members name */
1477     const char *am_path;     /* path (ie. lib(foo.o)) */
1478     Sym      *am_syms;       /* start of global symbols */
1479     char      *am_strs;      /* associated string table start */
1480     Xword     am_symn;       /* no. of global symbols */
1481 } Ar_mem;

1483 typedef struct ar_aux {

```

```

1484     Sym_desc *au_syms;      /* internal symbol descriptor */
1485     Ar_mem   *au_mem;       /* associated member */
1486 } Ar_aux;

1488 #define FLG_ARMEM_PROC (Ar_mem *)-1

1490 typedef struct ar_desc {
1491     const char *ad_name;    /* archive file name */
1492     Elf      *ad_elf;      /* elf descriptor for the archive */
1493     Elf_Arsym *ad_start;   /* archive symbol table start */
1494     Ar_aux    *ad_aux;     /* auxiliary symbol information */
1495     dev_t     ad_stdev;    /* device id and inode number for */
1496     ino_t     ad_stino;    /* multiple inclusion checks */
1497     ofl_flag_t ad_flags;   /* archive specific cmd line flags */
1498 } Ar_desc;

1500 /*
1501 * Define any archive descriptor flags. NOTE, make sure they do not clash with
1502 * any output file descriptor archive extraction flags, as these are saved in
1503 * the same entry (see MSK_OF1_ARCHIVE).
1504 */
1505 #define FLG_ARD_EXTRACT 0x00010000 /* archive member has been extracted */

1507 /* Mapfile versions supported by libld */
1508 #define MFV_NONE 0 /* Not a valid version */
1509 #define MFV_SYSV 1 /* Original System V syntax */
1510 #define MFV_SOLARIS 2 /* Solaris mapfile syntax */
1511 #define MFV_NUM 3 /* # of mapfile versions */

1514 /*
1515 * Function Declarations.
1516 */
1517 #if defined(_ELF64)

1519 #define ld_create_outfile ld64_create_outfile
1520 #define ld_ent_setup ld64_ent_setup
1521 #define ld_init_strings ld64_init_strings
1522 #define ld_init_target ld64_init_target
1523 #define ld_make_sections ld64_make_sections
1524 #define ld_main ld64_main
1525 #define ld_ofl_cleanup ld64_ofl_cleanup
1526 #define ld_process_mem ld64_process_mem
1527 #define ld_reloc_init ld64_reloc_init
1528 #define ld_reloc_process ld64_reloc_process
1529 #define ld_sym_validate ld64_sym_validate
1530 #define ld_update_outfile ld64_update_outfile

1532 #else

1534 #define ld_create_outfile ld32_create_outfile
1535 #define ld_ent_setup ld32_ent_setup
1536 #define ld_init_strings ld32_init_strings
1537 #define ld_init_target ld32_init_target
1538 #define ld_make_sections ld32_make_sections
1539 #define ld_main ld32_main
1540 #define ld_ofl_cleanup ld32_ofl_cleanup
1541 #define ld_process_mem ld32_process_mem
1542 #define ld_reloc_init ld32_reloc_init
1543 #define ld_reloc_process ld32_reloc_process
1544 #define ld_sym_validate ld32_sym_validate
1545 #define ld_update_outfile ld32_update_outfile

1547 #endif

1549 extern int ld_getopt(Lm_list *, int, int, char **);

```

```
1551 extern int          ld32_main(int, char **, Half);
1552 extern int          ld64_main(int, char **, Half);

1554 extern uintptr_t    ld_create_outfile(Of1_desc *);
1555 extern uintptr_t    ld_ent_setup(Of1_desc *, Xword);
1556 extern uintptr_t    ld_init_strings(Of1_desc *);
1557 extern int          ld_init_target(Lm_list *, Half mach);
1558 extern uintptr_t    ld_make_sections(Of1_desc *);
1559 extern void          ld_ofl_cleanup(Of1_desc *);
1560 extern Ifl_desc      *ld_process_mem(const char *, const char *, char *,
1561                                     size_t, Of1_desc *, Rej_desc *);
1562 extern uintptr_t    ld_reloc_init(Of1_desc *);
1563 extern uintptr_t    ld_reloc_process(Of1_desc *);
1564 extern uintptr_t    ld_sym_validate(Of1_desc *);
1565 extern uintptr_t    ld_update_outfile(Of1_desc *);

1567 #ifdef __cplusplus
1568 }
1569 #endif

1571 #endif /* _LIBLD_H */
```

```

*****
87999 Thu Jun 30 22:01:27 2016
new/usr/src/cmd/sgs/libconv/common/corenote.c
Code review comments from jeffpc
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */

22 /*
23  * Copyright 2010 Sun Microsystems, Inc. All rights reserved.
24  * Use is subject to license terms.
25 */
26 /*
27  * Copyright 2012 DEY Storage Systems, Inc. All rights reserved.
28  * Copyright (c) 2013, Joyent, Inc. All rights reserved.
29 */

31 /*
32  * String conversion routines the system structs found in
33  * Solaris core file note sections. These items are not
34  * ELF constructs. However, elfdump contains code for decoding
35  * them, and therefore requires formatting support.
36 */
37 #include <stdio.h>
38 #include <procfs.h>
39 #include <sys/corectl.h>
40 #include <sys/secflags.h>
41 #endif /* ! codereview */
42 #include <string.h>
43 #include <conv.h>
44 #include <corenote_msg.h>

46 const char *
47 conv_cnote_type(Word type, Conv_fmt_flags_t fmt_flags,
48 Conv_inv_buf_t *inv_buf)
49 {
50     static const Msg types[] = {
51         MSG_NT_PRSTATUS, MSG_NT_PRFPREG,
52         MSG_NT_PRPSINFO, MSG_NT_PRXREG,
53         MSG_NT_PLATFORM, MSG_NT_AUXV,
54         MSG_NT_GWINDOWS, MSG_NT_ASR,
55         MSG_NT_LDT, MSG_NT_PSTATUS,
56         0, 0,
57         MSG_NT_PSINFO, MSG_NT_PRCRED,

```

```

58         MSG_NT_UTSNAME, MSG_NT_LWPSTATUS,
59         MSG_NT_LWPSINFO, MSG_NT_PRPRIV,
60         MSG_NT_PRPRIVINFO, MSG_NT_CONTENT,
61         MSG_NT_ZONENAME, MSG_NT_FDINFO,
62         MSG_NT_SPYMASTER, MSG_NT_SECFLAGS
40         MSG_NT_SPYMASTER
63     };
64 #if NT_NUM != NT_SECFLAGS
42 #if NT_NUM != NT_SPYMASTER
65 #error "NT_NUM has grown. Update core note types[]"
66 #endif
67     static const conv_ds_msg_t ds_types = {
68         CONV_DS_MSG_INIT(NT_PRSTATUS, types) };
69     static const conv_ds_t *ds[] = { CONV_DS_ADDR(ds_types), NULL };

72     return (conv_map_ds(ELFOSABI_NONE, EM_NONE, type, ds, fmt_flags,
73 inv_buf));
74 }

77 const char *
78 conv_cnote_auxv_type(Word type, Conv_fmt_flags_t fmt_flags,
79 Conv_inv_buf_t *inv_buf)
80 {
81     static const Msg types_0_22[] = {
82         MSG_AUXV_AT_NULL, MSG_AUXV_AT_IGNORE,
83         MSG_AUXV_AT_EXECD, MSG_AUXV_AT_PHDR,
84         MSG_AUXV_AT_PHENT, MSG_AUXV_AT_PHNUM,
85         MSG_AUXV_AT_PAGESZ, MSG_AUXV_AT_BASE,
86         MSG_AUXV_AT_FLAGS, MSG_AUXV_AT_ENTRY,
87         MSG_AUXV_AT_NOTELF, MSG_AUXV_AT_UID,
88         MSG_AUXV_AT_EUID, MSG_AUXV_AT_GID,
89         MSG_AUXV_AT_EGID, MSG_AUXV_AT_PLATFORM,
90         MSG_AUXV_AT_HWCAP, MSG_AUXV_AT_CLKTCK,
91         MSG_AUXV_AT_FPUCW, MSG_AUXV_AT_DCACHEBSIZE,
92         MSG_AUXV_AT_ICACHEBSIZE, MSG_AUXV_AT_UCACHEBSIZE,
93         MSG_AUXV_AT_IGNOREPPC
94     };
95     static const conv_ds_msg_t ds_types_0_22 = {
96         CONV_DS_MSG_INIT(0, types_0_22) };

98     static const Msg types_2000_2011[] = {
99         MSG_AUXV_AT_SUN_UID, MSG_AUXV_AT_SUN_RUID,
100        MSG_AUXV_AT_SUN_GID, MSG_AUXV_AT_SUN_RGID,
101        MSG_AUXV_AT_SUN_LDELF, MSG_AUXV_AT_SUN_LDSHDR,
102        MSG_AUXV_AT_SUN_LDNAME, MSG_AUXV_AT_SUN_LPAGESZ,
103        MSG_AUXV_AT_SUN_PLATFORM, MSG_AUXV_AT_SUN_HWCAP,
104        MSG_AUXV_AT_SUN_IFLUSH, MSG_AUXV_AT_SUN_CPU
105     };
106     static const conv_ds_msg_t ds_types_2000_2011 = {
107         CONV_DS_MSG_INIT(2000, types_2000_2011) };

109     static const Msg types_2014_2023[] = {
110        MSG_AUXV_AT_SUN_EXECNAME, MSG_AUXV_AT_SUN_MMU,
111        MSG_AUXV_AT_SUN_LDDATA, MSG_AUXV_AT_SUN_AUXFLAGS,
112        MSG_AUXV_AT_SUN_EMULATOR, MSG_AUXV_AT_SUN_BRANDNAME,
113        MSG_AUXV_AT_SUN_BRAND_AUX1, MSG_AUXV_AT_SUN_BRAND_AUX2,
114        MSG_AUXV_AT_SUN_BRAND_AUX3, MSG_AUXV_AT_SUN_HWCAP2,
92        MSG_AUXV_AT_SUN_BRAND_AUX3, MSG_AUXV_AT_SUN_HWCAP2
115     };
116     static const conv_ds_msg_t ds_types_2014_2023 = {
117         CONV_DS_MSG_INIT(2014, types_2014_2023) };

119     static const conv_ds_t *ds[] = {
120         CONV_DS_ADDR(ds_types_0_22), CONV_DS_ADDR(ds_types_2000_2011),

```



```

121         CONV_DS_ADDR(ds_types_2014_2023), NULL };
123     return (conv_map_ds(ELFOSABI_NONE, EM_NONE, type, ds, fmt_flags,
124         inv_buf));
125 }
_____unchanged_portion_omitted_____
2589 #define PROCSECFLSZ      CONV_EXPN_FIELD_DEF_PREFIX_SIZE +      \
2590     MSG_ASLR_SIZE        + CONV_EXPN_FIELD_DEF_SEP_SIZE +      \
2591     MSG_FORBIDNULLMAP_SIZE + CONV_EXPN_FIELD_DEF_SEP_SIZE +      \
2592     MSG_NOEXECSTACK_SIZE + CONV_EXPN_FIELD_DEF_SEP_SIZE +      \
2593     CONV_INV_BUFSIZE     + CONV_EXPN_FIELD_DEF_SUFFIX_SIZE

2595 /*
2596  * Ensure that Conv_cnote_pr_secflags_buf_t is large enough:
2597  *
2598  * PROCSECFLSZ is the real minimum size of the buffer required by
2599  * conv_prsecflags(). However, Conv_cnote_pr_secflags_buf_t uses
2600  * CONV_CNOTE_PSECFLAGS_FLAG_BUFSIZE to set the buffer size. We do things this
2601  * way because the definition of PROCSECFLSZ uses information that is not
2602  * available in the environment of other programs that include the conv.h
2603  * header file.
2604  */
2605 #if (CONV_PRSECFLAGS_BUFSIZE != PROCSECFLSZ) && !defined(__lint)
2606 #define REPORT_BUFSIZE PROCSECFLSZ
2607 #include "report_bufsize.h"
2608 #error "CONV_PRSECFLAGS_BUFSIZE does not match PROCSECFLSZ"
2609 #endif

2611 const char *
2612 conv_prsecflags(secflagset_t flags, Conv_fmt_flags_t fmt_flags,
2613     Conv_secflags_buf_t *secflags_buf)
2614 {
2615     /*
2616      * The values are initialized later, based on position in this array
2617      */
2618     static Val_desc vda[] = {
2619         { 0, MSG_ASLR },
2620         { 0, MSG_FORBIDNULLMAP },
2621         { 0, MSG_NOEXECSTACK },
2622         { 0, 0 }
2623     };
2624     static CONV_EXPN_FIELD_ARG conv_arg = {
2625         NULL, sizeof (secflags_buf->buf)
2626     };
2627     int i;

2629     for (i = 0; vda[i].v_msg != 0; i++)
2630         vda[i].v_val = secflag_to_bit(i);

2632     if (flags == 0)
2633         return (MSG_ORIG(MSG_GBL_ZERO));

2635     conv_arg.buf = secflags_buf->buf;
2636     conv_arg.oflags = conv_arg.rflags = flags;
2637     (void) conv_expn_field(&conv_arg, vda, fmt_flags);

2639     return ((const char *)secflags_buf->buf);
2640 }
2641 #endif /* ! codereview */

```

```

*****
39726 Thu Jun 30 22:01:28 2016
new/usr/src/cmd/sgs/libconv/common/corenote.msg
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
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 #
23 # Copyright 2010 Sun Microsystems, Inc. All rights reserved.
24 # Use is subject to license terms.
25 #
26 # Copyright 2012 DEY Storage Systems, Inc. All rights reserved.
27 # Copyright (c) 2013, Joyent, Inc. All rights reserved.
28 #
30 @ MSG_NT_PRSTATUS           "[ NT_PRSTATUS ]"
31 @ MSG_NT_PRFPREG            "[ NT_PRFPREG ]"
32 @ MSG_NT_PRPSINFO           "[ NT_PRPSINFO ]"
33 @ MSG_NT_PRXREG             "[ NT_PRXREG ]"
34 @ MSG_NT_PLATFORM           "[ NT_PLATFORM ]"
35 @ MSG_NT_AUXV                "[ NT_AUXV ]"
36 @ MSG_NT_GWINDOWS           "[ NT_GWINDOWS ]"
37 @ MSG_NT_ASRS               "[ NT_ASRS ]"
38 @ MSG_NT_LDT                 "[ NT_LDT ]"
39 @ MSG_NT_PSTATUS            "[ NT_PSTATUS ]"
40 @ MSG_NT_PSINFO             "[ NT_PSINFO ]"
41 @ MSG_NT_PRCRED             "[ NT_PRCRED ]"
42 @ MSG_NT_UTSNAME            "[ NT_UTSNAME ]"
43 @ MSG_NT_LWPSTATUS           "[ NT_LWPSTATUS ]"
44 @ MSG_NT_LWPSINFO           "[ NT_LWPSINFO ]"
45 @ MSG_NT_PRPRIV             "[ NT_PRPRIV ]"
46 @ MSG_NT_PRPRIVINFO         "[ NT_PRPRIVINFO ]"
47 @ MSG_NT_CONTENT            "[ NT_CONTENT ]"
48 @ MSG_NT_ZONENAME           "[ NT_ZONENAME ]"
49 @ MSG_NT_FDINFO             "[ NT_FDINFO ]"
50 @ MSG_NT_SPYMASTER          "[ NT_SPYMASTER ]"
51 @ MSG_NT_SECFLAGS           "[ NT_SECFLAGS ]"
52 #endif /* ! codereview */

55 @ MSG_AUXV_AF_SUN_SETUGID   "AF_SUN_SETUGID"
56 @ MSG_AUXV_AF_SUN_HWCAPVERIFY "AF_SUN_HWCAPVERIFY"
57 @ MSG_AUXV_AF_SUN_NOPLM     "AF_SUN_NOPLM"

```

```

60 @ MSG_AUXV_AT_NULL          "NULL"
61 @ MSG_AUXV_AT_IGNORE        "IGNORE"
62 @ MSG_AUXV_AT_EXECONF       "EXECFND"
63 @ MSG_AUXV_AT_PHDR          "PHDR"
64 @ MSG_AUXV_AT_PHEMT         "PHEMT"
65 @ MSG_AUXV_AT_PHNUM         "PHNUM"
66 @ MSG_AUXV_AT_PAGESZ       "PAGESZ"
67 @ MSG_AUXV_AT_BASE          "BASE"
68 @ MSG_AUXV_AT_FLAGS         "FLAGS"
69 @ MSG_AUXV_AT_ENTRY         "ENTRY"
70 @ MSG_AUXV_AT_NOTELF        "NOTELF"
71 @ MSG_AUXV_AT_UID           "UID"
72 @ MSG_AUXV_AT_EUID          "EUID"
73 @ MSG_AUXV_AT_GID           "GID"
74 @ MSG_AUXV_AT_EGID          "EGID"
75 @ MSG_AUXV_AT_PLATFORM     "PLATFORM"
76 @ MSG_AUXV_AT_HWCAP         "HWCAP"
77 @ MSG_AUXV_AT_CLKTCK        "CLKTCK"
78 @ MSG_AUXV_AT_FPUCW         "FPUCW"
79 @ MSG_AUXV_AT_DCACHEBSIZE   "DCACHEBSIZE"
80 @ MSG_AUXV_AT_ICACHEBSIZE   "ICACHEBSIZE"
81 @ MSG_AUXV_AT_UCACHEBSIZE   "UCACHEBSIZE"
82 @ MSG_AUXV_AT_IGNOREPPC     "IGNOREPPC"
83 @ MSG_AUXV_AT_SUN_UID       "SUN_UID"
84 @ MSG_AUXV_AT_SUN_RUID      "SUN_RUID"
85 @ MSG_AUXV_AT_SUN_GID       "SUN_GID"
86 @ MSG_AUXV_AT_SUN_RGID      "SUN_RGID"
87 @ MSG_AUXV_AT_SUN_LDELF     "SUN_LDELF"
88 @ MSG_AUXV_AT_SUN_LDSDR     "SUN_LDSDR"
89 @ MSG_AUXV_AT_SUN_LDNAME     "SUN_LDNAME"
90 @ MSG_AUXV_AT_SUN_LPAGESZ    "SUN_LPAGESZ"
91 @ MSG_AUXV_AT_SUN_PLATFORM   "SUN_PLATFORM"
92 @ MSG_AUXV_AT_SUN_HWCAP      "SUN_HWCAP"
93 @ MSG_AUXV_AT_SUN_IFLUSH     "SUN_IFLUSH"
94 @ MSG_AUXV_AT_SUN_CPU        "SUN_CPU"
95 @ MSG_AUXV_AT_SUN_EXECPNAME "SUN_EXECPNAME"
96 @ MSG_AUXV_AT_SUN_MMU        "SUN_MMU"
97 @ MSG_AUXV_AT_SUN_LDDATA     "SUN_LDDATA"
98 @ MSG_AUXV_AT_SUN_AUXFLAGS   "SUN_AUXFLAGS"
99 @ MSG_AUXV_AT_SUN_EMULATOR  "SUN_EMULATOR"
100 @ MSG_AUXV_AT_SUN_BRANDNAME  "SUN_BRANDNAME"
101 @ MSG_AUXV_AT_SUN_BRAND_AUX1 "SUN_BRAND_AUX1"
102 @ MSG_AUXV_AT_SUN_BRAND_AUX2 "SUN_BRAND_AUX2"
103 @ MSG_AUXV_AT_SUN_BRAND_AUX3 "SUN_BRAND_AUX3"
104 @ MSG_AUXV_AT_SUN_HWCAP2     "SUN_HWCAP2"

106 @ MSG_CC_CONTENT_STACK      "STACK"
107 @ MSG_CC_CONTENT_HEAP        "HEAP"
108 @ MSG_CC_CONTENT_SHFILE      "SHFILE"
109 @ MSG_CC_CONTENT_SHANON      "SHANON"
110 @ MSG_CC_CONTENT_TEXT        "TEXT"
111 @ MSG_CC_CONTENT_DATA        "DATA"
112 @ MSG_CC_CONTENT_RODATA      "RODATA"
113 @ MSG_CC_CONTENT_ANON        "ANON"
114 @ MSG_CC_CONTENT_SHM         "SHM"
115 @ MSG_CC_CONTENT_ISM         "ISM"
116 @ MSG_CC_CONTENT_DISM        "DISM"
117 @ MSG_CC_CONTENT_CTF         "CTF"
118 @ MSG_CC_CONTENT_SYMTAB      "SYMTAB"

121 @ MSG_ERRNO_EPERM           "[ EPERM ]"
122 @ MSG_ERRNO_ENOENT           "[ ENOENT ]"
123 @ MSG_ERRNO_ESRCH            "[ ESRCH ]"

```

1
2
3

```

124 @ MSG_ERRNO_EINTR      "[ EINTR ]"          # 4
125 @ MSG_ERRNO_EIO        "[ EIO ]"            # 5
126 @ MSG_ERRNO_ENXIO      "[ ENXIO ]"          # 6
127 @ MSG_ERRNO_E2BIG      "[ E2BIG ]"            # 7
128 @ MSG_ERRNO_ENOEXEC    "[ ENOEXEC ]"          # 8
129 @ MSG_ERRNO_EBADF      "[ EBADF ]"            # 9
130 @ MSG_ERRNO_ECHILD     "[ ECHILD ]"          # 10
131 @ MSG_ERRNO_EAGAIN     "[ EAGAIN ]"         # 11
132 @ MSG_ERRNO_ENOMEM     "[ ENOMEM ]"         # 12
133 @ MSG_ERRNO_EACCES     "[ EACCES ]"         # 13
134 @ MSG_ERRNO_EFAULT     "[ EFAULT ]"         # 14
135 @ MSG_ERRNO_ENOTBLK    "[ ENOTBLK ]"       # 15
136 @ MSG_ERRNO_EBUSY     "[ EBUSY ]"          # 16
137 @ MSG_ERRNO_EXIST      "[ EXIST ]"          # 17
138 @ MSG_ERRNO_EXDEV     "[ EXDEV ]"          # 18
139 @ MSG_ERRNO_ENODEV     "[ ENODEV ]"         # 19
140 @ MSG_ERRNO_ENOTDIR    "[ ENOTDIR ]"       # 20
141 @ MSG_ERRNO_EISDIR     "[ EISDIR ]"         # 21
142 @ MSG_ERRNO_EINVAL     "[ EINVAL ]"         # 22
143 @ MSG_ERRNO_ENFILE     "[ ENFILE ]"         # 23
144 @ MSG_ERRNO_EMFILE     "[ EMFILE ]"         # 24
145 @ MSG_ERRNO_ENOTTY     "[ ENOTTY ]"         # 25
146 @ MSG_ERRNO_ETXTBSY   "[ ETXTBSY ]"       # 26
147 @ MSG_ERRNO_EFBIG     "[ EFBIG ]"          # 27
148 @ MSG_ERRNO_ENOSPC     "[ ENOSPC ]"         # 28
149 @ MSG_ERRNO_ESPIPE     "[ ESPIPE ]"         # 29
150 @ MSG_ERRNO_EROFS      "[ EROFS ]"          # 30
151 @ MSG_ERRNO_EMLINK     "[ EMLINK ]"        # 31
152 @ MSG_ERRNO_EPIPE     "[ EPIPE ]"          # 32
153 @ MSG_ERRNO_EDOM       "[ EDOM ]"           # 33
154 @ MSG_ERRNO_ERANGE     "[ ERANGE ]"        # 34
155 @ MSG_ERRNO_ENOMSG     "[ ENOMSG ]"        # 35
156 @ MSG_ERRNO_EIDRM      "[ EIDRM ]"         # 36
157 @ MSG_ERRNO_ECHRNG     "[ ECHRNG ]"         # 37
158 @ MSG_ERRNO_EL2NSYNC   "[ EL2NSYNC ]"      # 38
159 @ MSG_ERRNO_EL3HLT     "[ EL3HLT ]"         # 39
160 @ MSG_ERRNO_EL3RST     "[ EL3RST ]"         # 40
161 @ MSG_ERRNO_ELNRRNG    "[ ELNRRNG ]"       # 41
162 @ MSG_ERRNO_EUNATCH    "[ EUNATCH ]"       # 42
163 @ MSG_ERRNO_ENOCSI     "[ ENOCSI ]"        # 43
164 @ MSG_ERRNO_EL2HLT     "[ EL2HLT ]"         # 44
165 @ MSG_ERRNO_EDEADLK    "[ EDEADLK ]"       # 45
166 @ MSG_ERRNO_ENOLCK     "[ ENOLCK ]"         # 46
167 @ MSG_ERRNO_ECANCELED  "[ ECANCELED ]"     # 47
168 @ MSG_ERRNO_ENOTSUP    "[ ENOTSUP ]"       # 48
169 @ MSG_ERRNO_EDQUOT     "[ EDQUOT ]"        # 49
170 @ MSG_ERRNO_EBADE     "[ EBADE ]"         # 50
171 @ MSG_ERRNO_EBADR     "[ EBADR ]"         # 51
172 @ MSG_ERRNO_EXFULL     "[ EXFULL ]"        # 52
173 @ MSG_ERRNO_ENOANO     "[ ENOANO ]"        # 53
174 @ MSG_ERRNO_EBADRQC    "[ EBADRQC ]"       # 54
175 @ MSG_ERRNO_EBADSLT    "[ EBADSLT ]"       # 55
176 @ MSG_ERRNO_EDEADLOCK  "[ EDEADLOCK ]"    # 56
177 @ MSG_ERRNO_EBFONT     "[ EBFONT ]"        # 57
178 @ MSG_ERRNO_EOWNERDEAD "[ EOWNERDEAD ]"   # 58
179 @ MSG_ERRNO_ENOTRECOVERABLE "[ ENOTRECOVERABLE ]" # 59
180 @ MSG_ERRNO_ENOSTR     "[ ENOSTR ]"        # 60
181 @ MSG_ERRNO_ENODATA    "[ ENODATA ]"       # 61
182 @ MSG_ERRNO_ETIME      "[ ETIME ]"         # 62
183 @ MSG_ERRNO_ENOSR     "[ ENOSR ]"         # 63
184 @ MSG_ERRNO_ENONET     "[ ENONET ]"        # 64
185 @ MSG_ERRNO_ENOPKG     "[ ENOPKG ]"        # 65
186 @ MSG_ERRNO_EREMOTE    "[ EREMOTE ]"       # 66
187 @ MSG_ERRNO_ENOLINK    "[ ENOLINK ]"       # 67
188 @ MSG_ERRNO_EADV       "[ EADV ]"          # 68
189 @ MSG_ERRNO_ESRMNT     "[ ESRMNT ]"        # 69

```

```

190 @ MSG_ERRNO_ECOMM      "[ ECOMM ]"          # 70
191 @ MSG_ERRNO_EPROTO     "[ EPROTO ]"          # 71
192 @ MSG_ERRNO_ELOCKUNMAPPED "[ ELOCKUNMAPPED ]" # 72
193 @ MSG_ERRNO_ENOTACTIVE "[ ENOTACTIVE ]"    # 73
194 @ MSG_ERRNO_EMULTIHOP  "[ EMULTIHOP ]"    # 74
195 # errno 75 and 76 are not defined
196 @ MSG_ERRNO_EBADMSG    "[ EBADMSG ]"       # 77
197 @ MSG_ERRNO_ENAMETOOLONG "[ ENAMETOOLONG ]" # 78
198 @ MSG_ERRNO_EOVERFLOW  "[ EOVERFLOW ]"    # 79
199 @ MSG_ERRNO_ENOTUNIQ   "[ ENOTUNIQ ]"     # 80
200 @ MSG_ERRNO_EBADFD     "[ EBADF ]"         # 81
201 @ MSG_ERRNO_EREMCHG    "[ EREMCHG ]"      # 82
202 @ MSG_ERRNO_ELIBACC    "[ ELIBACC ]"      # 83
203 @ MSG_ERRNO_ELIBBAD    "[ ELIBBAD ]"      # 84
204 @ MSG_ERRNO_ELIBSCN    "[ ELIBSCN ]"     # 85
205 @ MSG_ERRNO_ELIBMAX    "[ ELIBMAX ]"     # 86
206 @ MSG_ERRNO_ELIBEXEC   "[ ELIBEXEC ]"    # 87
207 @ MSG_ERRNO_EILSEQ     "[ EILSEQ ]"       # 88
208 @ MSG_ERRNO_ENOSYS     "[ ENOSYS ]"       # 89
209 @ MSG_ERRNO_ELOOP     "[ ELOOP ]"        # 90
210 @ MSG_ERRNO_ERESTART   "[ ERESTART ]"     # 91
211 @ MSG_ERRNO ESTRPIPE   "[ ESTRPIPE ]"     # 92
212 @ MSG_ERRNO_ ENOTEMPTY "[ ENOTEMPTY ]"    # 93
213 @ MSG_ERRNO_EUSERS     "[ EUSERS ]"       # 94
214 @ MSG_ERRNO_ENOTSOCK   "[ ENOTSOCK ]"     # 95
215 @ MSG_ERRNO_EDESTADDRREQ "[ EDESTADDRREQ ]" # 96
216 @ MSG_ERRNO EMSGSIZE   "[ EMSGSIZE ]"     # 97
217 @ MSG_ERRNO_EPROTOTYPE "[ EPROTOTYPE ]"   # 98
218 @ MSG_ERRNO_ENOPROTOOPT "[ ENOPROTOOPT ]"  # 99
219 # Gap in ERRNO values 100-119
220 @ MSG_ERRNO_EPROTONOSUPPORT "[ EPROTONOSUPPORT ]" # 120
221 @ MSG_ERRNO_ESOCKTNOSUPPORT "[ ESOCKTNOSUPPORT ]" # 121
222 @ MSG_ERRNO_EOPNOTSUPP   "[ EOPNOTSUPP ]"    # 122
223 @ MSG_ERRNO_EPFNOSUPPORT "[ EPFNOSUPPORT ]"  # 123
224 @ MSG_ERRNO_EAFNOSUPPORT "[ EAFNOSUPPORT ]"  # 124
225 @ MSG_ERRNO_EADDRINUSE   "[ EADDRINUSE ]"    # 125
226 @ MSG_ERRNO_EADDRNOTAVAIL "[ EADDRNOTAVAIL ]" # 126
227 @ MSG_ERRNO_ENETDOWN    "[ ENETDOWN ]"     # 127
228 @ MSG_ERRNO_ENETUNREACH "[ ENETUNREACH ]"   # 128
229 @ MSG_ERRNO_ENETRESET   "[ ENETRESET ]"    # 129
230 @ MSG_ERRNO_ECONNABORTED "[ ECONNABORTED ]"  # 130
231 @ MSG_ERRNO_ECONNRESET   "[ ECONNRESET ]"    # 131
232 @ MSG_ERRNO_ENOBUFS     "[ ENOBUFS ]"      # 132
233 @ MSG_ERRNO_EISCONN     "[ EISCONN ]"      # 133
234 @ MSG_ERRNO_ENOTCONN    "[ ENOTCONN ]"     # 134
235 #XENIX has 135 - 142
236 @ MSG_ERRNO_ESHUTDOWN   "[ ESHUTDOWN ]"    # 143
237 @ MSG_ERRNO_ETOOMANYREFS "[ ETOOMANYREFS ]"  # 144
238 @ MSG_ERRNO_ETIMEOUT     "[ ETIMEOUT ]"     # 145
239 @ MSG_ERRNO_ECONNREFUSED "[ ECONNREFUSED ]"  # 146
240 @ MSG_ERRNO_EHOSTDOWN    "[ EHOSTDOWN ]"    # 147
241 @ MSG_ERRNO_EHOSTUNREACH "[ EHOSTUNREACH ]" # 148
242 @ MSG_ERRNO_EALREADY     "[ EALREADY ]"     # 149
243 @ MSG_ERRNO_EINPROGRESS  "[ EINPROGRESS ]"   # 150
244 @ MSG_ERRNO_ESTALE      "[ ESTALE ]"        # 151

247 @ MSG_PR_MODEL_UNKNOWN "[ PR_MODEL_UNKNOWN ]"
248 @ MSG_PR_MODEL_ILP32   "[ PR_MODEL_ILP32 ]"
249 @ MSG_PR_MODEL_LP64    "[ PR_MODEL_LP64 ]"

251 @ MSG_PR_FLAGS_STOPPED "[ PR_STOPPED ]"
252 @ MSG_PR_FLAGS_ISTOP   "[ PR_ISTOP ]"
253 @ MSG_PR_FLAGS_DSTOP   "[ PR_DSTOP ]"
254 @ MSG_PR_FLAGS_STEP    "[ PR_STEP ]"
255 @ MSG_PR_FLAGS_ASLEEP  "[ PR_ASLEEP ]"

```

```

256 @ MSG_PR_FLAGS_PCINVAL      "PR_PCINVAL"
257 @ MSG_PR_FLAGS_ASLWP       "PR_ASLWP"
258 @ MSG_PR_FLAGS_AGENT        "PR_AGENT"
259 @ MSG_PR_FLAGS_DETACH       "PR_DETACH"
260 @ MSG_PR_FLAGS_DAEMON       "PR_DAEMON"
261 @ MSG_PR_FLAGS_IDLE         "PR_IDLE"
262 @ MSG_PR_FLAGS_ISSYS        "PR_ISSYS"
263 @ MSG_PR_FLAGS_VFORKP       "PR_VFORKP"
264 @ MSG_PR_FLAGS_ORPHAN       "PR_ORPHAN"
265 @ MSG_PR_FLAGS_NOSIGCHLD    "PR_NOSIGCHLD"
266 @ MSG_PR_FLAGS_WAITPID      "PR_WAITPID"
267 @ MSG_PR_FLAGS_FORK         "PR_FORK"
268 @ MSG_PR_FLAGS_RLC          "PR_RLC"
269 @ MSG_PR_FLAGS_KLC          "PR_KLC"
270 @ MSG_PR_FLAGS_ASYNC        "PR_ASYNC"
271 @ MSG_PR_FLAGS_MSACCT       "PR_MSACCT"
272 @ MSG_PR_FLAGS_BPTADJ       "PR_BPTADJ"
273 @ MSG_PR_FLAGS_PTRACE       "PR_PTRACE"
274 @ MSG_PR_FLAGS_MSFORK       "PR_MSFORK"
275 @ MSG_PR_FLAGS_PCOMPAT      "PR_PCOMPAT"

```

```

278 @ MSG_PROC_FLAG_SSYS        "SSYS"
279 @ MSG_PROC_FLAG_SMSACCT     "SMSACCT"

```

```

281 @ MSG_ASLR                   "ASLR"
282 @ MSG_FORBIDNULLMAP          "FORBIDNULLMAP"
283 @ MSG_NOEXECSTACK           "NOEXECSTACK"

```

```
285 #endif /* ! codereview */
```

```

286 @ MSG_PS_NONE                 "[ PS_NONE ]"
287 @ MSG_PS_QUERY               "[ PS_QUERY ]"
288 @ MSG_PS_MYID                "[ PS_MYID ]"
289 @ MSG_PS_SOFT                "[ PS_SOFT ]"
290 @ MSG_PS_HARD                "[ PS_HARD ]"
291 @ MSG_PS_QUERY_TYPE          "[ PS_QUERY_TYPE ]"

```

```

294 @ MSG_REG_SPARC_G0           "[ r0/g0 ]"
295 @ MSG_REG_SPARC_G1           "[ r1/g1 ]"
296 @ MSG_REG_SPARC_G2           "[ r2/g2 ]"
297 @ MSG_REG_SPARC_G3           "[ r3/g3 ]"
298 @ MSG_REG_SPARC_G4           "[ r4/g4 ]"
299 @ MSG_REG_SPARC_G5           "[ r5/g5 ]"
300 @ MSG_REG_SPARC_G6           "[ r6/g6 ]"
301 @ MSG_REG_SPARC_G7           "[ r7/g7 ]"
302 @ MSG_REG_SPARC_O0           "[ r8/o0 ]"
303 @ MSG_REG_SPARC_O1           "[ r9/o1 ]"
304 @ MSG_REG_SPARC_O2           "[ r10/o2 ]"
305 @ MSG_REG_SPARC_O3           "[ r11/o3 ]"
306 @ MSG_REG_SPARC_O4           "[ r12/o4 ]"
307 @ MSG_REG_SPARC_O5           "[ r13/o5 ]"
308 @ MSG_REG_SPARC_O6           "[ r14/o6/sp ]"
309 @ MSG_REG_SPARC_O7           "[ r15/o7 ]"
310 @ MSG_REG_SPARC_L0           "[ r16/l0 ]"
311 @ MSG_REG_SPARC_L1           "[ r17/l1 ]"
312 @ MSG_REG_SPARC_L2           "[ r18/l2 ]"
313 @ MSG_REG_SPARC_L3           "[ r19/l3 ]"
314 @ MSG_REG_SPARC_L4           "[ r20/l4 ]"
315 @ MSG_REG_SPARC_L5           "[ r21/l5 ]"
316 @ MSG_REG_SPARC_L6           "[ r22/l6 ]"
317 @ MSG_REG_SPARC_L7           "[ r23/l7 ]"
318 @ MSG_REG_SPARC_I0           "[ r24/i0 ]"
319 @ MSG_REG_SPARC_I1           "[ r25/i1 ]"
320 @ MSG_REG_SPARC_I2           "[ r26/i2 ]"
321 @ MSG_REG_SPARC_I3           "[ r27/i3 ]"

```

```

322 @ MSG_REG_SPARC_I4           "[ r28/i4 ]"
323 @ MSG_REG_SPARC_I5           "[ r29/i5 ]"
324 @ MSG_REG_SPARC_I6           "[ r30/i6/fp ]"
325 @ MSG_REG_SPARC_I7           "[ r31/i7 ]"
326 @ MSG_REG_SPARC_CCR         "[ ccr ]"
327 @ MSG_REG_SPARC_PSR         "[ psr ]"
328 @ MSG_REG_SPARC_PC          "[ pc ]"
329 @ MSG_REG_SPARC_nPC         "[ npc ]"
330 @ MSG_REG_SPARC_Y            "[ y ]"
331 @ MSG_REG_SPARC_ASI         "[ asi ]"
332 @ MSG_REG_SPARC_FPRS        "[ fprs ]"
333 @ MSG_REG_SPARC_WIM         "[ wim ]"
334 @ MSG_REG_SPARC_TBR         "[ tbr ]"

```

```

336 @ MSG_REG_AMD64_R15         "[ r15 ]"
337 @ MSG_REG_AMD64_R14         "[ r14 ]"
338 @ MSG_REG_AMD64_R13         "[ r13 ]"
339 @ MSG_REG_AMD64_R12         "[ r12 ]"
340 @ MSG_REG_AMD64_R11         "[ r11 ]"
341 @ MSG_REG_AMD64_R10         "[ r10 ]"
342 @ MSG_REG_AMD64_R9          "[ r9 ]"
343 @ MSG_REG_AMD64_R8          "[ r8 ]"
344 @ MSG_REG_AMD64_RDI         "[ rdi ]"
345 @ MSG_REG_AMD64_RSI         "[ rsi ]"
346 @ MSG_REG_AMD64_RBP         "[ rbp ]"
347 @ MSG_REG_AMD64_RBX         "[ rbx ]"
348 @ MSG_REG_AMD64_RDX         "[ rdx ]"
349 @ MSG_REG_AMD64_RCX         "[ rcx ]"
350 @ MSG_REG_AMD64_RAX         "[ rax ]"
351 @ MSG_REG_AMD64_TRAPNO      "[ trapno ]"
352 @ MSG_REG_AMD64_ERR         "[ err ]"
353 @ MSG_REG_AMD64_RIP         "[ rip ]"
354 @ MSG_REG_AMD64_CS          "[ cs ]"
355 @ MSG_REG_AMD64_RFL         "[ rfl ]"
356 @ MSG_REG_AMD64_RSP         "[ rsp ]"
357 @ MSG_REG_AMD64_SS          "[ ss ]"
358 @ MSG_REG_AMD64_FS          "[ fs ]"
359 @ MSG_REG_AMD64_GS          "[ gs ]"
360 @ MSG_REG_AMD64_ES          "[ es ]"
361 @ MSG_REG_AMD64_DS          "[ ds ]"
362 @ MSG_REG_AMD64_FSBASE     "[ fsbase ]"
363 @ MSG_REG_AMD64_GSBASE     "[ gsbase ]"

```

```

365 @ MSG_REG_I86_GS            "[ gs ]"
366 @ MSG_REG_I86_FS            "[ fs ]"
367 @ MSG_REG_I86_ES            "[ es ]"
368 @ MSG_REG_I86_DS            "[ ds ]"
369 @ MSG_REG_I86 EDI           "[ edi ]"
370 @ MSG_REG_I86_ESI           "[ esi ]"
371 @ MSG_REG_I86_EBP           "[ ebp ]"
372 @ MSG_REG_I86_ESP           "[ esp ]"
373 @ MSG_REG_I86_EBX           "[ ebx ]"
374 @ MSG_REG_I86_EDX           "[ edx ]"
375 @ MSG_REG_I86_ECX           "[ ecx ]"
376 @ MSG_REG_I86_EAX           "[ eax ]"
377 @ MSG_REG_I86_TRAPNO       "[ trapno ]"
378 @ MSG_REG_I86_ERR           "[ err ]"
379 @ MSG_REG_I86_EIP           "[ eip ]"
380 @ MSG_REG_I86_CS            "[ cs ]"
381 @ MSG_REG_I86_EFL           "[ efl ]"
382 @ MSG_REG_I86_UESP         "[ uesp ]"
383 @ MSG_REG_I86_SS            "[ ss ]"

```

```

385 @ MSG_PR_WHY_REQUESTED      "[ PR_REQUESTED ]"
386 @ MSG_PR_WHY_SIGNALLED     "[ PR_SIGNALLED ]"
387 @ MSG_PR_WHY_SYSENTRY      "[ PR_SYSENTRY ]"

```

```

388 @ MSG_PR_WHY_SYSEXIT      "[ PR_SYSEXIT ]"
389 @ MSG_PR_WHY_JOBCONTROL   "[ PR_JOBCONTROL ]"
390 @ MSG_PR_WHY_FAULTED      "[ PR_FAULTED ]"
391 @ MSG_PR_WHY_SUSPENDED    "[ PR_SUSPENDED ]"
392 @ MSG_PR_WHY_CHECKPOINT   "[ PR_CHECKPOINT ]"

394 @ MSG_PRIV_ALL            "[ PRIV_ALL ]"
395 @ MSG_PRIV_MULTIPLE       "[ PRIV_MULTIPLE ]"
396 @ MSG_PRIV_NONE           "[ PRIV_NONE ]"
397 @ MSG_PRIV_ALLZONE        "[ PRIV_ALLZONE ]"
398 @ MSG_PRIV_GLOBAL         "[ PRIV_ALLGLOBAL ]"

401 @ MSG_SA_ONSTACK          "SA_ONSTACK"
402 @ MSG_SA_RESETHAND        "SA_RESETHAND"
403 @ MSG_SA_RESTART          "SA_RESTART"
404 @ MSG_SA_SIGINFO          "SA_SIGINFO"
405 @ MSG_SA_NODEFER          "SA_NODEFER"
406 @ MSG_SA_NOCLDWAIT        "SA_NOCLDWAIT"
407 @ MSG_SA_NOCLDSTOP        "SA_NOCLDSTOP"

410 @ MSG_SOBJ_NONE           "[ SOBJ_NONE ]"
411 @ MSG_SOBJ_MUTEX          "[ SOBJ_MUTEX ]"
412 @ MSG_SOBJ_RWLOCK         "[ SOBJ_RWLOCK ]"
413 @ MSG_SOBJ_CV             "[ SOBJ_CV ]"
414 @ MSG_SOBJ_SEMA           "[ SOBJ_SEMA ]"
415 @ MSG_SOBJ_USER           "[ SOBJ_USER ]"
416 @ MSG_SOBJ_USER_PI        "[ SOBJ_USER_PI ]"
417 @ MSG_SOBJ_SHUTTLE        "[ SOBJ_SHUTTLE ]"

420 @ MSG_SS_ONSTACK          "SS_ONSTACK"
421 @ MSG_SS_DISABLE          "SS_DISABLE"

424 @ MSG_SI_NOINFO           "[ SI_NOINFO ]"
425 @ MSG_SI_DTRACE           "[ SI_DTRACE ]"
426 @ MSG_SI_RCTL             "[ SI_RCTL ]"
427 @ MSG_SI_USER             "[ SI_USER ]"
428 @ MSG_SI_LWP              "[ SI_LWP ]"
429 @ MSG_SI_QUEUE            "[ SI_QUEUE ]"
430 @ MSG_SI_TIMER            "[ SI_TIMER ]"
431 @ MSG_SI_ASYNCIO          "[ SI_ASYNCIO ]"
432 @ MSG_SI_MESGQ           "[ SI_MESGQ ]"

434 @ MSG_SI_ILL_ILLOPC       "[ ILL_ILLOPC ]"
435 @ MSG_SI_ILL_ILLOPN       "[ ILL_ILLOPN ]"
436 @ MSG_SI_ILL_ILLADR       "[ ILL_ILLADR ]"
437 @ MSG_SI_ILL_ILLTRP       "[ ILL_ILLTRP ]"
438 @ MSG_SI_ILL_PRVOPC       "[ ILL_PRVOPC ]"
439 @ MSG_SI_ILL_PRVREG       "[ ILL_PRVREG ]"
440 @ MSG_SI_ILL_COPROC       "[ ILL_COPROC ]"
441 @ MSG_SI_ILL_BADSTK       "[ ILL_BADSTK ]"

443 @ MSG_SI_EMT_TAGOVF       "[ EMT_TAGOVF ]"
444 @ MSG_SI_EMT_CPCOVF       "[ EMT_CPCOVF ]"

446 @ MSG_SI_FPE_INTDIV       "[ FPE_INTDIV ]"
447 @ MSG_SI_FPE_INTOVF       "[ FPE_INTOVF ]"
448 @ MSG_SI_FPE_FLTDIV       "[ FPE_FLTDIV ]"
449 @ MSG_SI_FPE_FLTOVF       "[ FPE_FLTOVF ]"
450 @ MSG_SI_FPE_FLTUND       "[ FPE_FLTUND ]"
451 @ MSG_SI_FPE_FLTRES       "[ FPE_FLTRES ]"
452 @ MSG_SI_FPE_FLTINV       "[ FPE_FLTINV ]"
453 @ MSG_SI_FPE_FLTSUB       "[ FPE_FLTSUB ]"

```

```

454 @ MSG_SI_FPE_FLTDEN       "[ FPE_FLTDEN ]"

456 @ MSG_SI_SEGV_MAPERR      "[ SEGV_MAPERR ]"
457 @ MSG_SI_SEGV_ACCERR      "[ SEGV_ACCERR ]"

459 @ MSG_SI_BUS_ADRALN       "[ BUS_ADRALN ]"
460 @ MSG_SI_BUS_ADRERR       "[ BUS_ADRERR ]"
461 @ MSG_SI_BUS_OBJERR       "[ BUS_OBJERR ]"

463 @ MSG_SI_TRAP_BRKPT       "[ TRAP_BRKPT ]"
464 @ MSG_SI_TRAP_TRACE       "[ TRAP_TRACE ]"
465 @ MSG_SI_TRAP_RWATCH      "[ TRAP_RWATCH ]"
466 @ MSG_SI_TRAP_WWATCH      "[ TRAP_WWATCH ]"
467 @ MSG_SI_TRAP_XWATCH      "[ TRAP_XWATCH ]"
468 @ MSG_SI_TRAP_DTRACE      "[ TRAP_DTRACE ]"

470 @ MSG_SI_CLD_EXITED       "[ CLD_EXITED ]"
471 @ MSG_SI_CLD_KILLED       "[ CLD_KILLED ]"
472 @ MSG_SI_CLD_DUMPED       "[ CLD_DUMPED ]"
473 @ MSG_SI_CLD_TRAPPED      "[ CLD_TRAPPED ]"
474 @ MSG_SI_CLD_STOPPED      "[ CLD_STOPPED ]"
475 @ MSG_SI_CLD_CONTINUED    "[ CLD_CONTINUED ]"

477 @ MSG_SI_POLL_IN          "[ POLL_IN ]"
478 @ MSG_SI_POLL_OUT         "[ POLL_OUT ]"
479 @ MSG_SI_POLL_MSG         "[ POLL_MSG ]"
480 @ MSG_SI_POLL_ERR         "[ POLL_ERR ]"
481 @ MSG_SI_POLL_PRI         "[ POLL_PRI ]"
482 @ MSG_SI_POLL_HUP         "[ POLL_HUP ]"

484 @ MSG_SIGHUP              "[ SIGHUP ]"
485 @ MSG_SIGHUP_ALT          "SIGHUP"
486 @ MSG_SIGINT              "[ SIGINT ]"
487 @ MSG_SIGINT_ALT          "SIGINT"
488 @ MSG_SIGQUIT             "[ SIGQUIT ]"
489 @ MSG_SIGQUIT_ALT         "SIGQUIT"
490 @ MSG_SIGILL              "[ SIGILL ]"
491 @ MSG_SIGILL_ALT          "SIGILL"
492 @ MSG_SIGTRAP             "[ SIGTRAP ]"
493 @ MSG_SIGTRAP_ALT         "SIGTRAP"
494 @ MSG_SIGABRT              "[ SIGABRT ]"
495 @ MSG_SIGABRT_ALT         "SIGABRT"
496 @ MSG_SIGEMT              "[ SIGEMT ]"
497 @ MSG_SIGEMT_ALT          "SIGEMT"
498 @ MSG_SIGFPE              "[ SIGFPE ]"
499 @ MSG_SIGFPE_ALT          "SIGFPE"
500 @ MSG_SIGKILL             "[ SIGKILL ]"
501 @ MSG_SIGKILL_ALT         "SIGKILL"
502 @ MSG_SIGBUS              "[ SIGBUS ]"
503 @ MSG_SIGBUS_ALT          "SIGBUS"
504 @ MSG_SIGSEGV             "[ SIGSEGV ]"
505 @ MSG_SIGSEGV_ALT         "SIGSEGV"
506 @ MSG_SIGSYS              "[ SIGSYS ]"
507 @ MSG_SIGSYS_ALT          "SIGSYS"
508 @ MSG_SIGPIPE             "[ SIGPIPE ]"
509 @ MSG_SIGPIPE_ALT         "SIGPIPE"
510 @ MSG_SIGALRM             "[ SIGALRM ]"
511 @ MSG_SIGALRM_ALT         "SIGALRM"
512 @ MSG_SIGTERM             "[ SIGTERM ]"
513 @ MSG_SIGTERM_ALT         "SIGTERM"
514 @ MSG_SIGUSR1             "[ SIGUSR1 ]"
515 @ MSG_SIGUSR1_ALT         "SIGUSR1"
516 @ MSG_SIGUSR2             "[ SIGUSR2 ]"
517 @ MSG_SIGUSR2_ALT         "SIGUSR2"
518 @ MSG_SIGCHLD             "[ SIGCHLD ]"
519 @ MSG_SIGCHLD_ALT         "SIGCHLD"

```

```

520 @ MSG_SIGPWR           "[ SIGPWR ]"
521 @ MSG_SIGPWR_ALT      "SIGPWR"
522 @ MSG_SIGWINCH        "[ SIGWINCH ]"
523 @ MSG_SIGWINCH_ALT    "SIGWINCH"
524 @ MSG_SIGURG          "[ SIGURG ]"
525 @ MSG_SIGURG_ALT      "SIGURG"
526 @ MSG_SIGPOLL         "[ SIGPOLL ]"
527 @ MSG_SIGPOLL_ALT     "SIGPOLL"
528 @ MSG_SIGSTOP         "[ SIGSTOP ]"
529 @ MSG_SIGSTOP_ALT     "SIGSTOP"
530 @ MSG_SIGTSTP         "[ SIGTSTP ]"
531 @ MSG_SIGTSTP_ALT     "SIGTSTP"
532 @ MSG_SIGCONT         "[ SIGCONT ]"
533 @ MSG_SIGCONT_ALT     "SIGCONT"
534 @ MSG_SIGTTIN         "[ SIGTTIN ]"
535 @ MSG_SIGTTIN_ALT     "SIGTTIN"
536 @ MSG_SIGTTOU         "[ SIGTTOU ]"
537 @ MSG_SIGTTOU_ALT     "SIGTTOU"
538 @ MSG_SIGVTALRM       "[ SIGVTALRM ]"
539 @ MSG_SIGVTALRM_ALT   "SIGVTALRM"
540 @ MSG_SIGPROF          "[ SIGPROF ]"
541 @ MSG_SIGPROF_ALT     "SIGPROF"
542 @ MSG_SIGXCPU         "[ SIGXCPU ]"
543 @ MSG_SIGXCPU_ALT     "SIGXCPU"
544 @ MSG_SIGXFSZ         "[ SIGXFSZ ]"
545 @ MSG_SIGXFSZ_ALT     "SIGXFSZ"
546 @ MSG_SIGWAITING      "[ SIGWAITING ]"
547 @ MSG_SIGWAITING_ALT  "SIGWAITING"
548 @ MSG_SIGLWP          "[ SIGLWP ]"
549 @ MSG_SIGLWP_ALT      "SIGLWP"
550 @ MSG_SIGFREEZE       "[ SIGFREEZE ]"
551 @ MSG_SIGFREEZE_ALT   "SIGFREEZE"
552 @ MSG_SIGTHAW         "[ SIGTHAW ]"
553 @ MSG_SIGTHAW_ALT     "SIGTHAW"
554 @ MSG_SIGCANCEL       "[ SIGCANCEL ]"
555 @ MSG_SIGCANCEL_ALT   "SIGCANCEL"
556 @ MSG_SIGLOST         "[ SIGLOST ]"
557 @ MSG_SIGLOST_ALT     "SIGLOST"
558 @ MSG_SIGXRES         "[ SIGXRES ]"
559 @ MSG_SIGXRES_ALT     "SIGXRES"
560 @ MSG_SIGJVM1         "[ SIGJVM1 ]"
561 @ MSG_SIGJVM1_ALT     "SIGJVM1"
562 @ MSG_SIGJVM2         "[ SIGJVM2 ]"
563 @ MSG_SIGJVM2_ALT     "SIGJVM2"

565 @ MSG_FLTILL          "[ FLTILL ]"
566 @ MSG_FLTILL_ALT      "FLTILL"
567 @ MSG_FLTPRIV         "[ FLTPRIV ]"
568 @ MSG_FLTPRIV_ALT     "FLTPRIV"
569 @ MSG_FLTBPT         "[ FLTBPT ]"
570 @ MSG_FLTBPT_ALT     "FLTBPT"
571 @ MSG_FLTTRACE        "[ FLTTRACE ]"
572 @ MSG_FLTTRACE_ALT    "FLTTRACE"
573 @ MSG_FLTACCESS       "[ FLTACCESS ]"
574 @ MSG_FLTACCESS_ALT   "FLTACCESS"
575 @ MSG_FLTBOUNDS       "[ FLTBOUNDS ]"
576 @ MSG_FLTBOUNDS_ALT   "FLTBOUNDS"
577 @ MSG_FLTIOVF         "[ FLTIOVF ]"
578 @ MSG_FLTIOVF_ALT     "FLTIOVF"
579 @ MSG_FLTIZDIV        "[ FLTIZDIV ]"
580 @ MSG_FLTIZDIV_ALT    "FLTIZDIV"
581 @ MSG_FLTFPE          "[ FLTFPE ]"
582 @ MSG_FLTFPE_ALT     "FLTFPE"
583 @ MSG_FLTSTACK        "[ FLTSTACK ]"
584 @ MSG_FLTSTACK_ALT    "FLTSTACK"
585 @ MSG_FLTPAGE         "[ FLTPAGE ]"

```

```

586 @ MSG_FLTPAGE_ALT    "FLTPAGE"
587 @ MSG_FLTWATCH       "[ FLTWATCH ]"
588 @ MSG_FLTWATCH_ALT   "FLTWATCH"
589 @ MSG_FLTFCOVF       "[ FLTFCOVF ]"
590 @ MSG_FLTFCOVF_ALT   "FLTFCOVF"

592 @ MSG_SYS_EXIT       "[ exit ]"           # 1
593 @ MSG_SYS_EXIT_ALT   "exit"
594 @ MSG_SYS_2          "2"           # 2 (u)
595 @ MSG_SYS_READ       "[ read ]"     # 3
596 @ MSG_SYS_READ_ALT   "read"
597 @ MSG_SYS_WRITE      "[ write ]"    # 4
598 @ MSG_SYS_WRITE_ALT  "write"
599 @ MSG_SYS_OPEN       "[ open ]"     # 5
600 @ MSG_SYS_OPEN_ALT   "open"
601 @ MSG_SYS_CLOSE     "[ close ]"    # 6
602 @ MSG_SYS_CLOSE_ALT "close"
603 @ MSG_SYS_7          "7"           # 7 (u)
604 @ MSG_SYS_8          "8"           # 8 (u)
605 @ MSG_SYS_LINK       "[ link ]"     # 9
606 @ MSG_SYS_LINK_ALT   "link"
607 @ MSG_SYS_UNLINK     "[ unlink ]"   # 10
608 @ MSG_SYS_UNLINK_ALT "unlink"
609 @ MSG_SYS_11         "11"          # 11 (u)
610 @ MSG_SYS_CHDIR      "[ chdir ]"    # 12
611 @ MSG_SYS_CHDIR_ALT  "chdir"
612 @ MSG_SYS_TIME       "[ time ]"     # 13
613 @ MSG_SYS_TIME_ALT   "time"
614 @ MSG_SYS_MKNOD     "[ mknod ]"    # 14
615 @ MSG_SYS_MKNOD_ALT "mknod"
616 @ MSG_SYS_CHMOD     "[ chmod ]"     # 15
617 @ MSG_SYS_CHMOD_ALT "chmod"
618 @ MSG_SYS_CHOWN     "[ chown ]"    # 16
619 @ MSG_SYS_CHOWN_ALT "chown"
620 @ MSG_SYS_BRK       "[ brk ]"      # 17
621 @ MSG_SYS_BRK_ALT   "brk"
622 @ MSG_SYS_STAT      "[ stat ]"      # 18
623 @ MSG_SYS_STAT_ALT  "stat"
624 @ MSG_SYS_LSEEK     "[ lseek ]"    # 19
625 @ MSG_SYS_LSEEK_ALT "lseek"
626 @ MSG_SYS_GETPID    "[ getpid ]"    # 20
627 @ MSG_SYS_GETPID_ALT "getpid"
628 @ MSG_SYS_MOUNT     "[ mount ]"    # 21
629 @ MSG_SYS_MOUNT_ALT "mount"
630 @ MSG_SYS_22        "22"          # 22 (u)
631 @ MSG_SYS_SETUID     "[ setuid ]"   # 23
632 @ MSG_SYS_SETUID_ALT "setuid"
633 @ MSG_SYS_GETUID     "[ getuid ]"   # 24
634 @ MSG_SYS_GETUID_ALT "getuid"
635 @ MSG_SYS_STIME     "[ stime ]"     # 25
636 @ MSG_SYS_STIME_ALT "stime"
637 @ MSG_SYS_PCSAMPLE  "[ pcsample ]"  # 26
638 @ MSG_SYS_PCSAMPLE_ALT "pcsample"
639 @ MSG_SYS_ALARM     "[ alarm ]"     # 27
640 @ MSG_SYS_ALARM_ALT "alarm"
641 @ MSG_SYS_FSTAT     "[ fstat ]"     # 28
642 @ MSG_SYS_FSTAT_ALT "fstat"
643 @ MSG_SYS_PAUSE     "[ pause ]"    # 29
644 @ MSG_SYS_PAUSE_ALT "pause"
645 @ MSG_SYS_30        "30"          # 30 (u)
646 @ MSG_SYS_STTY      "[ stty ]"     # 31
647 @ MSG_SYS_STTY_ALT  "stty"
648 @ MSG_SYS_GTTY      "[ gtty ]"     # 32
649 @ MSG_SYS_GTTY_ALT  "gtty"
650 @ MSG_SYS_ACCESS     "[ access ]"   # 33
651 @ MSG_SYS_ACCESS_ALT "access"

```

```

652 @ MSG_SYS_NICE          "[ nice ]"          # 34
653 @ MSG_SYS_NICE_ALT     "nice"              # 35
654 @ MSG_SYS_STATFS       "[ statfs ]"       # 35
655 @ MSG_SYS_STATFS_ALT   "statfs"            # 35
656 @ MSG_SYS_SYNC         "[ sync ]"          # 36
657 @ MSG_SYS_SYNC_ALT     "sync"              # 36
658 @ MSG_SYS_KILL         "[ kill ]"          # 37
659 @ MSG_SYS_KILL_ALT     "kill"              # 37
660 @ MSG_SYS_FSTATFS      "[ fstatfs ]"     # 38
661 @ MSG_SYS_FSTATFS_ALT  "fstatfs"          # 38
662 @ MSG_SYS_PGRPSYS     "[ pgrpsys ]"     # 39
663 @ MSG_SYS_PGRPSYS_ALT  "pgrpsys"          # 39
664 @ MSG_SYS_UUCOPYSTR    "[ uucopystr ]"   # 40
665 @ MSG_SYS_UUCOPYSTR_ALT "uucopystr"        # 40
666 @ MSG_SYS_41           "41"                        # 41 (u)
667 @ MSG_SYS_PIPE        "[ pipe ]"            # 42
668 @ MSG_SYS_PIPE_ALT     "pipe"                      # 42
669 @ MSG_SYS_TIMES       "[ times ]"           # 43
670 @ MSG_SYS_TIMES_ALT   "times"                # 43
671 @ MSG_SYS_PROFIL      "[ profil ]"          # 44
672 @ MSG_SYS_PROFIL_ALT  "profil"              # 44
673 @ MSG_SYS_FACCESSAT  "[ faccessat ]"   # 45
674 @ MSG_SYS_FACCESSAT_ALT "faccessat"       # 45
675 @ MSG_SYS_SETGID     "[ setgid ]"          # 46
676 @ MSG_SYS_SETGID_ALT "setgid"            # 46
677 @ MSG_SYS_GETGID     "[ getgid ]"          # 47
678 @ MSG_SYS_GETGID_ALT "getgid"            # 47
679 @ MSG_SYS_48          "48"                        # 48 (u)
680 @ MSG_SYS_MSGSYS     "[ msgsys ]"          # 49
681 @ MSG_SYS_MSGSYS_ALT "msgsys"                # 49
682 @ MSG_SYS_SYSI86     "[ sysi86 ]"          # 50
683 @ MSG_SYS_SYSI86_ALT "sysi86"              # 50
684 @ MSG_SYS_ACCT       "[ acct ]"            # 51
685 @ MSG_SYS_ACCT_ALT   "acct"                  # 51
686 @ MSG_SYS_SHMSYS    "[ shmsys ]"          # 52
687 @ MSG_SYS_SHMSYS_ALT "shmsys"              # 52
688 @ MSG_SYS_SEMSYS    "[ semsys ]"          # 53
689 @ MSG_SYS_SEMSYS_ALT "semsys"              # 53
690 @ MSG_SYS_IOCTL     "[ ioctl ]"           # 54
691 @ MSG_SYS_IOCTL_ALT "ioctl"                # 54
692 @ MSG_SYS_UADMIN    "[ uadmin ]"          # 55
693 @ MSG_SYS_UADMIN_ALT "uadmin"              # 55
694 @ MSG_SYS_FCHOWNAT  "[ fchownat ]"        # 56
695 @ MSG_SYS_FCHOWNAT_ALT "fchownat"         # 56
696 @ MSG_SYS_UTSSYS   "[ utssys ]"          # 57
697 @ MSG_SYS_UTSSYS_ALT "utssys"              # 57
698 @ MSG_SYS_FDSYNC   "[ fdsync ]"          # 58
699 @ MSG_SYS_FDSYNC_ALT "fdsync"              # 58
700 @ MSG_SYS_EXECEVE   "[ execve ]"          # 59
701 @ MSG_SYS_EXECEVE_ALT "execve"            # 59
702 @ MSG_SYS_UMASK     "[ umask ]"           # 60
703 @ MSG_SYS_UMASK_ALT "umask"                # 60
704 @ MSG_SYS_CHROOT    "[ chroot ]"          # 61
705 @ MSG_SYS_CHROOT_ALT "chroot"              # 61
706 @ MSG_SYS_FCNTL     "[ fcntl ]"           # 62
707 @ MSG_SYS_FCNTL_ALT "fcntl"                # 62
708 @ MSG_SYS_ULIMIT    "[ ulimit ]"          # 63
709 @ MSG_SYS_ULIMIT_ALT "ulimit"              # 63
710 @ MSG_SYS_RENAMEAT  "[ renameat ]"        # 64
711 @ MSG_SYS_RENAMEAT_ALT "renameat"          # 64
712 @ MSG_SYS_UNLINKAT  "[ unlinkat ]"        # 65
713 @ MSG_SYS_UNLINKAT_ALT "unlinkat"          # 65
714 @ MSG_SYS_FSTATAT   "[ fstatat ]"         # 66
715 @ MSG_SYS_FSTATAT_ALT "fstatat"           # 66
716 @ MSG_SYS_FSTATAT64 "[ fstatat64 ]"   # 67
717 @ MSG_SYS_FSTATAT64_ALT "fstatat64"       # 67

```

```

718 @ MSG_SYS_OPENAT     "[ openat ]"        # 68
719 @ MSG_SYS_OPENAT_ALT "openat"              # 68
720 @ MSG_SYS_OPENAT64   "[ openat64 ]"       # 69
721 @ MSG_SYS_OPENAT64_ALT "openat64"         # 69
722 @ MSG_SYS_TASKSYS    "[ tasksys ]"         # 70
723 @ MSG_SYS_TASKSYS_ALT "tasksys"            # 70
724 @ MSG_SYS_ACCTCTL    "[ acctctl ]"         # 71
725 @ MSG_SYS_ACCTCTL_ALT "acctctl"            # 71
726 @ MSG_SYS_EXACCTSYS "[ exacctsyz ]"        # 72
727 @ MSG_SYS_EXACCTSYS_ALT "exacctsyz"        # 72
728 @ MSG_SYS_GETPAGESIZES "[ getpagesizes ]" # 73
729 @ MSG_SYS_GETPAGESIZES_ALT "getpagesizes" # 73
730 @ MSG_SYS_RCTLSYS    "[ rctlsys ]"         # 74
731 @ MSG_SYS_RCTLSYS_ALT "rctlsys"            # 74
732 @ MSG_SYS_SIDSYS     "[ sidsys ]"          # 75
733 @ MSG_SYS_SIDSYS_ALT "sidsys"             # 75
734 @ MSG_SYS_76         "76"                  # 76 (u)
735 @ MSG_SYS_LWP_PARK  "[ lwp_park ]"        # 77
736 @ MSG_SYS_LWP_PARK_ALT "lwp_park"         # 77
737 @ MSG_SYS_SENDFILEV "[ sendfilev ]"    # 78
738 @ MSG_SYS_SENDFILEV_ALT "sendfilev"       # 78
739 @ MSG_SYS_RMDIR     "[ rmdir ]"           # 79
740 @ MSG_SYS_RMDIR_ALT "rmdir"                # 79
741 @ MSG_SYS_MKDIR     "[ mkdir ]"           # 80
742 @ MSG_SYS_MKDIR_ALT "mkdir"                # 80
743 @ MSG_SYS_GETDENTS  "[ getdents ]"        # 81
744 @ MSG_SYS_GETDENTS_ALT "getdents"         # 81
745 @ MSG_SYS_PRIVSYS  "[ privsys ]"          # 82
746 @ MSG_SYS_PRIVSYS_ALT "privsys"           # 82
747 @ MSG_SYS_UCREDSYS "[ ucredsys ]"        # 83
748 @ MSG_SYS_UCREDSYS_ALT "ucredsys"         # 83
749 @ MSG_SYS_SYSFS     "[ sysfs ]"           # 84
750 @ MSG_SYS_SYSFS_ALT "sysfs"                # 84
751 @ MSG_SYS_GETMSG    "[ getmsg ]"          # 85
752 @ MSG_SYS_GETMSG_ALT "getmsg"              # 85
753 @ MSG_SYS_PUTMSG    "[ putmsg ]"          # 86
754 @ MSG_SYS_PUTMSG_ALT "putmsg"              # 86
755 @ MSG_SYS_87        "87"                  # 87 (u)
756 @ MSG_SYS_LSTAT     "[ lstat ]"           # 88
757 @ MSG_SYS_LSTAT_ALT "lstat"                # 88
758 @ MSG_SYS_SYMLINK  "[ symlink ]"         # 89
759 @ MSG_SYS_SYMLINK_ALT "symlink"            # 89
760 @ MSG_SYS_READLINK "[ readlink ]"        # 90
761 @ MSG_SYS_READLINK_ALT "readlink"         # 90
762 @ MSG_SYS_SETGROUPS "[ setgroups ]"       # 91
763 @ MSG_SYS_SETGROUPS_ALT "setgroups"        # 91
764 @ MSG_SYS_GETGROUPS "[ getgroups ]"       # 92
765 @ MSG_SYS_GETGROUPS_ALT "getgroups"        # 92
766 @ MSG_SYS_FCHMOD    "[ fchmod ]"          # 93
767 @ MSG_SYS_FCHMOD_ALT "fchmod"              # 93
768 @ MSG_SYS_FCHOWN    "[ fchown ]"         # 94
769 @ MSG_SYS_FCHOWN_ALT "fchown"             # 94
770 @ MSG_SYS_SIGPROCMASK "[ sigprocmask ]" # 95
771 @ MSG_SYS_SIGPROCMASK_ALT "sigprocmask"    # 95
772 @ MSG_SYS_SIGSUSPEND "[ sigsuspend ]"     # 96
773 @ MSG_SYS_SIGSUSPEND_ALT "sigsuspend"      # 96
774 @ MSG_SYS_SIGALTSTACK "[ sigaltstack ]"   # 97
775 @ MSG_SYS_SIGALTSTACK_ALT "sigaltstack"    # 97
776 @ MSG_SYS_SIGACTION "[ sigaction ]"       # 98
777 @ MSG_SYS_SIGACTION_ALT "sigaction"        # 98
778 @ MSG_SYS_SIGPENDING "[ sigpending ]"     # 99
779 @ MSG_SYS_SIGPENDING_ALT "sigpending"      # 99
780 @ MSG_SYS_CONTEXT   "[ context ]"         # 100
781 @ MSG_SYS_CONTEXT_ALT "context"           # 100
782 @ MSG_SYS_101       "101"                 # 101(u)
783 @ MSG_SYS_102       "102"                 # 102(u)

```

```

784 @ MSG_SYS_STATVFS      "[ statvfs ]" # 103
785 @ MSG_SYS_STATVFS_ALT  "statvfs"
786 @ MSG_SYS_FSTATVFS     "[ fstatvfs ]" # 104
787 @ MSG_SYS_FSTATVFS_ALT "fstatvfs"
788 @ MSG_SYS_GETLOADAVG   "[ getloadavg ]" # 105
789 @ MSG_SYS_GETLOADAVG_ALT "getloadavg"
790 @ MSG_SYS_NFSSYS       "[ nfssys ]" # 106
791 @ MSG_SYS_NFSSYS_ALT   "nfssys"
792 @ MSG_SYS_WAITID       "[ waitid ]" # 107
793 @ MSG_SYS_WAITID_ALT   "waitid"
794 @ MSG_SYS_SIGSENDSYS   "[ sigsendsys ]" # 108
795 @ MSG_SYS_SIGSENDSYS_ALT "sigsendsys"
796 @ MSG_SYS_HRTSYS       "[ hrtsys ]" # 109
797 @ MSG_SYS_HRTSYS_ALT   "hrtsys"
798 @ MSG_SYS_UTIMESYS     "[ utimesys ]" # 110
799 @ MSG_SYS_UTIMESYS_ALT "utimesys"
800 @ MSG_SYS_SIGRESEND    "[ sigresend ]" # 111
801 @ MSG_SYS_SIGRESEND_ALT "sigresend"
802 @ MSG_SYS_PRIOCNLSYS   "[ pricnlsys ]" # 112
803 @ MSG_SYS_PRIOCNLSYS_ALT "pricnlsys"
804 @ MSG_SYS_PATHCONF     "[ pathconf ]" # 113
805 @ MSG_SYS_PATHCONF_ALT "pathconf"
806 @ MSG_SYS_MINCORE     "[ mincore ]" # 114
807 @ MSG_SYS_MINCORE_ALT "mincore"
808 @ MSG_SYS_MMAP         "[ mmap ]" # 115
809 @ MSG_SYS_MMAP_ALT     "mmap"
810 @ MSG_SYS_MPROTECT     "[ mprotect ]" # 116
811 @ MSG_SYS_MPROTECT_ALT "mprotect"
812 @ MSG_SYS_MUNMAP       "[ munmap ]" # 117
813 @ MSG_SYS_MUNMAP_ALT   "munmap"
814 @ MSG_SYS_FPATHCONF    "[ fpathconf ]" # 118
815 @ MSG_SYS_FPATHCONF_ALT "fpathconf"
816 @ MSG_SYS_VFORK        "[ vfork ]" # 119
817 @ MSG_SYS_VFORK_ALT    "vfork"
818 @ MSG_SYS_FCHDIR       "[ fchdir ]" # 120
819 @ MSG_SYS_FCHDIR_ALT   "fchdir"
820 @ MSG_SYS_READV        "[ readv ]" # 121
821 @ MSG_SYS_READV_ALT    "readv"
822 @ MSG_SYS_WRITEV       "[ writev ]" # 122
823 @ MSG_SYS_WRITEV_ALT   "writev"
824 @ MSG_SYS_123          "123" # 123(u)
825 @ MSG_SYS_124          "124" # 124(u)
826 @ MSG_SYS_125          "125" # 125(u)
827 @ MSG_SYS_126          "126" # 126(u)
828 @ MSG_SYS_MMAPOBJ      "[ mmapobj ]" # 127
829 @ MSG_SYS_MMAPOBJ_ALT  "mmapobj"
830 @ MSG_SYS_SETRLIMIT    "[ setrlimit ]" # 128
831 @ MSG_SYS_SETRLIMIT_ALT "setrlimit"
832 @ MSG_SYS_GETRLIMIT    "[ getrlimit ]" # 129
833 @ MSG_SYS_GETRLIMIT_ALT "getrlimit"
834 @ MSG_SYS_LCHOWN       "[ lchown ]" # 130
835 @ MSG_SYS_LCHOWN_ALT   "lchown"
836 @ MSG_SYS_MEMCNTL     "[ memcntl ]" # 131
837 @ MSG_SYS_MEMCNTL_ALT "memcntl"
838 @ MSG_SYS_GETPMSG      "[ getpmsg ]" # 132
839 @ MSG_SYS_GETPMSG_ALT  "getpmsg"
840 @ MSG_SYS_PUTPMSG      "[ putpmsg ]" # 133
841 @ MSG_SYS_PUTPMSG_ALT  "putpmsg"
842 @ MSG_SYS_RENAME       "[ rename ]" # 134
843 @ MSG_SYS_RENAME_ALT   "rename"
844 @ MSG_SYS_UNAME        "[ uname ]" # 135
845 @ MSG_SYS_UNAME_ALT    "uname"
846 @ MSG_SYS_SETEGID     "[ setegid ]" # 136
847 @ MSG_SYS_SETEGID_ALT "setegid"
848 @ MSG_SYS_SYSCONFIG    "[ sysconfig ]" # 137
849 @ MSG_SYS_SYSCONFIG_ALT "sysconfig"

```

```

850 @ MSG_SYS_ADJTIME      "[ adjtime ]" # 138
851 @ MSG_SYS_ADJTIME_ALT  "adjtime"
852 @ MSG_SYS_SYSTEMINFO   "[ systeminfo ]" # 139
853 @ MSG_SYS_SYSTEMINFO_ALT "systeminfo"
854 @ MSG_SYS_SHAREFS      "[ sharefs ]" # 140
855 @ MSG_SYS_SHAREFS_ALT  "sharefs"
856 @ MSG_SYS_SETEUID      "[ seteuid ]" # 141
857 @ MSG_SYS_SETEUID_ALT  "seteuid"
858 @ MSG_SYS_FORKSYS      "[ forksys ]" # 142
859 @ MSG_SYS_FORKSYS_ALT  "forksys"
860 @ MSG_SYS_143          "143" # 143(u)
861 @ MSG_SYS_SIGTIMEDWAIT "[ sigtimedwait ]" # 144
862 @ MSG_SYS_SIGTIMEDWAIT_ALT "sigtimedwait"
863 @ MSG_SYS_LWP_INFO     "[ lwp_info ]" # 145
864 @ MSG_SYS_LWP_INFO_ALT "lwp_info"
865 @ MSG_SYS_YIELD       "[ yield ]" # 146
866 @ MSG_SYS_YIELD_ALT   "yield"
867 @ MSG_SYS_147          "147" # 147(u)
868 @ MSG_SYS_LWP_SEMA_POST "[ lwp_sema_post ]" # 148
869 @ MSG_SYS_LWP_SEMA_POST_ALT "lwp_sema_post"
870 @ MSG_SYS_LWP_SEMA_TRYWAIT "[ lwp_sema_trywait ]" # 149
871 @ MSG_SYS_LWP_SEMA_TRYWAIT_ALT "lwp_sema_trywait"
872 @ MSG_SYS_LWP_DETACH   "[ lwp_detach ]" # 150
873 @ MSG_SYS_LWP_DETACH_ALT "lwp_detach"
874 @ MSG_SYS_CORECTL     "[ corectl ]" # 151
875 @ MSG_SYS_CORECTL_ALT "corectl"
876 @ MSG_SYS_MODCTL      "[ modctl ]" # 152
877 @ MSG_SYS_MODCTL_ALT  "modctl"
878 @ MSG_SYS_FCHROOT     "[ fchroot ]" # 153
879 @ MSG_SYS_FCHROOT_ALT "fchroot"
880 @ MSG_SYS_154          "154" # 154(u)
881 @ MSG_SYS_VHANGUP     "[ vhangup ]" # 155
882 @ MSG_SYS_VHANGUP_ALT "vhangup"
883 @ MSG_SYS_GETTIMEOFDAY "[ gettimeofday ]" # 156
884 @ MSG_SYS_GETTIMEOFDAY_ALT "gettimeofday"
885 @ MSG_SYS_GETTITIMER   "[ getitimer ]" # 157
886 @ MSG_SYS_GETTITIMER_ALT "getitimer"
887 @ MSG_SYS_SETTITIMER   "[ setitimer ]" # 158
888 @ MSG_SYS_SETTITIMER_ALT "setitimer"
889 @ MSG_SYS_LWP_CREATE   "[ lwp_create ]" # 159
890 @ MSG_SYS_LWP_CREATE_ALT "lwp_create"
891 @ MSG_SYS_LWP_EXIT     "[ lwp_exit ]" # 160
892 @ MSG_SYS_LWP_EXIT_ALT "lwp_exit"
893 @ MSG_SYS_LWP_SUSPEND  "[ lwp_suspend ]" # 161
894 @ MSG_SYS_LWP_SUSPEND_ALT "lwp_suspend"
895 @ MSG_SYS_LWP_CONTINUE "[ lwp_continue ]" # 162
896 @ MSG_SYS_LWP_CONTINUE_ALT "lwp_continue"
897 @ MSG_SYS_LWP_KILL     "[ lwp_kill ]" # 163
898 @ MSG_SYS_LWP_KILL_ALT "lwp_kill"
899 @ MSG_SYS_LWP_SELF     "[ lwp_self ]" # 164
900 @ MSG_SYS_LWP_SELF_ALT "lwp_self"
901 @ MSG_SYS_LWP_SIGMASK  "[ lwp_sigmask ]" # 165
902 @ MSG_SYS_LWP_SIGMASK_ALT "lwp_sigmask"
903 @ MSG_SYS_LWP_PRIVATE  "[ lwp_private ]" # 166
904 @ MSG_SYS_LWP_PRIVATE_ALT "lwp_private"
905 @ MSG_SYS_LWP_WAIT     "[ lwp_wait ]" # 167
906 @ MSG_SYS_LWP_WAIT_ALT "lwp_wait"
907 @ MSG_SYS_LWP_MUTEX_WAKEUP "[ lwp_mutex_wakeup ]" # 168
908 @ MSG_SYS_LWP_MUTEX_WAKEUP_ALT "lwp_mutex_wakeup"
909 @ MSG_SYS_169          "169" # 169(u)
910 @ MSG_SYS_LWP_COND_WAIT "[ lwp_cond_wait ]" # 170
911 @ MSG_SYS_LWP_COND_WAIT_ALT "lwp_cond_wait"
912 @ MSG_SYS_LWP_COND_SIGNAL "[ lwp_cond_signal ]" # 171
913 @ MSG_SYS_LWP_COND_SIGNAL_ALT "lwp_cond_signal"
914 @ MSG_SYS_LWP_COND_BROADCAST "[ lwp_cond_broadcast ]" # 172
915 @ MSG_SYS_LWP_COND_BROADCAST_ALT "lwp_cond_broadcast"

```



```

916 @ MSG_SYS_PREAD          "[ pread ]"          # 173
917 @ MSG_SYS_PREAD_ALT     "pread"              #
918 @ MSG_SYS_PWRITE        "[ pwrite ]"          # 174
919 @ MSG_SYS_PWRITE_ALT    "pwrite"              #
920 @ MSG_SYS_LLSEEK        "[ llseek ]"          # 175
921 @ MSG_SYS_LLSEEK_ALT    "llseek"              #
922 @ MSG_SYS_INST_SYNC     "[ inst_sync ]"        # 176
923 @ MSG_SYS_INST_SYNC_ALT "inst_sync"          #
924 @ MSG_SYS_BRAND         "[ brand ]"          # 177
925 @ MSG_SYS_BRAND_ALT     "brand"              #
926 @ MSG_SYS_KAIO          "[ kaio ]"           # 178
927 @ MSG_SYS_KAIO_ALT      "kaio"              #
928 @ MSG_SYS_CPC           "[ cpc ]"           # 179
929 @ MSG_SYS_CPC_ALT       "cpc"              #
930 @ MSG_SYS_LGRPSYS       "[ lgrpsys ]"        # 180
931 @ MSG_SYS_LGRPSYS_ALT   "lgrpsys"          #
932 @ MSG_SYS_RUSAGESYS     "[ rusagesys ]"       # 181
933 @ MSG_SYS_RUSAGESYS_ALT "rusagesys"        #
934 @ MSG_SYS_PORT          "[ port ]"          # 182
935 @ MSG_SYS_PORT_ALT      "port"              #
936 @ MSG_SYS_POLLSYS       "[ pollsys ]"        # 183
937 @ MSG_SYS_POLLSYS_ALT   "pollsys"          #
938 @ MSG_SYS_LABELSYS      "[ labelsys ]"       # 184
939 @ MSG_SYS_LABELSYS_ALT   "labelsys"         #
940 @ MSG_SYS_ACL           "[ acl ]"           # 185
941 @ MSG_SYS_ACL_ALT       "acl"              #
942 @ MSG_SYS_AUDITSYS      "[ auditsys ]"       # 186
943 @ MSG_SYS_AUDITSYS_ALT  "auditsys"         #
944 @ MSG_SYS_PROCESSOR_BIND "[ processor_bind ]" # 187
945 @ MSG_SYS_PROCESSOR_BIND_ALT "processor_bind" #
946 @ MSG_SYS_PROCESSOR_INFO "[ processor_info ]" # 188
947 @ MSG_SYS_PROCESSOR_INFO_ALT "processor_info" #
948 @ MSG_SYS_P_ONLINE      "[ p_online ]"       # 189
949 @ MSG_SYS_P_ONLINE_ALT  "p_online"         #
950 @ MSG_SYS_SIGQUEUE      "[ sigqueue ]"       # 190
951 @ MSG_SYS_SIGQUEUE_ALT  "sigqueue"         #
952 @ MSG_SYS_CLOCK_GETTIME "[ clock_gettime ]" # 191
953 @ MSG_SYS_CLOCK_GETTIME_ALT "clock_gettime" #
954 @ MSG_SYS_CLOCK_SETTIME "[ clock_settime ]" # 192
955 @ MSG_SYS_CLOCK_SETTIME_ALT "clock_settime" #
956 @ MSG_SYS_CLOCK_GETRES  "[ clock_getres ]" # 193
957 @ MSG_SYS_CLOCK_GETRES_ALT "clock_getres" #
958 @ MSG_SYS_TIMER_CREATE  "[ timer_create ]" # 194
959 @ MSG_SYS_TIMER_CREATE_ALT "timer_create" #
960 @ MSG_SYS_TIMER_DELETE  "[ timer_delete ]" # 195
961 @ MSG_SYS_TIMER_DELETE_ALT "timer_delete" #
962 @ MSG_SYS_TIMER_SETTIME "[ timer_settime ]" # 196
963 @ MSG_SYS_TIMER_SETTIME_ALT "timer_settime" #
964 @ MSG_SYS_TIMER_GETTIME "[ timer_gettime ]" # 197
965 @ MSG_SYS_TIMER_GETTIME_ALT "timer_gettime" #
966 @ MSG_SYS_TIMER_GETOVERRUN "[ timer_getoverrun ]" # 198
967 @ MSG_SYS_TIMER_GETOVERRUN_ALT "timer_getoverrun" #
968 @ MSG_SYS_NANOSLEEP     "[ nanosleep ]"   # 199
969 @ MSG_SYS_NANOSLEEP_ALT "nanosleep"       #
970 @ MSG_SYS_FACL          "[ facl ]"         # 200
971 @ MSG_SYS_FACL_ALT      "facl"           #
972 @ MSG_SYS_DOOR          "[ door ]"         # 201
973 @ MSG_SYS_DOOR_ALT      "door"           #
974 @ MSG_SYS_SETREUID      "[ setreuid ]"     # 202
975 @ MSG_SYS_SETREUID_ALT  "setreuid"       #
976 @ MSG_SYS_SETREGID      "[ setregid ]"     # 203
977 @ MSG_SYS_SETREGID_ALT  "setregid"       #
978 @ MSG_SYS_INSTALL_UTRAP "[ install_utrap ]" # 204
979 @ MSG_SYS_INSTALL_UTRAP_ALT "install_utrap" #
980 @ MSG_SYS_SIGNOTIFY     "[ signotify ]"   # 205
981 @ MSG_SYS_SIGNOTIFY_ALT "signotify"       #

```

```

982 @ MSG_SYS_SCHEDCTL      "[ schedctl ]"     # 206
983 @ MSG_SYS_SCHEDCTL_ALT  "schedctl"        #
984 @ MSG_SYS_PSET          "[ pset ]"         # 207
985 @ MSG_SYS_PSET_ALT     "pset"           #
986 @ MSG_SYS_SPARC_UTRAP_INSTALL "[ sparc_utrap_install ]" # 208
987 @ MSG_SYS_SPARC_UTRAP_INSTALL_ALT "sparc_utrap_install" #
988 @ MSG_SYS_RESOLVEPATH  "[ resolvepath ]"  # 209
989 @ MSG_SYS_RESOLVEPATH_ALT "resolvepath" #
990 @ MSG_SYS_LWP_MUTEX_TIMEDLOCK "[ lwp_mutex_timedlock ]" # 210
991 @ MSG_SYS_LWP_MUTEX_TIMEDLOCK_ALT "lwp_mutex_timedlock" #
992 @ MSG_SYS_LWP_SEMA_TIMEDWAIT "[ lwp_sema_timedwait ]" # 211
993 @ MSG_SYS_LWP_SEMA_TIMEDWAIT_ALT "lwp_sema_timedwait" #
994 @ MSG_SYS_LWP_RWLOCK_SYS "[ lwp_rwlock_sys ]" # 212
995 @ MSG_SYS_LWP_RWLOCK_SYS_ALT "lwp_rwlock_sys" #
996 @ MSG_SYS_GETDENTS64   "[ getdents64 ]"   # 213
997 @ MSG_SYS_GETDENTS64_ALT "getdents64" #
998 @ MSG_SYS_MMAP64       "[ mmap64 ]"       # 214
999 @ MSG_SYS_MMAP64_ALT   "mmap64"         #
1000 @ MSG_SYS_STAT64      "[ stat64 ]"       # 215
1001 @ MSG_SYS_STAT64_ALT  "stat64"         #
1002 @ MSG_SYS_LSTAT64     "[ lstat64 ]"      # 216
1003 @ MSG_SYS_LSTAT64_ALT "lstat64"        #
1004 @ MSG_SYS_FSTAT64     "[ fstat64 ]"      # 217
1005 @ MSG_SYS_FSTAT64_ALT "fstat64"        #
1006 @ MSG_SYS_STATVFS64   "[ statvfs64 ]"    # 218
1007 @ MSG_SYS_STATVFS64_ALT "statvfs64" #
1008 @ MSG_SYS_FSTATVFS64 "[ fstatvfs64 ]"   # 219
1009 @ MSG_SYS_FSTATVFS64_ALT "fstatvfs64" #
1010 @ MSG_SYS_SETRLIMIT64 "[ setrlimit64 ]" # 220
1011 @ MSG_SYS_SETRLIMIT64_ALT "setrlimit64" #
1012 @ MSG_SYS_GETRLIMIT64 "[ getrlimit64 ]" # 221
1013 @ MSG_SYS_GETRLIMIT64_ALT "getrlimit64" #
1014 @ MSG_SYS_PREAD64     "[ pread64 ]"     # 222
1015 @ MSG_SYS_PREAD64_ALT "pread64"        #
1016 @ MSG_SYS_PWRITE64    "[ pwrite64 ]"    # 223
1017 @ MSG_SYS_PWRITE64_ALT "pwrite64" #
1018 @ MSG_SYS_224         "224"             # 224(u)
1019 @ MSG_SYS_OPEN64      "[ open64 ]"       # 225
1020 @ MSG_SYS_OPEN64_ALT  "open64"         #
1021 @ MSG_SYS_RPCSYS      "[ rpcsys ]"       # 226
1022 @ MSG_SYS_RPCSYS_ALT  "rpcsys"         #
1023 @ MSG_SYS_ZONE        "[ zone ]"         # 227
1024 @ MSG_SYS_ZONE_ALT   "zone"           #
1025 @ MSG_SYS_AUTOFSSYS  "[ autofssys ]"    # 228
1026 @ MSG_SYS_AUTOFSSYS_ALT "autofssys" #
1027 @ MSG_SYS_GETCWD     "[ getcwd ]"       # 229
1028 @ MSG_SYS_GETCWD_ALT "getcwd"         #
1029 @ MSG_SYS_SO_SOCKET  "[ so_socket ]"    # 230
1030 @ MSG_SYS_SO_SOCKET_ALT "so_socket" #
1031 @ MSG_SYS_SO_SOCKETPAIR "[ so_socketpair ]" # 231
1032 @ MSG_SYS_SO_SOCKETPAIR_ALT "so_socketpair" #
1033 @ MSG_SYS_BIND       "[ bind ]"         # 232
1034 @ MSG_SYS_BIND_ALT   "bind"           #
1035 @ MSG_SYS_LISTEN     "[ listen ]"       # 233
1036 @ MSG_SYS_LISTEN_ALT "listen"         #
1037 @ MSG_SYS_ACCEPT     "[ accept ]"       # 234
1038 @ MSG_SYS_ACCEPT_ALT "accept"         #
1039 @ MSG_SYS_CONNECT    "[ connect ]"      # 235
1040 @ MSG_SYS_CONNECT_ALT "connect"        #
1041 @ MSG_SYS_SHUTDOWN    "[ shutdown ]"    # 236
1042 @ MSG_SYS_SHUTDOWN_ALT "shutdown" #
1043 @ MSG_SYS_RECV        "[ recv ]"         # 237
1044 @ MSG_SYS_RECV_ALT    "recv"           #
1045 @ MSG_SYS_RECVFROM    "[ recvfrom ]"    # 238
1046 @ MSG_SYS_RECVFROM_ALT "recvfrom" #
1047 @ MSG_SYS_RECVMSG     "[ recvmsg ]"     # 239

```

```

1048 @ MSG_SYS_RECVMSG_ALT      "recvmsg"
1049 @ MSG_SYS_SEND              "[ send ]"                # 240
1050 @ MSG_SYS_SEND_ALT          "send"
1051 @ MSG_SYS_SENDDMSG          "[ sendmsg ]"            # 241
1052 @ MSG_SYS_SENDDMSG_ALT     "sendmsg"
1053 @ MSG_SYS_SENDTO            "[ sendto ]"             # 242
1054 @ MSG_SYS_SENDTO_ALT       "sendto"
1055 @ MSG_SYS_GETPEERNAME       "[ getpeername ]"       # 243
1056 @ MSG_SYS_GETPEERNAME_ALT  "getpeername"
1057 @ MSG_SYS_GETSOCKNAME       "[ getsockname ]"      # 244
1058 @ MSG_SYS_GETSOCKNAME_ALT  "getsockname"
1059 @ MSG_SYS_GETSOCKOPT        "[ getsockopt ]"       # 245
1060 @ MSG_SYS_GETSOCKOPT_ALT    "getsockopt"
1061 @ MSG_SYS_SETSOCKOPT        "[ setsockopt ]"       # 246
1062 @ MSG_SYS_SETSOCKOPT_ALT   "setsockopt"
1063 @ MSG_SYS_SOCKCONFIG        "[ sockconfig ]"      # 247
1064 @ MSG_SYS_SOCKCONFIG_ALT    "sockconfig"
1065 @ MSG_SYS_NTP_GETTIME       "[ ntp_gettime ]"      # 248
1066 @ MSG_SYS_NTP_GETTIME_ALT   "ntp_gettime"
1067 @ MSG_SYS_NTP_ADJTIME       "[ ntp_adjtime ]"     # 249
1068 @ MSG_SYS_NTP_ADJTIME_ALT   "ntp_adjtime"
1069 @ MSG_SYS_LWP_MUTEX_UNLOCK  "[ lwp_mutex_unlock ]" # 250
1070 @ MSG_SYS_LWP_MUTEX_UNLOCK_ALT "lwp_mutex_unlock"
1071 @ MSG_SYS_LWP_MUTEX_TRYLOCK "[ lwp_mutex_trylock ]" # 251
1072 @ MSG_SYS_LWP_MUTEX_TRYLOCK_ALT "lwp_mutex_trylock"
1073 @ MSG_SYS_LWP_MUTEX_REGISTER "[ lwp_mutex_register ]" # 252
1074 @ MSG_SYS_LWP_MUTEX_REGISTER_ALT "lwp_mutex_register"
1075 @ MSG_SYS_CLADM            "[ cladm ]"              # 253
1076 @ MSG_SYS_CLADM_ALT        "cladm"
1077 @ MSG_SYS_UUCOPY           "[ uucopy ]"            # 254
1078 @ MSG_SYS_UUCOPY_ALT       "uucopy"
1079 @ MSG_SYS_UMOUNT2          "[ umount2 ]"          # 255
1080 @ MSG_SYS_UMOUNT2_ALT     "umount2"

1082 @ MSG_PR_O_RDONLY         "O_RDONLY"
1083 @ MSG_PR_O_WRONLY         "O_WRONLY"
1084 @ MSG_PR_O_RDWR           "O_RDWR"
1085 @ MSG_PR_O_SEARCH         "O_SEARCH"
1086 @ MSG_PR_O_EXEC            "O_EXEC"
1087 @ MSG_PR_O_NDELAY         "O_NDELAY"
1088 @ MSG_PR_O_NONBLOCK       "O_NONBLOCK"
1089 @ MSG_PR_O_APPEND         "O_APPEND"
1090 @ MSG_PR_O_SYNC            "O_SYNC"
1091 @ MSG_PR_O_DSYNC           "O_DSYNC"
1092 @ MSG_PR_O_RSYNC           "O_RSYNC"
1093 @ MSG_PR_O_CREAT           "O_CREAT"
1094 @ MSG_PR_O_TRUNC           "O_TRUNC"
1095 @ MSG_PR_O_EXCL            "O_EXCL"
1096 @ MSG_PR_O_NOCTTY         "O_NOCTTY"
1097 @ MSG_PR_O_LARGEFILE      "O_LARGEFILE"
1098 @ MSG_PR_O_XATTR           "O_XATTR"
1099 @ MSG_PR_O_NOFOLLOW       "O_NOFOLLOW"
1100 @ MSG_PR_O_NOLINKS        "O_NOLINKS"

1102 @ MSG_S_IFIFO             "S_IFIFO"
1103 @ MSG_S_IFCHR             "S_IFCHR"
1104 @ MSG_S_IFDIR             "S_IFDIR"
1105 @ MSG_S_IFNAM             "S_IFNAM"
1106 @ MSG_S_IFBLK             "S_IFBLK"
1107 @ MSG_S_IFREG             "S_IFREG"
1108 @ MSG_S_IFLNK             "S_IFLNK"
1109 @ MSG_S_IFSOCK            "S_IFSOCK"
1110 @ MSG_S_IFDOOR            "S_IFDOOR"
1111 @ MSG_S_IFPORT            "S_IFPORT"
1112 @ MSG_S_ISUID             "S_ISUID"
1113 @ MSG_S_ISGID             "S_ISGID"

```

```

1114 @ MSG_S_ISVTX            "S_ISVTX"
1115 @ MSG_S_IRUSR             "S_IRUSR"
1116 @ MSG_S_IWUSR            "S_IWUSR"
1117 @ MSG_S_IXUSR            "S_IXUSR"
1118 @ MSG_S_IRGRP            "S_IRGRP"
1119 @ MSG_S_IWGRP            "S_IWGRP"
1120 @ MSG_S_IXGRP            "S_IXGRP"
1121 @ MSG_S_IROTH            "S_IROTH"
1122 @ MSG_S_IWOTH            "S_IWOTH"
1123 @ MSG_S_IXOTH            "S_IXOTH"

1125 @ MSG_GBL_ZERO          "0"

1127 @ MSG_FMT_INT            "%d"
1128 @ MSG_FMT_WORD           "%u"
1129 @ MSG_FMT_HEXINT         "%#x"

```

```

*****
34856 Thu Jun 30 22:01:29 2016
new/usr/src/cmd/sgs/libconv/common/dynamic.c
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
    unchanged_portion_omitted

419 const conv_ds_t **
420 conv_dyn_tag_strings(conv_iter_osabi_t osabi, Half mach,
421     Conv_fmt_flags_t fmt_flags)
422 {
423     /*
424      * Maximum # of items that can be in the returned array. Size this
425      * by counting the maximum depth in the switch statement that fills
426      * retarr at the end of this function.
427      */
428     #define MAX_RET 12

430     /*
431      * Generic dynamic tags:
432      * - Note hole between DT_FLAGS and DT_PREINIT_ARRAY (tag 32).
433      *   We use a 0, which is the signal for "not defined".
434      * - This range has alternative names for dump, requiring an
435      *   additional array.
436      */
437     static const Msg     tags_null_cf[] = {
438         MSG_DT_NULL_CF,           MSG_DT_NEEDED_CF,
439         MSG_DT_PLTRELSZ_CF,      MSG_DT_PLTGOT_CF,
440         MSG_DT_HASH_CF,         MSG_DT_STRTAB_CF,
441         MSG_DT_SYMTAB_CF,       MSG_DT_RELA_CF,
442         MSG_DT_RELASZ_CF,       MSG_DT_RELAENT_CF,
443         MSG_DT_STRSZ_CF,        MSG_DT_SYMENT_CF,
444         MSG_DT_INIT_CF,         MSG_DT_FINI_CF,
445         MSG_DT_SONAME_CF,       MSG_DT_RPATH_CF,
446         MSG_DT_SYMBOLIC_CF,     MSG_DT_REL_CF,
447         MSG_DT_RELSZ_CF,        MSG_DT_RELENT_CF,
448         MSG_DT_PLTREL_CF,       MSG_DT_DEBUG_CF,
449         MSG_DT_TEXTREL_CF,      MSG_DT_JMPREL_CF,
450         MSG_DT_BIND_NOW_CF,     MSG_DT_INIT_ARRAY_CF,
451         MSG_DT_FINI_ARRAY_CF,   MSG_DT_INIT_ARRAYSZ_CF,
452         MSG_DT_FINI_ARRAYSZ_CF, MSG_DT_RUNPATH_CF,
453         MSG_DT_FLAGS_CF,        0,
454         MSG_DT_PREINIT_ARRAY_CF, MSG_DT_PREINIT_ARRAYSZ_CF
455     };
456     static const Msg     tags_null_cfnf[] = {
457         MSG_DT_NULL_CFNFP,      MSG_DT_NEEDED_CFNFP,
458         MSG_DT_PLTRELSZ_CFNFP,  MSG_DT_PLTGOT_CFNFP,
459         MSG_DT_HASH_CFNFP,      MSG_DT_STRTAB_CFNFP,
460         MSG_DT_SYMTAB_CFNFP,    MSG_DT_RELA_CFNFP,
461         MSG_DT_RELASZ_CFNFP,    MSG_DT_RELAENT_CFNFP,
462         MSG_DT_STRSZ_CFNFP,     MSG_DT_SYMENT_CFNFP,
463         MSG_DT_INIT_CFNFP,      MSG_DT_FINI_CFNFP,
464         MSG_DT_SONAME_CFNFP,    MSG_DT_RPATH_CFNFP,
465         MSG_DT_SYMBOLIC_CFNFP,  MSG_DT_REL_CFNFP,
466         MSG_DT_RELSZ_CFNFP,     MSG_DT_RELENT_CFNFP,
467         MSG_DT_PLTREL_CFNFP,    MSG_DT_DEBUG_CFNFP,
468         MSG_DT_TEXTREL_CFNFP,   MSG_DT_JMPREL_CFNFP,
469         MSG_DT_BIND_NOW_CFNFP,  MSG_DT_INIT_ARRAY_CFNFP,
470         MSG_DT_FINI_ARRAY_CFNFP, MSG_DT_INIT_ARRAYSZ_CFNFP,
471         MSG_DT_FINI_ARRAYSZ_CFNFP, MSG_DT_RUNPATH_CFNFP,
472         MSG_DT_FLAGS_CFNFP,     0,
473         MSG_DT_PREINIT_ARRAY_CFNFP, MSG_DT_PREINIT_ARRAYSZ_CFNFP
474     };
};

```

```

475     static const Msg     tags_null_nf[] = {
476         MSG_DT_NULL_NF,         MSG_DT_NEEDED_NF,
477         MSG_DT_PLTRELSZ_NF,    MSG_DT_PLTGOT_NF,
478         MSG_DT_HASH_NF,       MSG_DT_STRTAB_NF,
479         MSG_DT_SYMTAB_NF,     MSG_DT_RELA_NF,
480         MSG_DT_RELASZ_NF,     MSG_DT_RELAENT_NF,
481         MSG_DT_STRSZ_NF,      MSG_DT_SYMENT_NF,
482         MSG_DT_INIT_NF,       MSG_DT_FINI_NF,
483         MSG_DT_SONAME_NF,     MSG_DT_RPATH_NF,
484         MSG_DT_SYMBOLIC_NF,   MSG_DT_REL_NF,
485         MSG_DT_RELSZ_NF,      MSG_DT_RELENT_NF,
486         MSG_DT_PLTREL_NF,     MSG_DT_DEBUG_NF,
487         MSG_DT_TEXTREL_NF,    MSG_DT_JMPREL_NF,
488         MSG_DT_BIND_NOW_NF,   MSG_DT_INIT_ARRAY_NF,
489         MSG_DT_FINI_ARRAY_NF, MSG_DT_INIT_ARRAYSZ_NF,
490         MSG_DT_FINI_ARRAYSZ_NF, MSG_DT_RUNPATH_NF,
491         MSG_DT_FLAGS_NF,      0,
492         MSG_DT_PREINIT_ARRAY_NF, MSG_DT_PREINIT_ARRAYSZ_NF
493     };
494     static const Msg     tags_null_dmp[] = {
495         MSG_DT_NULL_CFNFP,      MSG_DT_NEEDED_CFNFP,
496         MSG_DT_PLTRELSZ_DMP,    MSG_DT_PLTGOT_CFNFP,
497         MSG_DT_HASH_CFNFP,     MSG_DT_STRTAB_CFNFP,
498         MSG_DT_SYMTAB_CFNFP,    MSG_DT_RELA_CFNFP,
499         MSG_DT_RELASZ_CFNFP,    MSG_DT_RELAENT_CFNFP,
500         MSG_DT_STRSZ_CFNFP,     MSG_DT_SYMENT_CFNFP,
501         MSG_DT_INIT_CFNFP,      MSG_DT_FINI_CFNFP,
502         MSG_DT_SONAME_CFNFP,    MSG_DT_RPATH_CFNFP,
503         MSG_DT_SYMBOLIC_DMP,    MSG_DT_REL_CFNFP,
504         MSG_DT_RELSZ_CFNFP,     MSG_DT_RELENT_CFNFP,
505         MSG_DT_PLTREL_CFNFP,    MSG_DT_DEBUG_CFNFP,
506         MSG_DT_TEXTREL_CFNFP,   MSG_DT_JMPREL_CFNFP,
507         MSG_DT_BIND_NOW_CFNFP,  MSG_DT_INIT_ARRAY_CFNFP,
508         MSG_DT_FINI_ARRAY_CFNFP, MSG_DT_INIT_ARRAYSZ_CFNFP,
509         MSG_DT_FINI_ARRAYSZ_CFNFP, MSG_DT_RUNPATH_CFNFP,
510         MSG_DT_FLAGS_CFNFP,     0,
511         MSG_DT_PREINIT_ARRAY_CFNFP, MSG_DT_PREINIT_ARRAYSZ_CFNFP
512     };
513     static const conv_ds_msg_t ds_null_cf = {
514         CONV_DS_MSG_INIT(DT_NULL, tags_null_cf) };
515     static const conv_ds_msg_t ds_null_cfnf = {
516         CONV_DS_MSG_INIT(DT_NULL, tags_null_cfnf) };
517     static const conv_ds_msg_t ds_null_nf = {
518         CONV_DS_MSG_INIT(DT_NULL, tags_null_nf) };
519     static const conv_ds_msg_t ds_null_dmp = {
520         CONV_DS_MSG_INIT(DT_NULL, tags_null_dmp) };

522     /*
523      * DT_SPARC_REGISTER was originally assigned 0x7000001. It is processor
524      * specific, and should have been in the range DT_LOPROC-DT_HIPROC
525      * instead of here. When the error was fixed,
526      * DT_DEPRECATED_SPARC_REGISTER was created to maintain backward
527      * compatibility.
528      */
529     static const Msg     tags_sdreg_cf[] = {
530         MSG_DT_DEP_SPARC_REG_CF };
531     static const Msg     tags_sdreg_cfnf[] = {
532         MSG_DT_DEP_SPARC_REG_CFNFP };
533     static const Msg     tags_sdreg_nf[] = {
534         MSG_DT_DEP_SPARC_REG_NF };

536     static const conv_ds_msg_t ds_sdreg_cf = {
537         CONV_DS_MSG_INIT(DT_DEPRECATED_SPARC_REGISTER, tags_sdreg_cf) };
538     static const conv_ds_msg_t ds_sdreg_cfnf = {
539         CONV_DS_MSG_INIT(DT_DEPRECATED_SPARC_REGISTER, tags_sdreg_cfnf) };
540     static const conv_ds_msg_t ds_sdreg_nf = {

```

```

541     CONV_DS_MSG_INIT(DT_DEPRECATED_SPARC_REGISTER, tags_sdreg_nf) };

544 /*
545  * SUNW: DT_LOOS -> DT_HIOS range. Note holes between DT_SUNW_TLSSORTSZ,
546  * DT_SUNW_STRPAD, and DT_SUNW_LDMACH. We handle the outliers
547  * separately below as single values.
548  */
549 static const Msg     tags_sunw_auxiliary_cf[] = {
550     MSG_DT_SUNW_AUXILIARY_CF,     MSG_DT_SUNW_RTLDINF_CF,
551     MSG_DT_SUNW_FILTER_CF,       MSG_DT_SUNW_CAP_CF,
552     MSG_DT_SUNW_SYMTAB_CF,       MSG_DT_SUNW_SYMSZ_CF,
553     MSG_DT_SUNW_SORTENT_CF,     MSG_DT_SUNW_SYMSORT_CF,
554     MSG_DT_SUNW_SYMSORTSZ_CF,   MSG_DT_SUNW_TLSSORT_CF,
555     MSG_DT_SUNW_TLSSORTSZ_CF,   MSG_DT_SUNW_CAPINFO_CF,
556     MSG_DT_SUNW_STRPAD_CF,     MSG_DT_SUNW_CAPCHAIN_CF,
557     MSG_DT_SUNW_LDMACH_CF,      0,
558     MSG_DT_SUNW_CAPCHAINENT_CF, 0,
559     MSG_DT_SUNW_CAPCHAINSZ_CF,  0,
560     0,                           0,
561     MSG_DT_SUNW_ASLR_CF
562     MSG_DT_SUNW_CAPCHAINSZ_CF
563 };
564 static const Msg     tags_sunw_auxiliary_cfnf[] = {
565     MSG_DT_SUNW_AUXILIARY_CFNFP,  MSG_DT_SUNW_RTLDINF_CFNFP,
566     MSG_DT_SUNW_FILTER_CFNFP,    MSG_DT_SUNW_CAP_CFNFP,
567     MSG_DT_SUNW_SYMTAB_CFNFP,    MSG_DT_SUNW_SYMSZ_CFNFP,
568     MSG_DT_SUNW_SORTENT_CFNFP,   MSG_DT_SUNW_SYMSORT_CFNFP,
569     MSG_DT_SUNW_TLSSORTSZ_CFNFP, MSG_DT_SUNW_TLSSORT_CFNFP,
570     MSG_DT_SUNW_STRPAD_CFNFP,    MSG_DT_SUNW_CAPINFO_CFNFP,
571     MSG_DT_SUNW_LDMACH_CFNFP,    0,
572     MSG_DT_SUNW_CAPCHAINENT_CFNFP, 0,
573     MSG_DT_SUNW_CAPCHAINSZ_CFNFP, 0,
574     0,                           0,
575     MSG_DT_SUNW_ASLR_CFNFP
576     MSG_DT_SUNW_CAPCHAINSZ_CFNFP
577 };
578 static const Msg     tags_sunw_auxiliary_nf[] = {
579     MSG_DT_SUNW_AUXILIARY_NF,     MSG_DT_SUNW_RTLDINF_NF,
580     MSG_DT_SUNW_FILTER_NF,       MSG_DT_SUNW_CAP_NF,
581     MSG_DT_SUNW_SYMTAB_NF,      MSG_DT_SUNW_SYMSZ_NF,
582     MSG_DT_SUNW_SORTENT_NF,     MSG_DT_SUNW_SYMSORT_NF,
583     MSG_DT_SUNW_TLSSORTSZ_NF,   MSG_DT_SUNW_TLSSORT_NF,
584     MSG_DT_SUNW_STRPAD_NF,     MSG_DT_SUNW_CAPINFO_NF,
585     MSG_DT_SUNW_LDMACH_NF,      0,
586     MSG_DT_SUNW_CAPCHAINENT_NF, 0,
587     MSG_DT_SUNW_CAPCHAINSZ_NF,  0,
588     0,                           0,
589     MSG_DT_SUNW_ASLR_NF
590     MSG_DT_SUNW_CAPCHAINSZ_NF
591 };
592 static const conv_ds_msg_t ds_sunw_auxiliary_cf = {
593     CONV_DS_MSG_INIT(DT_SUNW_AUXILIARY, tags_sunw_auxiliary_cf) };
594 static const conv_ds_msg_t ds_sunw_auxiliary_cfnf = {
595     CONV_DS_MSG_INIT(DT_SUNW_AUXILIARY, tags_sunw_auxiliary_cfnf) };
596 static const conv_ds_msg_t ds_sunw_auxiliary_nf = {
597     CONV_DS_MSG_INIT(DT_SUNW_AUXILIARY, tags_sunw_auxiliary_nf) };

598 /*
599  * GNU: (In DT_VALRNGLO section) DT_GNU_PRELINKED - DT_GNU_LIBLISTSZ
600  */
601 static const Msg     tags_gnu_prelinked_cf[] = {
602     MSG_DT_GNU_PRELINKED_CF,     MSG_DT_GNU_CONFLICTSZ_CF,
603     MSG_DT_GNU_LIBLISTSZ_CF

```

```

604     };
605 static const Msg     tags_gnu_prelinked_cfnf[] = {
606     MSG_DT_GNU_PRELINKED_CFNFP,  MSG_DT_GNU_CONFLICTSZ_CFNFP,
607     MSG_DT_GNU_LIBLISTSZ_CFNFP
608     };
609 static const Msg     tags_gnu_prelinked_nf[] = {
610     MSG_DT_GNU_PRELINKED_NF,     MSG_DT_GNU_CONFLICTSZ_NF,
611     MSG_DT_GNU_LIBLISTSZ_NF
612     };
613 static const conv_ds_msg_t ds_gnu_prelinked_cf = {
614     CONV_DS_MSG_INIT(DT_GNU_PRELINKED, tags_gnu_prelinked_cf) };
615 static const conv_ds_msg_t ds_gnu_prelinked_cfnf = {
616     CONV_DS_MSG_INIT(DT_GNU_PRELINKED, tags_gnu_prelinked_cfnf) };
617 static const conv_ds_msg_t ds_gnu_prelinked_nf = {
618     CONV_DS_MSG_INIT(DT_GNU_PRELINKED, tags_gnu_prelinked_nf) };

620 /*
621  * SUNW: DT_VALRNGLO - DT_VALRNGHI range.
622  */
623 static const Msg     tags_checksum_cf[] = {
624     MSG_DT_CHECKSUM_CF,         MSG_DT_PLTPADSZ_CF,
625     MSG_DT_MOVEENT_CF,        MSG_DT_MOVESZ_CF,
626     MSG_DT_FEATURE_1_CF,      MSG_DT_POSFLAG_1_CF,
627     MSG_DT_SYMINSZ_CF,       MSG_DT_SYMMENT_CF
628     };
629 static const Msg     tags_checksum_cfnf[] = {
630     MSG_DT_CHECKSUM_CFNFP,     MSG_DT_PLTPADSZ_CFNFP,
631     MSG_DT_MOVEENT_CFNFP,     MSG_DT_MOVESZ_CFNFP,
632     MSG_DT_FEATURE_1_CFNFP,   MSG_DT_POSFLAG_1_CFNFP,
633     MSG_DT_SYMINSZ_CFNFP,     MSG_DT_SYMMENT_CFNFP
634     };
635 static const Msg     tags_checksum_nf[] = {
636     MSG_DT_CHECKSUM_NF,        MSG_DT_PLTPADSZ_NF,
637     MSG_DT_MOVEENT_NF,        MSG_DT_MOVESZ_NF,
638     MSG_DT_FEATURE_1_NF,     MSG_DT_POSFLAG_1_NF,
639     MSG_DT_SYMINSZ_NF,       MSG_DT_SYMMENT_NF
640     };
641 static const conv_ds_msg_t ds_checksum_cf = {
642     CONV_DS_MSG_INIT(DT_CHECKSUM, tags_checksum_cf) };
643 static const conv_ds_msg_t ds_checksum_cfnf = {
644     CONV_DS_MSG_INIT(DT_CHECKSUM, tags_checksum_cfnf) };
645 static const conv_ds_msg_t ds_checksum_nf = {
646     CONV_DS_MSG_INIT(DT_CHECKSUM, tags_checksum_nf) };

648 /*
649  * GNU: (In DT_ADDRNGLO section) DT_GNU_HASH - DT_GNU_LIBLIST
650  */
651 static const Msg     tags_gnu_hash_cf[] = {
652     MSG_DT_GNU_HASH_CF,        MSG_DT_TLSDESC_PLT_CF,
653     MSG_DT_TLSDESC_GOT_CF,    MSG_DT_GNU_CONFLICT_CF,
654     MSG_DT_GNU_LIBLIST_CF
655     };
656 static const Msg     tags_gnu_hash_cfnf[] = {
657     MSG_DT_GNU_HASH_CFNFP,     MSG_DT_TLSDESC_PLT_CFNFP,
658     MSG_DT_TLSDESC_GOT_CFNFP,  MSG_DT_GNU_CONFLICT_CFNFP,
659     MSG_DT_GNU_LIBLIST_CFNFP
660     };
661 static const Msg     tags_gnu_hash_nf[] = {
662     MSG_DT_GNU_HASH_NF,        MSG_DT_TLSDESC_PLT_NF,
663     MSG_DT_TLSDESC_GOT_NF,    MSG_DT_GNU_CONFLICT_NF,
664     MSG_DT_GNU_LIBLIST_NF
665     };
666 static const conv_ds_msg_t ds_gnu_hash_cf = {
667     CONV_DS_MSG_INIT(DT_GNU_HASH, tags_gnu_hash_cf) };
668 static const conv_ds_msg_t ds_gnu_hash_cfnf = {
669     CONV_DS_MSG_INIT(DT_GNU_HASH, tags_gnu_hash_cfnf) };

```

```

670 static const conv_ds_msg_t ds_gnu_hash_nf = {
671     CONV_DS_MSG_INIT(DT_GNU_HASH, tags_gnu_hash_nf) };

673 /*
674  * SUNW: DT_ADDRNGLO - DT_ADDRNGHI range.
675  */
676 static const Msg     tags_config_cf[] = {
677     MSG_DT_CONFIG_CF,           MSG_DT_DEPAUDIT_CF,
678     MSG_DT_AUDIT_CF,           MSG_DT_PLTPAD_CF,
679     MSG_DT_MOVETAB_CF,         MSG_DT_SYMINFO_CF
680 };
681 static const Msg     tags_config_cfnf[] = {
682     MSG_DT_CONFIG_CFNFP,       MSG_DT_DEPAUDIT_CFNFP,
683     MSG_DT_AUDIT_CFNFP,        MSG_DT_PLTPAD_CFNFP,
684     MSG_DT_MOVETAB_CFNFP,      MSG_DT_SYMINFO_CFNFP
685 };
686 static const Msg     tags_config_nf[] = {
687     MSG_DT_CONFIG_NF,          MSG_DT_DEPAUDIT_NF,
688     MSG_DT_AUDIT_NF,           MSG_DT_PLTPAD_NF,
689     MSG_DT_MOVETAB_NF,        MSG_DT_SYMINFO_NF
690 };
691 static const conv_ds_msg_t ds_config_cf = {
692     CONV_DS_MSG_INIT(DT_CONFIG, tags_config_cf) };
693 static const conv_ds_msg_t ds_config_cfnf = {
694     CONV_DS_MSG_INIT(DT_CONFIG, tags_config_cfnf) };
695 static const conv_ds_msg_t ds_config_nf = {
696     CONV_DS_MSG_INIT(DT_CONFIG, tags_config_nf) };

698 /*
699  * SUNW: generic range. Note hole between DT_VERSYM and DT_RELACOUNT.
700  */
701 static const Msg     tags_versym_cf[] = { MSG_DT_VERSYM_CF };
702 static const Msg     tags_versym_cfnf[] = { MSG_DT_VERSYM_CFNFP };
703 static const Msg     tags_versym_nf[] = { MSG_DT_VERSYM_NF };
704 static const conv_ds_msg_t ds_versym_cf = {
705     CONV_DS_MSG_INIT(DT_VERSYM, tags_versym_cf) };
706 static const conv_ds_msg_t ds_versym_cfnf = {
707     CONV_DS_MSG_INIT(DT_VERSYM, tags_versym_cfnf) };
708 static const conv_ds_msg_t ds_versym_nf = {
709     CONV_DS_MSG_INIT(DT_VERSYM, tags_versym_nf) };

711 static const Msg     tags_relaount_cf[] = {
712     MSG_DT_RELACOUNT_CF,       MSG_DT_RELACOUNT_CF,
713     MSG_DT_FLAGS_1_CF,         MSG_DT_VERDEF_CF,
714     MSG_DT_VERDEFNUM_CF,      MSG_DT_VERNEED_CF,
715     MSG_DT_VERNEEDNUM_CF
716 };
717 static const Msg     tags_relaount_cfnf[] = {
718     MSG_DT_RELACOUNT_CFNFP,    MSG_DT_RELACOUNT_CFNFP,
719     MSG_DT_FLAGS_1_CFNFP,      MSG_DT_VERDEF_CFNFP,
720     MSG_DT_VERDEFNUM_CFNFP,    MSG_DT_VERNEED_CFNFP,
721     MSG_DT_VERNEEDNUM_CFNFP
722 };
723 static const Msg     tags_relaount_nf[] = {
724     MSG_DT_RELACOUNT_NF,       MSG_DT_RELACOUNT_NF,
725     MSG_DT_FLAGS_1_NF,         MSG_DT_VERDEF_NF,
726     MSG_DT_VERDEFNUM_NF,      MSG_DT_VERNEED_NF,
727     MSG_DT_VERNEEDNUM_NF
728 };
729 static const conv_ds_msg_t ds_relaount_cf = {
730     CONV_DS_MSG_INIT(DT_RELACOUNT, tags_relaount_cf) };
731 static const conv_ds_msg_t ds_relaount_cfnf = {
732     CONV_DS_MSG_INIT(DT_RELACOUNT, tags_relaount_cfnf) };
733 static const conv_ds_msg_t ds_relaount_nf = {
734     CONV_DS_MSG_INIT(DT_RELACOUNT, tags_relaount_nf) };

```

```

736 /*
737  * DT_LOPROC - DT_HIPROC range: solaris/sparc-only
738  */
739 static const Msg     tags_sparc_reg_cf[] = { MSG_DT_SPARC_REGISTER_CF };
740 static const Msg     tags_sparc_reg_cfnf[] = { MSG_DT_SPARC_REGISTER_CFNFP };
741 static const Msg     tags_sparc_reg_nf[] = { MSG_DT_SPARC_REGISTER_NF };
742 static const Msg     tags_sparc_reg_dmp[] = { MSG_DT_SPARC_REGISTER_DMP };
743 static const conv_ds_msg_t ds_sparc_reg_cf = {
744     CONV_DS_MSG_INIT(DT_SPARC_REGISTER, tags_sparc_reg_cf) };
745 static const conv_ds_msg_t ds_sparc_reg_cfnf = {
746     CONV_DS_MSG_INIT(DT_SPARC_REGISTER, tags_sparc_reg_cfnf) };
747 static const conv_ds_msg_t ds_sparc_reg_nf = {
748     CONV_DS_MSG_INIT(DT_SPARC_REGISTER, tags_sparc_reg_nf) };
749 static const conv_ds_msg_t ds_sparc_reg_dmp = {
750     CONV_DS_MSG_INIT(DT_SPARC_REGISTER, tags_sparc_reg_dmp) };

752 /*
753  * DT_LOPROC - DT_HIPROC range: Solaris osabi, all hardware
754  */
755 static const Msg     tags_auxiliary_cf[] = {
756     MSG_DT_AUXILIARY_CF,       MSG_DT_USED_CF,
757     MSG_DT_FILTER_CF
758 };
759 static const Msg     tags_auxiliary_cfnf[] = {
760     MSG_DT_AUXILIARY_CFNFP,    MSG_DT_USED_CFNFP,
761     MSG_DT_FILTER_CFNFP
762 };
763 static const Msg     tags_auxiliary_nf[] = {
764     MSG_DT_AUXILIARY_NF,       MSG_DT_USED_NF,
765     MSG_DT_FILTER_NF
766 };
767 static const conv_ds_msg_t ds_auxiliary_cf = {
768     CONV_DS_MSG_INIT(DT_AUXILIARY, tags_auxiliary_cf) };
769 static const conv_ds_msg_t ds_auxiliary_cfnf = {
770     CONV_DS_MSG_INIT(DT_AUXILIARY, tags_auxiliary_cfnf) };
771 static const conv_ds_msg_t ds_auxiliary_nf = {
772     CONV_DS_MSG_INIT(DT_AUXILIARY, tags_auxiliary_nf) };

775 static const conv_ds_t *retarr[MAX_RET];

777 int     ndx = 0;
778 int     fmt_osabi = CONV_TYPE_FMT_ALT(fmt_flags);
779 int     mach_sparc, osabi_solaris, osabi_linux;

783 osabi_solaris = (osabi == ELFOSABI_NONE) ||
784     (osabi == ELFOSABI_SOLARIS) || (osabi == CONV_OSABI_ALL);
785 osabi_linux = (osabi == ELFOSABI_LINUX) || (osabi == CONV_OSABI_ALL);
786 mach_sparc = (mach == EM_SPARC) || (mach == EM_SPARCV9) ||
787     (mach == EM_SPARC32PLUS) || (mach == CONV_MACH_ALL);

789 /*
790  * Fill in retarr with the descriptors for the messages that
791  * apply to the current osabi. Note that we order these items such
792  * that the more common are placed at the beginning, and the less
793  * likely at the end. This should speed the common case.
794  *
795  * Note that the CFNP and DMP styles are very similar, so they
796  * are combined in 'default', and fmt_osabi is consulted when there
797  * are differences.
798  */
799 switch (fmt_osabi) {
800 case CONV_FMT_ALT_CF:
801     retarr[ndx++] = CONV_DS_ADDR(ds_null_cf);

```

```

802     if (osabi_solaris)
803         retarr[ndx++] = CONV_DS_ADDR(ds_sunw_auxiliary_cf);
804     retarr[ndx++] = CONV_DS_ADDR(ds_checksum_cf);
805     retarr[ndx++] = CONV_DS_ADDR(ds_config_cf);
806     retarr[ndx++] = CONV_DS_ADDR(ds_versym_cf);
807     retarr[ndx++] = CONV_DS_ADDR(ds_relaount_cf);
808     if (osabi_solaris) {
809         retarr[ndx++] = CONV_DS_ADDR(ds_auxiliary_cf);
810         if (mach_sparc) {
811             retarr[ndx++] = CONV_DS_ADDR(ds_sparc_reg_cf);
812             retarr[ndx++] = CONV_DS_ADDR(ds_sdreg_cf);
813         }
814     }
815     if (osabi_linux) {
816         retarr[ndx++] = CONV_DS_ADDR(ds_gnu_prelinked_cf);
817         retarr[ndx++] = CONV_DS_ADDR(ds_gnu_hash_cf);
818     }
819     break;

821 case CONV_FMT_ALT_NF:
822     retarr[ndx++] = CONV_DS_ADDR(ds_null_nf);
823     if (osabi_solaris)
824         retarr[ndx++] = CONV_DS_ADDR(ds_sunw_auxiliary_nf);
825     retarr[ndx++] = CONV_DS_ADDR(ds_checksum_nf);
826     retarr[ndx++] = CONV_DS_ADDR(ds_config_nf);
827     retarr[ndx++] = CONV_DS_ADDR(ds_versym_nf);
828     retarr[ndx++] = CONV_DS_ADDR(ds_relaount_nf);
829     if (osabi_solaris) {
830         retarr[ndx++] = CONV_DS_ADDR(ds_auxiliary_nf);
831         if (mach_sparc) {
832             retarr[ndx++] = CONV_DS_ADDR(ds_sparc_reg_nf);
833             retarr[ndx++] = CONV_DS_ADDR(ds_sdreg_nf);
834         }
835     }
836     if (osabi_linux) {
837         retarr[ndx++] = CONV_DS_ADDR(ds_gnu_prelinked_nf);
838         retarr[ndx++] = CONV_DS_ADDR(ds_gnu_hash_nf);
839     }
840     break;
841 default:
842     /*
843     * The default style for the generic range is CFNP,
844     * while dump has a couple of different strings.
845     */

847     retarr[ndx++] = (fmt_osabi == CONV_FMT_ALT_DUMP) ?
848         CONV_DS_ADDR(ds_null_dmp) : CONV_DS_ADDR(ds_null_cfnf);
849     if (osabi_solaris)
850         retarr[ndx++] = CONV_DS_ADDR(ds_sunw_auxiliary_cfnf);
851     retarr[ndx++] = CONV_DS_ADDR(ds_checksum_cfnf);
852     retarr[ndx++] = CONV_DS_ADDR(ds_config_cfnf);
853     retarr[ndx++] = CONV_DS_ADDR(ds_versym_cfnf);
854     retarr[ndx++] = CONV_DS_ADDR(ds_relaount_cfnf);
855     if (osabi_solaris) {
856         retarr[ndx++] = CONV_DS_ADDR(ds_auxiliary_cfnf);
857         if (mach_sparc) {
858             /*
859             * The default style for DT_SPARC_REGISTER
860             * is the dump style, which omits the 'SPARC_'.
861             * CFNP keeps the prefix.
862             */
863             retarr[ndx++] =
864                 (fmt_osabi == CONV_FMT_ALT_CFNP) ?
865                 CONV_DS_ADDR(ds_sparc_reg_cfnf) :
866                 CONV_DS_ADDR(ds_sparc_reg_dmp);
867             retarr[ndx++] = CONV_DS_ADDR(ds_sdreg_cfnf);

```

```

868         }
869     }
870     if (osabi_linux) {
871         retarr[ndx++] = CONV_DS_ADDR(ds_gnu_prelinked_cfnf);
872         retarr[ndx++] = CONV_DS_ADDR(ds_gnu_hash_cfnf);
873     }
874     break;
875 }

877     retarr[ndx++] = NULL;
878     assert(ndx <= MAX_RET);
879     return (retarr);
880 }

```

unchanged_portion_omitted

```

*****
16568 Thu Jun 30 22:01:30 2016
new/usr/src/cmd/sgs/libconv/common/dynamic.msg
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
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 #
23 # Copyright (c) 1995, 2010, Oracle and/or its affiliates. All rights reserved.
24 #
25 #
26 @ MSG_DT_NULL_CF          "DT_NULL"          # 0
27 @ MSG_DT_NULL_CFNFP      "NULL"
28 @ MSG_DT_NULL_NF         "null"
29 @ MSG_DT_NEEDED_CF       "DT_NEEDED"        # 1
30 @ MSG_DT_NEEDED_CFNFP    "NEEDED"
31 @ MSG_DT_NEEDED_NF       "needed"
32 @ MSG_DT_PLTRELSZ_CF     "DT_PLTRELSZ"     # 2
33 @ MSG_DT_PLTRELSZ_CFNFP  "PLTRELSZ"
34 @ MSG_DT_PLTRELSZ_NF     "ptrelsz"
35 @ MSG_DT_PLTRELSZ_DMP    "PLTSZ"
36 @ MSG_DT_PLTGOT_CF       "DT_PLTGOT"        # 3
37 @ MSG_DT_PLTGOT_CFNFP    "PLTGOT"
38 @ MSG_DT_PLTGOT_NF       "pltgot"
39 @ MSG_DT_HASH_CF         "DT_HASH"          # 4
40 @ MSG_DT_HASH_CFNFP      "HASH"
41 @ MSG_DT_HASH_NF         "hash"
42 @ MSG_DT_STRTAB_CF       "DT_STRTAB"        # 5
43 @ MSG_DT_STRTAB_CFNFP    "STRTAB"
44 @ MSG_DT_STRTAB_NF       "strtab"
45 @ MSG_DT_SYMTAB_CF       "DT_SYMTAB"        # 6
46 @ MSG_DT_SYMTAB_CFNFP    "SYMTAB"
47 @ MSG_DT_SYMTAB_NF       "symtab"
48 @ MSG_DT_RELA_CF         "DT_RELA"          # 7
49 @ MSG_DT_RELA_CFNFP      "RELA"
50 @ MSG_DT_RELA_NF         "rela"
51 @ MSG_DT_RELASZ_CF       "DT_RELASZ"       # 8
52 @ MSG_DT_RELASZ_CFNFP    "RELASZ"
53 @ MSG_DT_RELASZ_NF       "relasz"
54 @ MSG_DT_RELAENT_CF     "DT_RELAENT"      # 9
55 @ MSG_DT_RELAENT_CFNFP   "RELAENT"
56 @ MSG_DT_RELAENT_NF      "relaent"
57 @ MSG_DT_STRSZ_CF        "DT_STRSZ"        # 10
58 @ MSG_DT_STRSZ_CFNFP     "STRSZ"

```

```

59 @ MSG_DT_STRSZ_NF       "strsz"
60 @ MSG_DT_SYMENT_CF      "DT_SYMENT"        # 11
61 @ MSG_DT_SYMENT_CFNFP   "SYMENT"
62 @ MSG_DT_SYMENT_NF      "syment"
63 @ MSG_DT_INIT_CF        "DT_INIT"          # 12
64 @ MSG_DT_INIT_CFNFP     "INIT"
65 @ MSG_DT_INIT_NF        "init"
66 @ MSG_DT_FINI_CF        "DT_FINI"          # 13
67 @ MSG_DT_FINI_CFNFP     "FINI"
68 @ MSG_DT_FINI_NF        "fini"
69 @ MSG_DT_SONAME_CF      "DT_SONAME"        # 14
70 @ MSG_DT_SONAME_CFNFP   "SONAME"
71 @ MSG_DT_SONAME_NF      "soname"
72 @ MSG_DT_RPATH_CF       "DT_RPATH"        # 15
73 @ MSG_DT_RPATH_CFNFP    "RPATH"
74 @ MSG_DT_RPATH_NF       "rpath"
75 @ MSG_DT_SYMBOLIC_CF    "DT_SYMBOLIC"      # 16
76 @ MSG_DT_SYMBOLIC_CFNFP "SYMBOLIC"
77 @ MSG_DT_SYMBOLIC_NF    "symbolic"
78 @ MSG_DT_SYMBOLIC_DMP   "SYMB"
79 @ MSG_DT_REL_CF         "DT_REL"          # 17
80 @ MSG_DT_REL_CFNFP      "REL"
81 @ MSG_DT_REL_NF         "rel"
82 @ MSG_DT_RELSZ_CF       "DT_RELSZ"        # 18
83 @ MSG_DT_RELSZ_CFNFP    "RELSZ"
84 @ MSG_DT_RELSZ_NF       "relsz"
85 @ MSG_DT_RELENT_CF      "DT_RELENT"       # 19
86 @ MSG_DT_RELENT_CFNFP   "RELENT"
87 @ MSG_DT_RELENT_NF      "relent"
88 @ MSG_DT_PLTREL_CF      "DT_PLTREL"       # 20
89 @ MSG_DT_PLTREL_CFNFP   "PLTREL"
90 @ MSG_DT_PLTREL_NF      "pltrel"
91 @ MSG_DT_DEBUG_CF       "DT_DEBUG"        # 21
92 @ MSG_DT_DEBUG_CFNFP    "DEBUG"
93 @ MSG_DT_DEBUG_NF       "debug"
94 @ MSG_DT_TEXTREL_CF     "DT_TEXTREL"     # 22
95 @ MSG_DT_TEXTREL_CFNFP  "TEXTREL"
96 @ MSG_DT_TEXTREL_NF     "textrel"
97 @ MSG_DT_JMPREL_CF      "DT_JMPREL"       # 23
98 @ MSG_DT_JMPREL_CFNFP   "JMPREL"
99 @ MSG_DT_JMPREL_NF      "jmplrel"
100 @ MSG_DT_BIND_NOW_CF    "DT_BIND_NOW"    # 24
101 @ MSG_DT_BIND_NOW_CFNFP "BIND_NOW"
102 @ MSG_DT_BIND_NOW_NF    "bind_now"
103 @ MSG_DT_INIT_ARRAY_CF  "DT_INIT_ARRAY"  # 25
104 @ MSG_DT_INIT_ARRAY_CFNFP "INIT_ARRAY"
105 @ MSG_DT_INIT_ARRAY_NF  "init_array"
106 @ MSG_DT_FINI_ARRAY_CF  "DT_FINI_ARRAY"  # 26
107 @ MSG_DT_FINI_ARRAY_CFNFP "FINI_ARRAY"
108 @ MSG_DT_FINI_ARRAY_NF  "fini_array"
109 @ MSG_DT_INIT_ARRAYSZ_CF "DT_INIT_ARRAYSZ" # 27
110 @ MSG_DT_INIT_ARRAYSZ_CFNFP "INIT_ARRAYSZ"
111 @ MSG_DT_INIT_ARRAYSZ_NF "init_arraysz"
112 @ MSG_DT_FINI_ARRAYSZ_CF "DT_FINI_ARRAYSZ" # 28
113 @ MSG_DT_FINI_ARRAYSZ_CFNFP "FINI_ARRAYSZ"
114 @ MSG_DT_FINI_ARRAYSZ_NF "fini_arraysz"
115 @ MSG_DT_RUNPATH_CF     "DT_RUNPATH"     # 29
116 @ MSG_DT_RUNPATH_CFNFP  "RUNPATH"
117 @ MSG_DT_RUNPATH_NF     "runpath"
118 @ MSG_DT_FLAGS_CF       "DT_FLAGS"        # 30
119 @ MSG_DT_FLAGS_CFNFP    "FLAGS"
120 @ MSG_DT_FLAGS_NF       "flags"
121 @ MSG_DT_PREINIT_ARRAY_CF "DT_PREINIT_ARRAY" # 32
122 @ MSG_DT_PREINIT_ARRAY_CFNFP "PREINIT_ARRAY"
123 @ MSG_DT_PREINIT_ARRAY_NF "preinit_array"
124 @ MSG_DT_PREINIT_ARRAYSZ_CF "DT_PREINIT_ARRAYSZ" # 33

```

```

125 @ MSG_DT_PREINIT_ARRAYSZ_CFNP      "PREINIT_ARRAYSZ"
126 @ MSG_DT_PREINIT_ARRAYSZ_NF        "preinit_arraysz"
127 @ MSG_DT_DEP_SPARC_REG_CF          "DT_DEPRECATED_SPARC_REGISTER" # 0x07000001
128 @ MSG_DT_DEP_SPARC_REG_CFNP       "DEPRECATED_SPARC_REGISTER"
129 @ MSG_DT_DEP_SPARC_REG_NF          "deprecated_sparc_register"
130 @ MSG_DT_SUNW_AUXILIARY_CF         "DT_SUNW_AUXILIARY" # 0x6000000d
131 @ MSG_DT_SUNW_AUXILIARY_CFNP       "SUNW_AUXILIARY"
132 @ MSG_DT_SUNW_AUXILIARY_NF         "sunw_auxiliary"
133 @ MSG_DT_SUNW_RTLDINF_CF           "DT_SUNW_RTLDINF" # 0x6000000e
134 @ MSG_DT_SUNW_RTLDINF_CFNP         "SUNW_RTLDINF"
135 @ MSG_DT_SUNW_RTLDINF_NF           "sunw_rtldinf"
136 @ MSG_DT_SUNW_FILTER_CF            "DT_SUNW_FILTER" # 0x6000000f
137 @ MSG_DT_SUNW_FILTER_CFNP          "SUNW_FILTER"
138 @ MSG_DT_SUNW_FILTER_NF            "sunw_filter"
139 @ MSG_DT_SUNW_CAP_CF               "DT_SUNW_CAP" # 0x60000010
140 @ MSG_DT_SUNW_CAP_CFNP             "SUNW_CAP"
141 @ MSG_DT_SUNW_CAP_NF               "sunw_cap"
142 @ MSG_DT_SUNW_SYMTAB_CF            "DT_SUNW_SYMTAB" # 0x60000011
143 @ MSG_DT_SUNW_SYMTAB_CFNP          "SUNW_SYMTAB"
144 @ MSG_DT_SUNW_SYMTAB_NF            "sunw_symtab"
145 @ MSG_DT_SUNW_SYMSZ_CF             "DT_SUNW_SYMSZ" # 0x60000012
146 @ MSG_DT_SUNW_SYMSZ_CFNP          "SUNW_SYMSZ"
147 @ MSG_DT_SUNW_SYMSZ_NF            "sunw_symsz"
148 @ MSG_DT_SUNW_SORTENT_CF           "DT_SUNW_SORTENT" # 0x60000013
149 @ MSG_DT_SUNW_SORTENT_CFNP         "SUNW_SORTENT"
150 @ MSG_DT_SUNW_SORTENT_NF           "sunw_sortent"
151 @ MSG_DT_SUNW_SYMSORT_CF           "DT_SUNW_SYMSORT" # 0x60000014
152 @ MSG_DT_SUNW_SYMSORT_CFNP        "SUNW_SYMSORT"
153 @ MSG_DT_SUNW_SYMSORT_NF           "sunw_symsort"
154 @ MSG_DT_SUNW_SYMSORTSZ_CF         "DT_SUNW_SYMSORTSZ" # 0x60000015
155 @ MSG_DT_SUNW_SYMSORTSZ_CFNP       "SUNW_SYMSORTSZ"
156 @ MSG_DT_SUNW_SYMSORTSZ_NF         "sunw_symsortsz"
157 @ MSG_DT_SUNW_TLSSORT_CF           "DT_SUNW_TLSSORT" # 0x60000016
158 @ MSG_DT_SUNW_TLSSORT_CFNP         "SUNW_TLSSORT"
159 @ MSG_DT_SUNW_TLSSORT_NF           "sunw_tlssort"
160 @ MSG_DT_SUNW_TLSSORTSZ_CF         "DT_SUNW_TLSSORTSZ" # 0x60000017
161 @ MSG_DT_SUNW_TLSSORTSZ_CFNP       "SUNW_TLSSORTSZ"
162 @ MSG_DT_SUNW_TLSSORTSZ_NF         "sunw_tlssortsz"
163 @ MSG_DT_SUNW_CAPINFO_CF           "DT_SUNW_CAPINFO" # 0x60000018
164 @ MSG_DT_SUNW_CAPINFO_CFNP        "SUNW_CAPINFO"
165 @ MSG_DT_SUNW_CAPINFO_NF           "sunw_capinfo"
166 @ MSG_DT_SUNW_STRPAD_CF            "DT_SUNW_STRPAD" # 0x60000019
167 @ MSG_DT_SUNW_STRPAD_CFNP          "SUNW_STRPAD"
168 @ MSG_DT_SUNW_STRPAD_NF            "sunw_strpad"
169 @ MSG_DT_SUNW_CAPCHAIN_CF          "DT_SUNW_CAPCHAIN" # 0x6000001a
170 @ MSG_DT_SUNW_CAPCHAIN_CFNP        "SUNW_CAPCHAIN"
171 @ MSG_DT_SUNW_CAPCHAIN_NF          "sunw_capchain"
172 @ MSG_DT_SUNW_LDMACH_CF            "DT_SUNW_LDMACH" # 0x6000001b
173 @ MSG_DT_SUNW_LDMACH_CFNP          "SUNW_LDMACH"
174 @ MSG_DT_SUNW_LDMACH_NF            "sunw_ldmach"
175 @ MSG_DT_SUNW_CAPCHAINENT_CF       "DT_SUNW_CAPCHAINENT" # 0x6000001d
176 @ MSG_DT_SUNW_CAPCHAINENT_CFNP     "SUNW_CAPCHAINENT"
177 @ MSG_DT_SUNW_CAPCHAINENT_NF       "sunw_capchainent"
178 @ MSG_DT_SUNW_CAPCHAINSZ_CF        "DT_SUNW_CAPCHAINSZ" # 0x6000001d
179 @ MSG_DT_SUNW_CAPCHAINSZ_CFNP      "SUNW_CAPCHAINSZ"
180 @ MSG_DT_SUNW_CAPCHAINSZ_NF        "sunw_capchainsz"
181 @ MSG_DT_SUNW_ASLR_CF              "DT_SUNW_ASLR" # 0x60000023
182 @ MSG_DT_SUNW_ASLR_CFNP            "SUNW_ASLR"
183 @ MSG_DT_SUNW_ASLR_NF              "sunw_aslr"
184 #endif /* ! codereview */

186 @ MSG_DT_GNU_PRELINKED_CF          "DT_GNU_PRELINKED" # 0x6ffffdf5
187 @ MSG_DT_GNU_PRELINKED_CFNP        "GNU_PRELINKED"
188 @ MSG_DT_GNU_PRELINKED_NF          "gnu_prelinked"
189 @ MSG_DT_GNU_CONFLICTSZ_CF         "DT_GNU_CONFLICTSZ" # 0x6ffffdf6
190 @ MSG_DT_GNU_CONFLICTSZ_CFNP       "GNU_CONFLICTSZ"

```

```

191 @ MSG_DT_GNU_CONFLICTSZ_NF         "gnu_conflictsz"
192 @ MSG_DT_GNU_LIBLISTSZ_CF          "DT_GNU_LIBLISTSZ" # 0x6ffffdf7
193 @ MSG_DT_GNU_LIBLISTSZ_CFNP        "GNU_LIBLISTSZ"
194 @ MSG_DT_GNU_LIBLISTSZ_NF          "gnu_liblistsz"
195 @ MSG_DT_CHECKSUM_CF               "DT_CHECKSUM" # 0x6ffffdf8
196 @ MSG_DT_CHECKSUM_CFNP             "CHECKSUM"
197 @ MSG_DT_CHECKSUM_NF                "checksum"
198 @ MSG_DT_PLTPADSZ_CF               "DT_PLTPADSZ" # 0x6ffffdf9
199 @ MSG_DT_PLTPADSZ_CFNP             "PLTPADSZ"
200 @ MSG_DT_PLTPADSZ_NF               "pltpadsz"
201 @ MSG_DT_MOVEENT_CF                "DT_MOVEENT" # 0x6ffffdfa
202 @ MSG_DT_MOVEENT_CFNP              "MOVEENT"
203 @ MSG_DT_MOVEENT_NF                "moveent"
204 @ MSG_DT_MOVESZ_CF                 "DT_MOVESZ" # 0x6ffffdfb
205 @ MSG_DT_MOVESZ_CFNP               "MOVESZ"
206 @ MSG_DT_MOVESZ_NF                 "movesz"
207 @ MSG_DT_FEATURE_1_CF              "DT_FEATURE_1" # 0x6ffffdfc
208 @ MSG_DT_FEATURE_1_CFNP            "FEATURE_1"
209 @ MSG_DT_FEATURE_1_NF              "feature_1"
210 @ MSG_DT_POSFLAG_1_CF              "DT_POSFLAG_1" # 0x6ffffdfd
211 @ MSG_DT_POSFLAG_1_CFNP            "POSFLAG_1"
212 @ MSG_DT_POSFLAG_1_NF              "posflag_1"
213 @ MSG_DT_SYMINSZ_CF                "DT_SYMINSZ" # 0x6ffffdfe
214 @ MSG_DT_SYMINSZ_CFNP              "SYMINSZ"
215 @ MSG_DT_SYMINSZ_NF                "syminsz"
216 @ MSG_DT_SYMINENT_CF               "DT_SYMINENT" # 0x6ffffdff
217 @ MSG_DT_SYMINENT_CFNP             "SYMINENT"
218 @ MSG_DT_SYMINENT_NF               "syminent"
219 @ MSG_DT_GNU_HASH_CF               "DT_GNU_HASH" # 0x6fffffef5
220 @ MSG_DT_GNU_HASH_CFNP             "GNU_HASH"
221 @ MSG_DT_GNU_HASH_NF               "gnu_hash"
222 @ MSG_DT_TLSDESC_PLT_CF            "DT_TLSDESC_PLT" # 0x6fffffef6
223 @ MSG_DT_TLSDESC_PLT_CFNP          "TLSDESC_PLT"
224 @ MSG_DT_TLSDESC_PLT_NF            "tlsdesc_plt"
225 @ MSG_DT_TLSDESC_GOT_CF            "DT_TLSDESC_GOT" # 0x6fffffef7
226 @ MSG_DT_TLSDESC_GOT_CFNP          "TLSDESC_GOT"
227 @ MSG_DT_TLSDESC_GOT_NF            "tlsdesc_got"
228 @ MSG_DT_GNU_CONFLICT_CF           "DT_GNU_CONFLICT" # 0x6fffffef8
229 @ MSG_DT_GNU_CONFLICT_CFNP         "GNU_CONFLICT"
230 @ MSG_DT_GNU_CONFLICT_NF           "gnu_conflict"
231 @ MSG_DT_GNU_LIBLIST_CF            "DT_GNU_LIBLIST" # 0x6fffffef9
232 @ MSG_DT_GNU_LIBLIST_CFNP          "GNU_LIBLIST"
233 @ MSG_DT_GNU_LIBLIST_NF            "gnu_liblist"
234 @ MSG_DT_CONFIG_CF                 "DT_CONFIG" # 0x6ffffefa
235 @ MSG_DT_CONFIG_CFNP               "CONFIG"
236 @ MSG_DT_CONFIG_NF                 "config"
237 @ MSG_DT_DEPAUDIT_CF               "DT_DEPAUDIT" # 0x6ffffefb
238 @ MSG_DT_DEPAUDIT_CFNP             "DEPAUDIT"
239 @ MSG_DT_DEPAUDIT_NF               "depaudit"
240 @ MSG_DT_AUDIT_CF                  "DT_AUDIT" # 0x6ffffefc
241 @ MSG_DT_AUDIT_CFNP                "AUDIT"
242 @ MSG_DT_AUDIT_NF                  "audit"
243 @ MSG_DT_PLTPAD_CF                 "DT_PLTPAD" # 0x6ffffefd
244 @ MSG_DT_PLTPAD_CFNP               "PLTPAD"
245 @ MSG_DT_PLTPAD_NF                 "pltpad"
246 @ MSG_DT_MOVETAB_CF                "DT_MOVETAB" # 0x6ffffefe
247 @ MSG_DT_MOVETAB_CFNP              "MOVETAB"
248 @ MSG_DT_MOVETAB_NF                "movetab"
249 @ MSG_DT_SYMINFO_CF                "DT_SYMINFO" # 0x6ffffeff
250 @ MSG_DT_SYMINFO_CFNP              "SYMINFO"
251 @ MSG_DT_SYMINFO_NF                "syminfo"
252 @ MSG_DT_VERSYM_CF                 "DT_VERSYM" # 0x6ffffeff0
253 @ MSG_DT_VERSYM_CFNP               "VERSYM"
254 @ MSG_DT_VERSYM_NF                 "versym"
255 @ MSG_DT_RELACOUNT_CF              "DT_RELACOUNT" # 0x6ffffeff9
256 @ MSG_DT_RELACOUNT_CFNP            "RELACOUNT"

```



```

257 @ MSG_DT_RELACOUNT_NF          "relaccount"
258 @ MSG_DT_RELACOUNT_CF          "DT_RELACOUNT"          # 0x6fffffff
259 @ MSG_DT_RELACOUNT_CFNFP       "RELACOUNT"
260 @ MSG_DT_RELACOUNT_NF          "relcount"
261 @ MSG_DT_FLAGS_1_CF           "DT_FLAGS_1"          # 0x6fffffff
262 @ MSG_DT_FLAGS_1_CFNFP        "FLAGS_1"
263 @ MSG_DT_FLAGS_1_NF           "flags_1"
264 @ MSG_DT_VERDEF_CF            "DT_VERDEF"           # 0x6fffffff
265 @ MSG_DT_VERDEF_CFNFP         "VERDEF"
266 @ MSG_DT_VERDEF_NF            "verdef"
267 @ MSG_DT_VERDEFNUM_CF         "DT_VERDEFNUM"        # 0x6fffffff
268 @ MSG_DT_VERDEFNUM_CFNFP       "VERDEFNUM"
269 @ MSG_DT_VERDEFNUM_NF         "verdefnum"
270 @ MSG_DT_VERNEED_CF           "DT_VERNEED"          # 0x6fffffff
271 @ MSG_DT_VERNEED_CFNFP        "VERNEED"
272 @ MSG_DT_VERNEED_NF           "verneed"
273 @ MSG_DT_VERNEEDNUM_CF        "DT_VERNEEDNUM"       # 0x6fffffff
274 @ MSG_DT_VERNEEDNUM_CFNFP      "VERNEEDNUM"
275 @ MSG_DT_VERNEEDNUM_NF        "verneednum"
276 @ MSG_DT_SPARC_REGISTER_CF     "DT_SPARC_REGISTER"   # 0x70000001
277 @ MSG_DT_SPARC_REGISTER_CFNFP  "SPARC_REGISTER"
278 @ MSG_DT_SPARC_REGISTER_NF    "sparc_register"
279 @ MSG_DT_SPARC_REGISTER_DMP   "REGISTER"
280 @ MSG_DT_AUXILIARY_CF          "DT_AUXILIARY"        # 0x7fffffff
281 @ MSG_DT_AUXILIARY_CFNFP       "AUXILIARY"
282 @ MSG_DT_AUXILIARY_NF         "auxiliary"
283 @ MSG_DT_USED_CF              "DT_USED"             # 0x7fffffff
284 @ MSG_DT_USED_CFNFP           "USED"
285 @ MSG_DT_USED_NF              "used"
286 @ MSG_DT_FILTER_CF            "DT_FILTER"           # 0x7fffffff
287 @ MSG_DT_FILTER_CFNFP         "FILTER"
288 @ MSG_DT_FILTER_NF            "filter"

291 @ MSG_DF_ORIGIN_CF            "DF_ORIGIN"           # 0x00000001
292 @ MSG_DF_ORIGIN_CFNFP         "ORIGIN"
293 @ MSG_DF_ORIGIN_NF            "origin"
294 @ MSG_DF_SYMBOLIC_CF          "DF_SYMBOLIC"         # 0x00000002
295 @ MSG_DF_SYMBOLIC_CFNFP       "SYMBOLIC"
296 @ MSG_DF_SYMBOLIC_NF         "symbolic"
297 @ MSG_DF_TEXTREL_CF           "DF_TEXTREL"          # 0x00000004
298 @ MSG_DF_TEXTREL_CFNFP        "TEXTREL"
299 @ MSG_DF_TEXTREL_NF           "textrel"
300 @ MSG_DF_BIND_NOW_CF          "DF_BIND_NOW"         # 0x00000008
301 @ MSG_DF_BIND_NOW_CFNFP       "BIND_NOW"
302 @ MSG_DF_BIND_NOW_NF         "bind_now"
303 @ MSG_DF_STATIC_TLS_CF        "DF_STATIC_TLS"       # 0x00000010
304 @ MSG_DF_STATIC_TLS_CFNFP     "STATIC_TLS"
305 @ MSG_DF_STATIC_TLS_NF        "static_tls"

308 @ MSG_DF_1_NOW_CF             "DF_1_NOW"            # 0x00000001
309 @ MSG_DF_1_NOW_CFNFP          "NOW"
310 @ MSG_DF_1_NOW_NF             "now"
311 @ MSG_DF_1_GLOBAL_CF          "DF_1_GLOBAL"         # 0x00000002
312 @ MSG_DF_1_GLOBAL_CFNFP       "GLOBAL"
313 @ MSG_DF_1_GLOBAL_NF          "global"
314 @ MSG_DF_1_GROUP_CF           "DF_1_GROUP"          # 0x00000004
315 @ MSG_DF_1_GROUP_CFNFP        "GROUP"
316 @ MSG_DF_1_GROUP_NF           "group"
317 @ MSG_DF_1_NODELETE_CF        "DF_1_NODELETE"       # 0x00000008
318 @ MSG_DF_1_NODELETE_CFNFP     "NODELETE"
319 @ MSG_DF_1_NODELETE_NF        "nodelete"
320 @ MSG_DF_1_LOADFLTR_CF        "DF_1_LOADFLTR"       # 0x00000010
321 @ MSG_DF_1_LOADFLTR_CFNFP     "LOADFLTR"
322 @ MSG_DF_1_LOADFLTR_NF        "loadfltr"

```

```

323 @ MSG_DF_1_INITFIRST_CF       "DF_1_INITFIRST"     # 0x00000020
324 @ MSG_DF_1_INITFIRST_CFNFP    "INITFIRST"
325 @ MSG_DF_1_INITFIRST_NF       "initfirst"
326 @ MSG_DF_1_NOOPEN_CF         "DF_1_NOOPEN"        # 0x00000040
327 @ MSG_DF_1_NOOPEN_CFNFP       "NOOPEN"
328 @ MSG_DF_1_NOOPEN_NF          "noopen"
329 @ MSG_DF_1_ORIGIN_CF          "DF_1_ORIGIN"        # 0x00000080
330 @ MSG_DF_1_ORIGIN_CFNFP       "ORIGIN"
331 @ MSG_DF_1_ORIGIN_NF          "origin"
332 @ MSG_DF_1_DIRECT_CF          "DF_1_DIRECT"        # 0x00000100
333 @ MSG_DF_1_DIRECT_CFNFP       "DIRECT"
334 @ MSG_DF_1_DIRECT_NF          "direct"
335 @ MSG_DF_1_TRANS_CF           "DF_1_TRANS"         # 0x00000200
336 @ MSG_DF_1_TRANS_CFNFP        "TRANS"
337 @ MSG_DF_1_TRANS_NF           "trans"
338 @ MSG_DF_1_INTERPOSE_CF       "DF_1_INTERPOSE"     # 0x00000400
339 @ MSG_DF_1_INTERPOSE_CFNFP     "INTERPOSE"
340 @ MSG_DF_1_INTERPOSE_NF       "interpose"
341 @ MSG_DF_1_INTERPOSE_DEF      "OBJECT-INTERPOSE"
342 @ MSG_DF_1_NODEFLIB_CF        "DF_1_NODEFLIB"     # 0x00000800
343 @ MSG_DF_1_NODEFLIB_CFNFP     "NODEFLIB"
344 @ MSG_DF_1_NODEFLIB_NF        "nodeflib"
345 @ MSG_DF_1_NODUMP_CF          "DF_1_NODUMP"        # 0x00001000
346 @ MSG_DF_1_NODUMP_CFNFP       "NODUMP"
347 @ MSG_DF_1_NODUMP_NF          "nodump"
348 @ MSG_DF_1_CONFALT_CF         "DF_1_CONFALT"       # 0x00002000
349 @ MSG_DF_1_CONFALT_CFNFP      "CONFALT"
350 @ MSG_DF_1_CONFALT_NF        "confalt"
351 @ MSG_DF_1_ENDFILTEE_CF       "DF_1_ENDFILTEE"     # 0x00004000
352 @ MSG_DF_1_ENDFILTEE_CFNFP    "ENDFILTEE"
353 @ MSG_DF_1_ENDFILTEE_NF       "endfiltee"
354 @ MSG_DF_1_DISPRELDNE_CF      "DF_1_DISPRELDNE"    # 0x00008000
355 @ MSG_DF_1_DISPRELDNE_CFNFP   "DISPRELDNE"
356 @ MSG_DF_1_DISPRELDNE_NF      "dispreldne"
357 @ MSG_DF_1_DISPRELDNE_DEF     "DISPLACE-RELOCS-DONE"
358 @ MSG_DF_1_DISPRELPND_CF      "DF_1_DISPRELPND"    # 0x00010000
359 @ MSG_DF_1_DISPRELPND_CFNFP    "DISPRELPND"
360 @ MSG_DF_1_DISPRELPND_NF      "disprelpnd"
361 @ MSG_DF_1_DISPRELPND_DEF     "DISPLACE-RELOCS-PEND"
362 @ MSG_DF_1_NODIRECT_CF        "DF_1_NODIRECT"      # 0x00020000
363 @ MSG_DF_1_NODIRECT_CFNFP     "NODIRECT"
364 @ MSG_DF_1_NODIRECT_NF        "nodirect"
365 @ MSG_DF_1_IGNMULDEF_CF       "DF_1_IGNMULDEF"     # 0x00040000
366 @ MSG_DF_1_IGNMULDEF_CFNFP    "IGNMULDEF"
367 @ MSG_DF_1_IGNMULDEF_NF       "ignmuldef"
368 @ MSG_DF_1_IGNMULDEF_DEF      "IGNORE-MULDEFS"
369 @ MSG_DF_1_NOKSYMS_CF         "DF_1_NOKSYMS"       # 0x00080000
370 @ MSG_DF_1_NOKSYMS_CFNFP      "NOKSYMS"
371 @ MSG_DF_1_NOKSYMS_NF         "noksyms"
372 @ MSG_DF_1_NOHDR_CF           "DF_1_NOHDR"         # 0x00100000
373 @ MSG_DF_1_NOHDR_CFNFP        "NOHDR"
374 @ MSG_DF_1_NOHDR_NF           "nohdr"
375 @ MSG_DF_1_EDITED_CF          "DF_1_EDITED"        # 0x00200000
376 @ MSG_DF_1_EDITED_CFNFP       "EDITED"
377 @ MSG_DF_1_EDITED_NF         "edited"
378 @ MSG_DF_1_NORELOC_CF         "DF_1_NORELOC"       # 0x00400000
379 @ MSG_DF_1_NORELOC_CFNFP      "NORELOC"
380 @ MSG_DF_1_NORELOC_NF         "noreloc"
381 @ MSG_DF_1_SYMINTPOSE_CF       "DF_1_SYMINTPOSE"    # 0x00800000
382 @ MSG_DF_1_SYMINTPOSE_CFNFP   "SYMINTPOSE"
383 @ MSG_DF_1_SYMINTPOSE_NF      "symintpose"
384 @ MSG_DF_1_SYMINTPOSE_DEF     "SYMBOL-INTERPOSE"
385 @ MSG_DF_1_GLOBAUDIT_CF       "DF_1_GLOBAUDIT"     # 0x01000000
386 @ MSG_DF_1_GLOBAUDIT_CFNFP    "GLOBAUDIT"
387 @ MSG_DF_1_GLOBAUDIT_NF       "globaudit"
388 @ MSG_DF_1_GLOBAUDIT_DEF      "GLOBAL-AUDITING"

```

```
389 @ MSG_DF_1_SINGLETON_CF      "DF_1_SINGLETON"      # 0x02000000
390 @ MSG_DF_1_SINGLETON_CFNP    "SINGLETON"
391 @ MSG_DF_1_SINGLETON_NF      "singleton"
392 @ MSG_DF_1_SINGLETON_DEF     "SINGLETON-EXISTS"

395 @ MSG_DF_P1_LAZYLOAD_CF     "DF_P1_LAZYLOAD"     # 0x00000001
396 @ MSG_DF_P1_LAZYLOAD_CFNP   "LAZYLOAD"
397 @ MSG_DF_P1_LAZYLOAD_NF     "lazyload"
398 @ MSG_DF_P1_LAZYLOAD_DEF    "LAZY"
399 @ MSG_DF_P1_GROUPPERM_CF    "DF_P1_GROUPPERM"   # 0x00000002
400 @ MSG_DF_P1_GROUPPERM_CFNP  "GROUPPERM"
401 @ MSG_DF_P1_GROUPPERM_NF    "groupperm"
402 @ MSG_DF_P1_GROUPPERM_DEF   "GROUP"
403 @ MSG_DF_P1_DEFERRED_CF     "DF_P1_DEFERRED"     # 0x00000004
404 @ MSG_DF_P1_DEFERRED_CFNP   "DEFERRED"
405 @ MSG_DF_P1_DEFERRED_NF     "deferred"
406 @ MSG_DF_P1_DEFERRED_DEF    "DEFERRED"

409 @ MSG_DTF_1_PARINIT_CF     "DTF_1_PARINIT"      # 0x00000001
410 @ MSG_DTF_1_PARINIT_CFNP    "PARINIT"
411 @ MSG_DTF_1_PARINIT_NF     "parinit"
412 @ MSG_DTF_1_CONFEXP_CF     "DTF_1_CONFEXP"     # 0x00000002
413 @ MSG_DTF_1_CONFEXP_CFNP   "CONFEXP"
414 @ MSG_DTF_1_CONFEXP_NF     "confexp"

417 @ MSG_BND_NEEDED           "NEEDED"
418 @ MSG_BND_REFER            "REFERENCED"
419 @ MSG_BND_FILTER            "FILTER"

422 @ MSG_BND_ADDED            "OBJECTS-ADDED"
423 @ MSG_BND_REEVAL           "OBJECTS-REEVALUATED"
424 @ MSG_BND_DELETED          "OBJECTS-DELETED"
425 @ MSG_BND_ATEXIT           "ATEXIT-PROCESSING"
426 @ MSG_BND_REVISIT         "(revisiting)"

428 @ MSG_STR_EMPTY            ""
430 @ MSG_GBL_ZERO             "0"
```

```

*****
65165 Thu Jun 30 22:01:31 2016
new/usr/src/cmd/sgs/libld/common/args.c
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
_____unchanged_portion_omitted_____

916 /*
917 * Parse the -z assert-deflib option. This option can appear in two different
918 * forms:
919 *   -z assert-deflib
920 *   -z assert-deflib=libfred.so
921 *
922 * Either form enables this option, the latter form marks libfred.so as an
923 * exempt library from the check. It is valid to have multiple invocations of
924 * the second form. We silently ignore multiple occurrences of the first form
925 * and multiple invocations of the first form when the second form also occurs.
926 *
927 * We only return false when we have an internal error, such as the failure of
928 * aplist_append. Every other time we return true, but we have the appropriate
929 * fatal flags set because of the ld_eprintf.
930 */
931 static int
932 assdeflib_parse(Of1_desc *of1, char *optarg)
933 {
934     size_t olen, mlen;
935     of1->of1_flags |= FLG_OF_ADEFLIB;

937     olen = strlen(optarg);
938     /* Minimum size of assert-deflib=lib%s.so */
939     mlen = MSG_ARG_ASSDEFLIB_SIZE + 1 + MSG_STR_LIB_SIZE +
940           MSG_STR_SOEXT_SIZE;
941     if (olen > MSG_ARG_ASSDEFLIB_SIZE) {
942         if (optarg[MSG_ARG_ASSDEFLIB_SIZE] != '=') {
943             ld_eprintf(of1, ERR_FATAL, "Missing =\n");
944             ld_eprintf(of1, ERR_FATAL, MSG_INTL(MSG_ARG_ILLEGAL),
945                       MSG_ORIG(MSG_ARG_ASSDEFLIB), optarg);
946         }
947         return (TRUE);
948     }
949     if (strcmp(optarg + MSG_ARG_ASSDEFLIB_SIZE + 1,
950             MSG_ORIG(MSG_STR_LIB), MSG_STR_LIB_SIZE) != 0 ||
951         strcmp(optarg + olen - MSG_STR_SOEXT_SIZE,
952             MSG_ORIG(MSG_STR_SOEXT)) != 0 || olen <= mlen) {
953         ld_eprintf(of1, ERR_FATAL,
954             MSG_INTL(MSG_ARG_ASSDEFLIB_MALFORMED), optarg);
955     }
956     return (TRUE);

957     if (aplist_append(&of1->of1_assdeflib, optarg +
958         MSG_ARG_ASSDEFLIB_SIZE + 1, AL_CNT_ASSDEFLIB) == NULL)
959         return (FALSE);
960 }

962     return (TRUE);
963 }

965 static int     optitle = 0;
966 /*
967 * Parsing options pass1 for process_flags().
968 */
969 static uintptr_t
970 parseopt_pass1(Of1_desc *of1, int argc, char **argv, int *usage)

```

```

971 {
972     int     c, ndx = optind;

974     /*
975     * The -32, -64 and -ztarget options are special, in that we validate
976     * them, but otherwise ignore them. libld.so (this code) is called
977     * from the ld front end program. ld has already examined the
978     * arguments to determine the output class and machine type of the
979     * output object, as reflected in the version (32/64) of ld_main()
980     * that was called and the value of the 'mach' argument passed.
981     * By time execution reaches this point, these options have already
982     * been seen and acted on.
983     */
984     while ((c = ld_getopt(of1->of1_lml, ndx, argc, argv)) != -1) {

986         switch (c) {
987         case '3':
988             DBG_CALL(DBG_args_option(of1->of1_lml, ndx, c, optarg));

990             /*
991             * -32 is processed by ld to determine the output class.
992             * Here we sanity check the option incase some other
993             * -3* option is mistakenly passed to us.
994             */
995             if (optarg[0] != '2')
996                 ld_eprintf(of1, ERR_FATAL,
997                     MSG_INTL(MSG_ARG_ILLEGAL),
998                     MSG_ORIG(MSG_ARG_3), optarg);
999             continue;

1001         case '6':
1002             DBG_CALL(DBG_args_option(of1->of1_lml, ndx, c, optarg));

1004             /*
1005             * -64 is processed by ld to determine the output class.
1006             * Here we sanity check the option incase some other
1007             * -6* option is mistakenly passed to us.
1008             */
1009             if (optarg[0] != '4')
1010                 ld_eprintf(of1, ERR_FATAL,
1011                     MSG_INTL(MSG_ARG_ILLEGAL),
1012                     MSG_ORIG(MSG_ARG_6), optarg);
1013             continue;

1015         case 'a':
1016             DBG_CALL(DBG_args_option(of1->of1_lml, ndx, c, NULL));
1017             aflag = TRUE;
1018             break;

1020         case 'b':
1021             DBG_CALL(DBG_args_option(of1->of1_lml, ndx, c, NULL));
1022             bflag = TRUE;

1024             /*
1025             * This is a hack, and may be undone later.
1026             * The -b option is only used to build the Unix
1027             * kernel and its related kernel-mode modules.
1028             * We do not want those files to get a .SUNW_ldynsym
1029             * section. At least for now, the kernel makes no
1030             * use of .SUNW_ldynsym, and we do not want to use
1031             * the space to hold it. Therefore, we overload
1032             * the use of -b to also imply -znoldynsym.
1033             */
1034             of1->of1_flags |= FLG_OF_NOLDYNSYM;
1035             break;

```

```

1037     case 'c':
1038         DBG_CALL(Dbg_args_option(ofl->ofl_lml, ndx, c, optarg));
1039         if (ofl->ofl_config)
1040             ld_eprintf(ofl, ERR_WARNING_NF,
1041                 MSG_INTL(MSG_ARG_MTONCE),
1042                 MSG_ORIG(MSG_ARG_C));
1043         else
1044             ofl->ofl_config = optarg;
1045         break;
1047     case 'C':
1048         DBG_CALL(Dbg_args_option(ofl->ofl_lml, ndx, c, NULL));
1049         demangle_flag = 1;
1050         break;
1052     case 'd':
1053         DBG_CALL(Dbg_args_option(ofl->ofl_lml, ndx, c, optarg));
1054         if ((optarg[0] == 'n') && (optarg[1] == '\0')) {
1055             if (dflag != SET_UNKNOWN)
1056                 ld_eprintf(ofl, ERR_WARNING_NF,
1057                     MSG_INTL(MSG_ARG_MTONCE),
1058                     MSG_ORIG(MSG_ARG_D));
1059             else
1060                 dflag = SET_FALSE;
1061         } else if ((optarg[0] == 'y') && (optarg[1] == '\0')) {
1062             if (dflag != SET_UNKNOWN)
1063                 ld_eprintf(ofl, ERR_WARNING_NF,
1064                     MSG_INTL(MSG_ARG_MTONCE),
1065                     MSG_ORIG(MSG_ARG_D));
1066             else
1067                 dflag = SET_TRUE;
1068         } else {
1069             ld_eprintf(ofl, ERR_FATAL,
1070                 MSG_INTL(MSG_ARG_ILLEGAL),
1071                 MSG_ORIG(MSG_ARG_D), optarg);
1072         }
1073         break;
1075     case 'e':
1076         DBG_CALL(Dbg_args_option(ofl->ofl_lml, ndx, c, optarg));
1077         if (ofl->ofl_entry)
1078             ld_eprintf(ofl, ERR_WARNING_NF,
1079                 MSG_INTL(MSG_MARG_MTONCE),
1080                 MSG_INTL(MSG_MARG_ENTRY));
1081         else
1082             ofl->ofl_entry = (void *)optarg;
1083         break;
1085     case 'f':
1086         DBG_CALL(Dbg_args_option(ofl->ofl_lml, ndx, c, optarg));
1087         if (ofl->ofl_filtees &&
1088             (!(ofl->ofl_flags & FLG_OF_AUX))) {
1089             ld_eprintf(ofl, ERR_FATAL,
1090                 MSG_INTL(MSG_MARG_INCOMP),
1091                 MSG_INTL(MSG_MARG_FILTER_AUX),
1092                 MSG_INTL(MSG_MARG_FILTER));
1093         } else {
1094             if ((ofl->ofl_filtees =
1095                 add_string(ofl->ofl_filtees, optarg)) ==
1096                 (const char *)S_ERROR)
1097                 return (S_ERROR);
1098             ofl->ofl_flags |= FLG_OF_AUX;
1099         }
1100         break;
1102     case 'F':

```

```

1103         DBG_CALL(Dbg_args_option(ofl->ofl_lml, ndx, c, optarg));
1104         if (ofl->ofl_filtees &&
1105             (ofl->ofl_flags & FLG_OF_AUX)) {
1106             ld_eprintf(ofl, ERR_FATAL,
1107                 MSG_INTL(MSG_MARG_INCOMP),
1108                 MSG_INTL(MSG_MARG_FILTER),
1109                 MSG_INTL(MSG_MARG_FILTER_AUX));
1110         } else {
1111             if ((ofl->ofl_filtees =
1112                 add_string(ofl->ofl_filtees, optarg)) ==
1113                 (const char *)S_ERROR)
1114                 return (S_ERROR);
1115         }
1116         break;
1118     case 'h':
1119         DBG_CALL(Dbg_args_option(ofl->ofl_lml, ndx, c, optarg));
1120         if (ofl->ofl_soname)
1121             ld_eprintf(ofl, ERR_WARNING_NF,
1122                 MSG_INTL(MSG_MARG_MTONCE),
1123                 MSG_INTL(MSG_MARG_SONAME));
1124         else
1125             ofl->ofl_soname = (const char *)optarg;
1126         break;
1128     case 'i':
1129         DBG_CALL(Dbg_args_option(ofl->ofl_lml, ndx, c, NULL));
1130         ofl->ofl_flags |= FLG_OF_IGNENV;
1131         break;
1133     case 'I':
1134         DBG_CALL(Dbg_args_option(ofl->ofl_lml, ndx, c, optarg));
1135         if (ofl->ofl_interp)
1136             ld_eprintf(ofl, ERR_WARNING_NF,
1137                 MSG_INTL(MSG_ARG_MTONCE),
1138                 MSG_ORIG(MSG_ARG_CI));
1139         else
1140             ofl->ofl_interp = (const char *)optarg;
1141         break;
1143     case 'l':
1144         DBG_CALL(Dbg_args_option(ofl->ofl_lml, ndx, c, optarg));
1145         /*
1146          * For now, count any library as a shared object. This
1147          * is used to size the internal symbol cache. This
1148          * value is recalculated later on actual file processing
1149          * to get an accurate shared object count.
1150          */
1151         ofl->ofl_soscnt++;
1152         break;
1154     case 'm':
1155         DBG_CALL(Dbg_args_option(ofl->ofl_lml, ndx, c, NULL));
1156         ofl->ofl_flags |= FLG_OF_GENMAP;
1157         break;
1159     case 'o':
1160         DBG_CALL(Dbg_args_option(ofl->ofl_lml, ndx, c, optarg));
1161         if (ofl->ofl_name)
1162             ld_eprintf(ofl, ERR_WARNING_NF,
1163                 MSG_INTL(MSG_MARG_MTONCE),
1164                 MSG_INTL(MSG_MARG_OUTFILE));
1165         else
1166             ofl->ofl_name = (const char *)optarg;
1167         break;

```

```

1169     case 'p':
1170         DBG_CALL(Dbg_args_option(ofl->ofl_lml, ndx, c, optarg));
1171
1172         /*
1173          * Multiple instances of this option may occur. Each
1174          * additional instance is effectively concatenated to
1175          * the previous separated by a colon.
1176          */
1177         if (*optarg != '\0') {
1178             if ((ofl->ofl_audit =
1179                 add_string(ofl->ofl_audit,
1180                     optarg)) == (const char *)S_ERROR)
1181                 return (S_ERROR);
1182         }
1183         break;
1184
1185     case 'P':
1186         DBG_CALL(Dbg_args_option(ofl->ofl_lml, ndx, c, optarg));
1187
1188         /*
1189          * Multiple instances of this option may occur. Each
1190          * additional instance is effectively concatenated to
1191          * the previous separated by a colon.
1192          */
1193         if (*optarg != '\0') {
1194             if ((ofl->ofl_depaudit =
1195                 add_string(ofl->ofl_depaudit,
1196                     optarg)) == (const char *)S_ERROR)
1197                 return (S_ERROR);
1198         }
1199         break;
1200
1201     case 'r':
1202         DBG_CALL(Dbg_args_option(ofl->ofl_lml, ndx, c, NULL));
1203         rflag = TRUE;
1204         break;
1205
1206     case 'R':
1207         DBG_CALL(Dbg_args_option(ofl->ofl_lml, ndx, c, optarg));
1208
1209         /*
1210          * Multiple instances of this option may occur. Each
1211          * additional instance is effectively concatenated to
1212          * the previous separated by a colon.
1213          */
1214         if (*optarg != '\0') {
1215             if ((ofl->ofl_rpath =
1216                 add_string(ofl->ofl_rpath,
1217                     optarg)) == (const char *)S_ERROR)
1218                 return (S_ERROR);
1219         }
1220         break;
1221
1222     case 's':
1223         DBG_CALL(Dbg_args_option(ofl->ofl_lml, ndx, c, NULL));
1224         sflag = TRUE;
1225         break;
1226
1227     case 't':
1228         DBG_CALL(Dbg_args_option(ofl->ofl_lml, ndx, c, NULL));
1229         ofl->ofl_flags |= FLG_OF_NOWARN;
1230         break;
1231
1232     case 'u':
1233         DBG_CALL(Dbg_args_option(ofl->ofl_lml, ndx, c, optarg));
1234         break;

```

```

1236     case 'z':
1237         DBG_CALL(Dbg_args_option(ofl->ofl_lml, ndx, c, optarg));
1238
1239         /*
1240          * For specific help, print our usage message and exit
1241          * immediately to ensure a 0 return code.
1242          */
1243         if (strncmp(optarg, MSG_ORIG(MSG_ARG_HELP),
1244             MSG_ARG_HELP_SIZE) == 0) {
1245             usage_msg(TRUE);
1246             exit(0);
1247         }
1248
1249         /*
1250          * For some options set a flag - further consistency
1251          * checks will be carried out in check_flags().
1252          */
1253         if ((strncmp(optarg, MSG_ORIG(MSG_ARG_LD32),
1254             MSG_ARG_LD32_SIZE) == 0) ||
1255             (strncmp(optarg, MSG_ORIG(MSG_ARG_LD64),
1256             MSG_ARG_LD64_SIZE) == 0)) {
1257             if (createargv(ofl, usage) == S_ERROR)
1258                 return (S_ERROR);
1259         }
1260         else if (
1261             strcmp(optarg, MSG_ORIG(MSG_ARG_DEFS)) == 0) {
1262             if (zdflag != SET_UNKNOWN)
1263                 ld_eprintf(ofl, ERR_WARNING_NF,
1264                     MSG_INTL(MSG_ARG_MTONCE),
1265                     MSG_ORIG(MSG_ARG_ZDEFNODEF));
1266             else
1267                 zdflag = SET_TRUE;
1268             ofl->ofl_guideflags |= FLG_OFG_NO_DEFS;
1269         } else if (strcmp(optarg,
1270             MSG_ORIG(MSG_ARG_NODEFS)) == 0) {
1271             if (zdflag != SET_UNKNOWN)
1272                 ld_eprintf(ofl, ERR_WARNING_NF,
1273                     MSG_INTL(MSG_ARG_MTONCE),
1274                     MSG_ORIG(MSG_ARG_ZDEFNODEF));
1275             else
1276                 zdflag = SET_FALSE;
1277             ofl->ofl_guideflags |= FLG_OFG_NO_DEFS;
1278         } else if (strcmp(optarg,
1279             MSG_ORIG(MSG_ARG_TEXT)) == 0) {
1280             if (ztflag &&
1281                 (ztflag != MSG_ORIG(MSG_ARG_ZTEXT)))
1282                 ld_eprintf(ofl, ERR_FATAL,
1283                     MSG_INTL(MSG_ARG_INCOMP),
1284                     MSG_ORIG(MSG_ARG_ZTEXT),
1285                     ztflag);
1286             ztflag = MSG_ORIG(MSG_ARG_ZTEXT);
1287         } else if (strcmp(optarg,
1288             MSG_ORIG(MSG_ARG_TEXTOFF)) == 0) {
1289             if (ztflag &&
1290                 (ztflag != MSG_ORIG(MSG_ARG_ZTEXTOFF)))
1291                 ld_eprintf(ofl, ERR_FATAL,
1292                     MSG_INTL(MSG_ARG_INCOMP),
1293                     MSG_ORIG(MSG_ARG_ZTEXTOFF),
1294                     ztflag);
1295             ztflag = MSG_ORIG(MSG_ARG_ZTEXTOFF);
1296         } else if (strcmp(optarg,
1297             MSG_ORIG(MSG_ARG_TEXTWARN)) == 0) {
1298             if (ztflag &&
1299                 (ztflag != MSG_ORIG(MSG_ARG_ZTEXTWARN)))
1300                 ld_eprintf(ofl, ERR_FATAL,

```

```

1301         MSG_INTL(MSG_ARG_INCOMP),
1302         MSG_ORIG(MSG_ARG_ZTEXTWARN),
1303         ztflag);
1304         ztflag = MSG_ORIG(MSG_ARG_ZTEXTWARN);

1306     /*
1307     * For other options simply set the ofl flags directly.
1308     */
1309     } else if (strcmp(optarg,
1310         MSG_ORIG(MSG_ARG_RESCAN)) == 0) {
1311         ofl->ofl_flags1 |= FLG_OF1_RESCAN;
1312     } else if (strcmp(optarg,
1313         MSG_ORIG(MSG_ARG_ABSEXEC)) == 0) {
1314         ofl->ofl_flags1 |= FLG_OF1_ABSEXEC;
1315     } else if (strcmp(optarg,
1316         MSG_ORIG(MSG_ARG_LOADFLTR)) == 0) {
1317         ztflag = TRUE;
1318     } else if (strcmp(optarg,
1319         MSG_ORIG(MSG_ARG_NORELOC)) == 0) {
1320         ofl->ofl_dtflags_1 |= DF_1_NORELOC;
1321     } else if (strcmp(optarg,
1322         MSG_ORIG(MSG_ARG_NOVERSION)) == 0) {
1323         ofl->ofl_flags |= FLG_OF1_NOVERSEC;
1324     } else if (strcmp(optarg,
1325         MSG_ORIG(MSG_ARG_MULDEFS)) == 0) {
1326         ofl->ofl_flags |= FLG_OF1_MULDEFS;
1327     } else if (strcmp(optarg,
1328         MSG_ORIG(MSG_ARG_REDLCSYM)) == 0) {
1329         ofl->ofl_flags |= FLG_OF1_REDLCSYM;
1330     } else if (strcmp(optarg,
1331         MSG_ORIG(MSG_ARG_INITFIRST)) == 0) {
1332         ofl->ofl_dtflags_1 |= DF_1_INITFIRST;
1333     } else if (strcmp(optarg,
1334         MSG_ORIG(MSG_ARG_NODELETE)) == 0) {
1335         ofl->ofl_dtflags_1 |= DF_1_NODELETE;
1336     } else if (strcmp(optarg,
1337         MSG_ORIG(MSG_ARG_NOPARTIAL)) == 0) {
1338         ofl->ofl_flags1 |= FLG_OF1_NOPARTI;
1339     } else if (strcmp(optarg,
1340         MSG_ORIG(MSG_ARG_NOOPEN)) == 0) {
1341         ofl->ofl_dtflags_1 |= DF_1_NOOPEN;
1342     } else if (strcmp(optarg,
1343         MSG_ORIG(MSG_ARG_NOW)) == 0) {
1344         ofl->ofl_dtflags_1 |= DF_1_NOW;
1345     } else if (strcmp(optarg,
1346         MSG_ORIG(MSG_ARG_BIND_NOW)) == 0) {
1347         ofl->ofl_dtflags_1 |= DF_1_BIND_NOW;
1348     } else if (strcmp(optarg,
1349         MSG_ORIG(MSG_ARG_ORIGIN)) == 0) {
1350         ofl->ofl_dtflags_1 |= DF_1_ORIGIN;
1351     } else if (strcmp(optarg,
1352         MSG_ORIG(MSG_ARG_NODEFAULTLIB)) == 0) {
1353         ofl->ofl_dtflags_1 |= DF_1_NODEFLIB;
1354     } else if (strcmp(optarg,
1355         MSG_ORIG(MSG_ARG_NODUMP)) == 0) {
1356         ofl->ofl_dtflags_1 |= DF_1_NODUMP;
1357     } else if (strcmp(optarg,
1358         MSG_ORIG(MSG_ARG_ENDFILTEE)) == 0) {
1359         ofl->ofl_dtflags_1 |= DF_1_ENDFILTEE;
1360     } else if (strcmp(optarg,
1361         MSG_ORIG(MSG_ARG_VERBOSE)) == 0) {
1362         ofl->ofl_flags |= FLG_OF1_VERBOSE;
1363     } else if (strcmp(optarg,
1364         MSG_ORIG(MSG_ARG_COMBRELOC)) == 0) {
1365         ofl->ofl_flags1 |= FLG_OF1_COMREL;
1366     } else if (strcmp(optarg,
1367         MSG_ORIG(MSG_ARG_NOCOMBRELOC)) == 0) {

```

```

1367         ofl->ofl_flags |= FLG_OF1_NOCOMREL;
1368     } else if (strcmp(optarg,
1369         MSG_ORIG(MSG_ARG_NOCOMPSTRTAB)) == 0) {
1370         ofl->ofl_flags1 |= FLG_OF1_NCSTTAB;
1371     } else if (strcmp(optarg,
1372         MSG_ORIG(MSG_ARG_NOINTERP)) == 0) {
1373         ofl->ofl_flags1 |= FLG_OF1_NOINTRP;
1374     } else if (strcmp(optarg,
1375         MSG_ORIG(MSG_ARG_INTERPOSE)) == 0) {
1376         zinflag = TRUE;
1377     } else if (strcmp(optarg,
1378         MSG_ORIG(MSG_ARG_IGNORE)) == 0) {
1379         ofl->ofl_flags1 |= FLG_OF1_IGNPRC;
1380     } else if (strcmp(optarg,
1381         MSG_ORIG(MSG_ARG_RELAXRELOC)) == 0) {
1382         ofl->ofl_flags1 |= FLG_OF1_RLXREL;
1383     } else if (strcmp(optarg,
1384         MSG_ORIG(MSG_ARG_NORELAXRELOC)) == 0) {
1385         ofl->ofl_flags1 |= FLG_OF1_NRLXREL;
1386     } else if (strcmp(optarg,
1387         MSG_ORIG(MSG_ARG_NOLDYNSYM)) == 0) {
1388         ofl->ofl_flags |= FLG_OF1_NOLDYNSYM;
1389     } else if (strcmp(optarg,
1390         MSG_ORIG(MSG_ARG_GLOBAUDIT)) == 0) {
1391         ofl->ofl_dtflags_1 |= DF_1_GLOBAUDIT;
1392     } else if (strcmp(optarg,
1393         MSG_ORIG(MSG_ARG_NOSIGHANDLER)) == 0) {
1394         ofl->ofl_flags1 |= FLG_OF1_NOSGHND;
1395     } else if (strcmp(optarg,
1396         MSG_ORIG(MSG_ARG_SYMBOLCAP)) == 0) {
1397         ofl->ofl_flags |= FLG_OF1_OTOSCAP;

1399     /*
1400     * Check archive group usage
1401     * -z rescan-start ... -z rescan-end
1402     * to ensure they don't overlap and are well formed.
1403     */
1404     } else if (strcmp(optarg,
1405         MSG_ORIG(MSG_ARG_RESCAN_START)) == 0) {
1406         if (ofl->ofl_ars_gsndx == 0) {
1407             ofl->ofl_ars_gsndx = ndx;
1408         } else if (ofl->ofl_ars_gsndx > 0) {
1409             /* Another group is still open */
1410             ld_errprintf(ofl, ERR_FATAL,
1411                 MSG_INTL(MSG_ARG_AR_GRP_OLAP),
1412                 MSG_INTL(MSG_MARG_AR_GRP));
1413             /* Don't report cascading errors */
1414             ofl->ofl_ars_gsndx = -1;
1415         }
1416     } else if (strcmp(optarg,
1417         MSG_ORIG(MSG_ARG_RESCAN_END)) == 0) {
1418         if (ofl->ofl_ars_gsndx > 0) {
1419             ofl->ofl_ars_gsndx = 0;
1420         } else if (ofl->ofl_ars_gsndx == 0) {
1421             /* There was no matching begin */
1422             ld_errprintf(ofl, ERR_FATAL,
1423                 MSG_INTL(MSG_ARG_AR_GRP_BAD),
1424                 MSG_INTL(MSG_MARG_AR_GRP_END),
1425                 MSG_INTL(MSG_MARG_AR_GRP_START));
1426             /* Don't report cascading errors */
1427             ofl->ofl_ars_gsndx = -1;
1428         }

1430     /*
1431     * If -z wrap is seen, enter the symbol to be wrapped
1432     * into the wrap AVL tree.

```

```

1433     */
1434     } else if (strncmp(optarg, MSG_ORIG(MSG_ARG_WRAP),
1435     MSG_ARG_WRAP_SIZE) == 0) {
1436         if (ld_wrap_enter(ofl,
1437         optarg + MSG_ARG_WRAP_SIZE) == NULL)
1438             return (S_ERROR);
1439     } else if (strncmp(optarg, MSG_ORIG(MSG_ARG_ASRLR),
1440     MSG_ARG_ASRLR_SIZE) == 0) {
1441         char *p = optarg + MSG_ARG_ASRLR_SIZE;
1442         if (*p == '\0') {
1443             ofl->ofl_aslr = 1;
1444         } else if (*p == '=') {
1445             p++;
1446
1447             if (strcmp(p,
1448             MSG_ORIG(MSG_ARG_ENABLED)) == 0) {
1449                 ofl->ofl_aslr = 1;
1450             } else if (strcmp(p,
1451             MSG_ORIG(MSG_ARG_DISABLED)) == 0) {
1452                 ofl->ofl_aslr = -1;
1453             } else {
1454                 ld_eprintf(ofl, ERR_FATAL,
1455                 MSG_INTL(MSG_ARG_ILLEGAL),
1456                 MSG_ORIG(MSG_ARG_ZASLR), p);
1457             }
1458         } else {
1459             ld_eprintf(ofl, ERR_FATAL,
1460             MSG_INTL(MSG_ARG_ILLEGAL),
1461             MSG_ORIG(MSG_ARG_Z), optarg);
1462             return (S_ERROR);
1463         }
1464     }
1465 #endif /* ! codereview */
1466     } else if ((strncmp(optarg, MSG_ORIG(MSG_ARG_GUIDE),
1467     MSG_ARG_GUIDE_SIZE) == 0) &&
1468     ((optarg[MSG_ARG_GUIDE_SIZE] == '=' ||
1469     (optarg[MSG_ARG_GUIDE_SIZE] == '\0')))) {
1470         if (!guidance_parse(ofl, optarg))
1471             return (S_ERROR);
1472     } else if (strcmp(optarg,
1473     MSG_ORIG(MSG_ARG_FATWARN)) == 0) {
1474         if (zfwflag == SET_FALSE) {
1475             ld_eprintf(ofl, ERR_WARNING_NF,
1476             MSG_INTL(MSG_ARG_MTONCE),
1477             MSG_ORIG(MSG_ARG_ZFATWNOFATW));
1478         } else {
1479             zfwflag = SET_TRUE;
1480             ofl->ofl_flags |= FLG_OF_FATWARN;
1481         }
1482     } else if (strcmp(optarg,
1483     MSG_ORIG(MSG_ARG_NOFATWARN)) == 0) {
1484         if (zfwflag == SET_TRUE)
1485             ld_eprintf(ofl, ERR_WARNING_NF,
1486             MSG_INTL(MSG_ARG_MTONCE),
1487             MSG_ORIG(MSG_ARG_ZFATWNOFATW));
1488         else
1489             zfwflag = SET_FALSE;
1490
1491     /*
1492     * Process everything related to -z assert-deflib. This
1493     * must be done in pass 1 because it gets used in pass
1494     * 2.
1495     */
1496     } else if (strncmp(optarg, MSG_ORIG(MSG_ARG_ASSDEFLIB),
1497     MSG_ARG_ASSDEFLIB_SIZE) == 0) {
1498         if (assdeflib_parse(ofl, optarg) != TRUE)

```

```

1499         return (S_ERROR);
1500     /*
1501     * The following options just need validation as they
1502     * are interpreted on the second pass through the
1503     * command line arguments.
1504     */
1505     } else if (
1506     strncmp(optarg, MSG_ORIG(MSG_ARG_INITARRAY),
1507     MSG_ARG_INITARRAY_SIZE) &&
1508     strncmp(optarg, MSG_ORIG(MSG_ARG_FINIARRAY),
1509     MSG_ARG_FINIARRAY_SIZE) &&
1510     strncmp(optarg, MSG_ORIG(MSG_ARG_PREINITARRAY),
1511     MSG_ARG_PREINITARRAY_SIZE) &&
1512     strncmp(optarg, MSG_ORIG(MSG_ARG_RTLDINFO),
1513     MSG_ARG_RTLDINFO_SIZE) &&
1514     strncmp(optarg, MSG_ORIG(MSG_ARG_DTRACE),
1515     MSG_ARG_DTRACE_SIZE) &&
1516     strcmp(optarg, MSG_ORIG(MSG_ARG_ALLEXTRT)) &&
1517     strcmp(optarg, MSG_ORIG(MSG_ARG_DFLEXTRT)) &&
1518     strcmp(optarg, MSG_ORIG(MSG_ARG_DIRECT)) &&
1519     strcmp(optarg, MSG_ORIG(MSG_ARG_NODIRECT)) &&
1520     strcmp(optarg, MSG_ORIG(MSG_ARG_GROUPEM)) &&
1521     strcmp(optarg, MSG_ORIG(MSG_ARG_NOGROUPEM)) &&
1522     strcmp(optarg, MSG_ORIG(MSG_ARG_NOLAZYLOAD)) &&
1523     strcmp(optarg, MSG_ORIG(MSG_ARG_NODEFERRED)) &&
1524     strcmp(optarg, MSG_ORIG(MSG_ARG_RECORD)) &&
1525     strcmp(optarg, MSG_ORIG(MSG_ARG_ALTEXEC64)) &&
1526     strcmp(optarg, MSG_ORIG(MSG_ARG_WEAKEXT)) &&
1527     strncmp(optarg, MSG_ORIG(MSG_ARG_TARGET),
1528     MSG_ARG_TARGET_SIZE) &&
1529     strcmp(optarg, MSG_ORIG(MSG_ARG_RESCAN_NOW)) &&
1530     strcmp(optarg, MSG_ORIG(MSG_ARG_DEFERRED))) {
1531         ld_eprintf(ofl, ERR_FATAL,
1532         MSG_INTL(MSG_ARG_ILLEGAL),
1533         MSG_ORIG(MSG_ARG_Z), optarg);
1534     }
1535
1536     break;
1537
1538     case 'D':
1539     /*
1540     * If we have not yet read any input files go ahead
1541     * and process any debugging options (this allows any
1542     * argument processing, entrance criteria and library
1543     * initialization to be displayed). Otherwise, if an
1544     * input file has been seen, skip interpretation until
1545     * process files (this allows debugging to be turned
1546     * on and off around individual groups of files).
1547     */
1548     Dflag = 1;
1549     if (ofl->ofl_objscnt == 0) {
1550         if (dbg_setup(ofl, optarg, 2) == 0)
1551             return (S_ERROR);
1552     }
1553
1554     /*
1555     * A diagnostic can only be provided after dbg_setup().
1556     * As this is the first diagnostic that can be produced
1557     * by ld(1), issue a title for timing and basic output.
1558     */
1559     if ((optitle == 0) && DBG_ENABLED) {
1560         optitle++;
1561         DBG_CALL(Dbg_basic_options(ofl->ofl_lml));
1562     }
1563     DBG_CALL(Dbg_args_option(ofl->ofl_lml, ndx, c, optarg));

```

```

1565         break;
1567     case 'B':
1568         DBG_CALL(Dbg_args_option(ofl->ofl_lml, ndx, c, optarg));
1569         if (strcmp(optarg, MSG_ORIG(MSG_ARG_DIRECT)) == 0) {
1570             if (Bdflag == SET_FALSE) {
1571                 ld_eprintf(ofl, ERR_FATAL,
1572                     MSG_INTL(MSG_ARG_INCOMP),
1573                     MSG_ORIG(MSG_ARG_BNODIRECT),
1574                     MSG_ORIG(MSG_ARG_BDIRECT));
1575             } else {
1576                 Bdflag = SET_TRUE;
1577                 ofl->ofl_guideflags |= FLG_OFG_NO_DB;
1578             }
1579         } else if (strcmp(optarg,
1580             MSG_ORIG(MSG_ARG_NODIRECT)) == 0) {
1581             if (Bdflag == SET_TRUE) {
1582                 ld_eprintf(ofl, ERR_FATAL,
1583                     MSG_INTL(MSG_ARG_INCOMP),
1584                     MSG_ORIG(MSG_ARG_BDIRECT),
1585                     MSG_ORIG(MSG_ARG_BNODIRECT));
1586             } else {
1587                 Bdflag = SET_FALSE;
1588                 ofl->ofl_guideflags |= FLG_OFG_NO_DB;
1589             }
1590         } else if (strcmp(optarg,
1591             MSG_ORIG(MSG_STR_SYMBOLIC)) == 0)
1592             Bsflag = TRUE;
1593         else if (strcmp(optarg, MSG_ORIG(MSG_ARG_REDUCE)) == 0)
1594             ofl->ofl_flags |= FLG_OF_PROCDRED;
1595         else if (strcmp(optarg, MSG_ORIG(MSG_STR_LOCAL)) == 0)
1596             Blflag = TRUE;
1597         else if (strcmp(optarg, MSG_ORIG(MSG_ARG_GROUP)) == 0)
1598             Bgflag = TRUE;
1599         else if (strcmp(optarg,
1600             MSG_ORIG(MSG_STR_ELIMINATE)) == 0)
1601             Beflag = TRUE;
1602         else if (strcmp(optarg,
1603             MSG_ORIG(MSG_ARG_TRANSLATOR)) == 0) {
1604             ld_eprintf(ofl, ERR_WARNING,
1605                 MSG_INTL(MSG_ARG_UNSUPPORTED),
1606                 MSG_ORIG(MSG_ARG_BTRANSLATOR));
1607         } else if (strcmp(optarg,
1608             MSG_ORIG(MSG_STR_LD_DYNAMIC)) &&
1609             strcmp(optarg, MSG_ORIG(MSG_ARG_STATIC))) {
1610             ld_eprintf(ofl, ERR_FATAL,
1611                 MSG_INTL(MSG_ARG_ILLEGAL),
1612                 MSG_ORIG(MSG_ARG_CB), optarg);
1613         }
1614         break;
1616     case 'G':
1617         DBG_CALL(Dbg_args_option(ofl->ofl_lml, ndx, c, NULL));
1618         Gflag = TRUE;
1619         break;
1621     case 'L':
1622         DBG_CALL(Dbg_args_option(ofl->ofl_lml, ndx, c, optarg));
1623         break;
1625     case 'M':
1626         DBG_CALL(Dbg_args_option(ofl->ofl_lml, ndx, c, optarg));
1627         if (aplist_append(&(ofl->ofl_maps), optarg,
1628             AL_CNT_OFL_MAPFILES) == NULL)
1629             return (S_ERROR);
1630         break;

```

```

1632     case 'N':
1633         DBG_CALL(Dbg_args_option(ofl->ofl_lml, ndx, c, optarg));
1634         break;
1636     case 'Q':
1637         DBG_CALL(Dbg_args_option(ofl->ofl_lml, ndx, c, optarg));
1638         if ((optarg[0] == 'n') && (optarg[1] == '\0')) {
1639             if (Qflag != SET_UNKNOWN)
1640                 ld_eprintf(ofl, ERR_WARNING_NF,
1641                     MSG_INTL(MSG_ARG_MTONCE),
1642                     MSG_ORIG(MSG_ARG_CQ));
1643             else
1644                 Qflag = SET_FALSE;
1645         } else if ((optarg[0] == 'y') && (optarg[1] == '\0')) {
1646             if (Qflag != SET_UNKNOWN)
1647                 ld_eprintf(ofl, ERR_WARNING_NF,
1648                     MSG_INTL(MSG_ARG_MTONCE),
1649                     MSG_ORIG(MSG_ARG_CQ));
1650             else
1651                 Qflag = SET_TRUE;
1652         } else {
1653             ld_eprintf(ofl, ERR_FATAL,
1654                 MSG_INTL(MSG_ARG_ILLEGAL),
1655                 MSG_ORIG(MSG_ARG_CQ), optarg);
1656         }
1657         break;
1659     case 'S':
1660         DBG_CALL(Dbg_args_option(ofl->ofl_lml, ndx, c, optarg));
1661         if (aplist_append(&lib_support, optarg,
1662             AL_CNT_SUPPORT) == NULL)
1663             return (S_ERROR);
1664         break;
1666     case 'V':
1667         DBG_CALL(Dbg_args_option(ofl->ofl_lml, ndx, c, NULL));
1668         if (!Vflag)
1669             (void) fprintf(stderr, MSG_ORIG(MSG_STR_STRNL),
1670                 ofl->ofl_sgsid);
1671         Vflag = TRUE;
1672         break;
1674     case 'Y':
1675         DBG_CALL(Dbg_args_option(ofl->ofl_lml, ndx, c, optarg));
1676         if (strncmp(optarg, MSG_ORIG(MSG_ARG_LCOM), 2) == 0) {
1677             if (Llibdir)
1678                 ld_eprintf(ofl, ERR_WARNING_NF,
1679                     MSG_INTL(MSG_ARG_MTONCE),
1680                     MSG_ORIG(MSG_ARG_CYL));
1681             else
1682                 Llibdir = optarg + 2;
1683         } else if (strncmp(optarg,
1684             MSG_ORIG(MSG_ARG_UCOM), 2) == 0) {
1685             if (Ulibdir)
1686                 ld_eprintf(ofl, ERR_WARNING_NF,
1687                     MSG_INTL(MSG_ARG_MTONCE),
1688                     MSG_ORIG(MSG_ARG_CYU));
1689             else
1690                 Ulibdir = optarg + 2;
1691         } else if (strncmp(optarg,
1692             MSG_ORIG(MSG_ARG_PCOM), 2) == 0) {
1693             if (Plibpath)
1694                 ld_eprintf(ofl, ERR_WARNING_NF,
1695                     MSG_INTL(MSG_ARG_MTONCE),
1696                     MSG_ORIG(MSG_ARG_CYP));

```



```

1697         else
1698             Plibpath = optarg + 2;
1699     } else {
1700         ld_eprintf(ofl, ERR_FATAL,
1701             MSG_INTL(MSG_ARG_ILLEGAL),
1702             MSG_ORIG(MSG_ARG_CY), optarg);
1703     }
1704     break;

1706 case '?':
1707     DBG_CALL(Dbg_args_option(ofl->ofl_lml, ndx, c, NULL));
1708     /*
1709      * If the option character is '-', we're looking at a
1710      * long option which couldn't be translated, display a
1711      * more useful error.
1712      */
1713     if (optopt == '-') {
1714         eprintf(ofl->ofl_lml, ERR_FATAL,
1715             MSG_INTL(MSG_ARG_LONG_UNKNOWN),
1716             argv[optind-1]);
1717     } else {
1718         eprintf(ofl->ofl_lml, ERR_FATAL,
1719             MSG_INTL(MSG_ARG_UNKNOWN), optopt);
1720     }
1721     (*usage)++;
1722     break;

1724 default:
1725     break;
1726 }

1728 /*
1729  * Update the argument index for the next getopt() iteration.
1730  */
1731     ndx = optind;
1732 }
1733 return (1);
1734 }

1736 /*
1737  * Parsing options pass2 for
1738  */
1739 static uintptr_t
1740 parseopt_pass2(Of1_desc *ofl, int argc, char **argv)
1741 {
1742     int    c, ndx = optind;

1744     while ((c = ld_getopt(ofl->ofl_lml, ndx, argc, argv)) != -1) {
1745         Ifl_desc    *ifl;
1746         Sym_desc    *sdp;

1748         switch (c) {
1749             case 'l':
1750                 DBG_CALL(Dbg_args_option(ofl->ofl_lml, ndx, c,
1751                     optarg));
1752                 if (ld_find_library(optarg, ofl) == S_ERROR)
1753                     return (S_ERROR);
1754                 break;
1755             case 'B':
1756                 DBG_CALL(Dbg_args_option(ofl->ofl_lml, ndx, c,
1757                     optarg));
1758                 if (strcmp(optarg,
1759                     MSG_ORIG(MSG_STR_LD_DYNAMIC)) == 0) {
1760                     if (ofl->ofl_flags & FLG_OF_DYNAMIC)
1761                         ofl->ofl_flags |=
1762                             FLG_OF_DYNLIBS;

```

```

1763         else {
1764             ld_eprintf(ofl, ERR_FATAL,
1765                 MSG_INTL(MSG_ARG_ST_INCOMP),
1766                 MSG_ORIG(MSG_ARG_BDYNAMIC));
1767         }
1768     } else if (strcmp(optarg,
1769         MSG_ORIG(MSG_ARG_STATIC)) == 0)
1770         ofl->ofl_flags &= ~FLG_OF_DYNLIBS;
1771     break;
1772 case 'L':
1773     DBG_CALL(Dbg_args_option(ofl->ofl_lml, ndx, c,
1774         optarg));
1775     if (ld_add_libdir(ofl, optarg) == S_ERROR)
1776         return (S_ERROR);
1777     break;
1778 case 'N':
1779     DBG_CALL(Dbg_args_option(ofl->ofl_lml, ndx, c,
1780         optarg));
1781     /*
1782      * Record DT_NEEDED string
1783      */
1784     if (!(ofl->ofl_flags & FLG_OF_DYNAMIC))
1785         ld_eprintf(ofl, ERR_FATAL,
1786             MSG_INTL(MSG_ARG_ST_INCOMP),
1787             MSG_ORIG(MSG_ARG_CN));
1788     if (((ifl = libld_calloc(1,
1789         sizeof (Ifl_desc))) == NULL) ||
1790         (aplist_append(&ofl->ofl_sos, ifl,
1791             AL_CNT_OF_LIBS) == NULL))
1792         return (S_ERROR);

1794     ifl->ifl_name = MSG_INTL(MSG_STR_COMMAND);
1795     ifl->ifl_soname = optarg;
1796     ifl->ifl_flags = (FLG_IF_NEEDSTR |
1797         FLG_IF_FILEREF | FLG_IF_DEPREQD);

1799     break;
1800 case 'D':
1801     DBG_CALL(Dbg_args_option(ofl->ofl_lml, ndx, c,
1802         optarg));
1803     (void) dbg_setup(ofl, optarg, 3);
1804     break;
1805 case 'u':
1806     DBG_CALL(Dbg_args_option(ofl->ofl_lml, ndx, c,
1807         optarg));
1808     if (ld_sym_add_u(optarg, ofl,
1809         MSG_STR_COMMAND) == (Sym_desc *)S_ERROR)
1810         return (S_ERROR);
1811     break;
1812 case 'z':
1813     DBG_CALL(Dbg_args_option(ofl->ofl_lml, ndx, c,
1814         optarg));
1815     if ((strcmp(optarg, MSG_ORIG(MSG_ARG_LD32),
1816         MSG_ARG_LD32_SIZE) == 0) ||
1817         (strcmp(optarg, MSG_ORIG(MSG_ARG_LD64),
1818         MSG_ARG_LD64_SIZE) == 0)) {
1819         if (createargv(ofl, 0) == S_ERROR)
1820             return (S_ERROR);
1821     } else if (strcmp(optarg,
1822         MSG_ORIG(MSG_ARG_ALLEXTRT)) == 0) {
1823         ofl->ofl_flags1 |= FLG_OF1_ALLEXRT;
1824         ofl->ofl_flags1 &= ~FLG_OF1_WEAKEXT;
1825     } else if (strcmp(optarg,
1826         MSG_ORIG(MSG_ARG_WEAKEXT)) == 0) {
1827         ofl->ofl_flags1 |= FLG_OF1_WEAKEXT;
1828         ofl->ofl_flags1 &= ~FLG_OF1_ALLEXRT;

```



```

1961         return (S_ERROR);
1962     } else if (++optind < argc)
1963         continue;
1964     }
1965     if (optind >= argc)
1966         break;
1967     ofl->ofl_objscnt++;
1968 }

1970 /* Did an unterminated archive group run off the end? */
1971 if (ofl->ofl_ars_gsndx > 0) {
1972     ld_eprintf(ofl, ERR_FATAL, MSG_INTL(MSG_ARG_AR_GRP_BAD),
1973             MSG_INTL(MSG_MARG_AR_GRP_START),
1974             MSG_INTL(MSG_MARG_AR_GRP_END));
1975     return (S_ERROR);
1976 }

1978 return (1);
1979 }

1981 uintptr_t
1982 ld_process_flags(Ofl_desc *ofl, int argc, char **argv)
1983 {
1984     int     usage = 0;    /* Collect all argument errors before exit */

1986     if (argc < 2) {
1987         usage_mesg(FALSE);
1988         return (S_ERROR);
1989     }

1991     /*
1992      * Option handling
1993      */
1994     opterr = 0;
1995     optind = 1;
1996     if (process_flags_com(ofl, argc, argv, &usage) == S_ERROR)
1997         return (S_ERROR);

1999     /*
2000      * Having parsed everything, did we have any usage errors.
2001      */
2002     if (usage) {
2003         eprintf(ofl->ofl_lml, ERR_FATAL, MSG_INTL(MSG_ARG_USEHELP));
2004         return (S_ERROR);
2005     }

2007     return (check_flags(ofl, argc));
2008 }

2010 /*
2011  * Pass 2 -- process_files: skips the flags collected in pass 1 and processes
2012  * files.
2013  */
2014 static uintptr_t
2015 process_files_com(Ofl_desc *ofl, int argc, char **argv)
2016 {
2017     for (; optind < argc; optind++) {
2018         int     fd;
2019         uintptr_t  open_ret;
2020         char    *path;
2021         Rej_desc  rej = { 0 };

2023         /*
2024          * If we detect some more options return to getopt().
2025          * Checking argv[optind][1] against null prevents a forever
2026          * loop if an unadorned '-' argument is passed to us.

```

```

2027         */
2028         while ((optind < argc) && (argv[optind][0] == '-')) {
2029             if (argv[optind][1] != '\0') {
2030                 if (parseopt_pass2(ofl, argc, argv) == S_ERROR)
2031                     return (S_ERROR);
2032             } else if (++optind < argc)
2033                 continue;
2034         }
2035         if (optind >= argc)
2036             break;

2038         path = argv[optind];
2039         if ((fd = open(path, O_RDONLY)) == -1) {
2040             int err = errno;

2042             ld_eprintf(ofl, ERR_FATAL,
2043                     MSG_INTL(MSG_SYS_OPEN), path, strerror(err));
2044             continue;
2045         }

2047         DBG_CALL(DBG_args_file(ofl->ofl_lml, optind, path));

2049         open_ret = ld_process_open(path, path, &fd, ofl,
2050             (FLG_IF_CMDLINE | FLG_IF_NEEDED), &rej, NULL);
2051         if (fd != -1)
2052             (void) close(fd);
2053         if (open_ret == S_ERROR)
2054             return (S_ERROR);

2056         /*
2057          * Check for mismatched input.
2058          */
2059         if (rej.rej_type) {
2060             Conv_reject_desc_buf_t rej_buf;

2062             ld_eprintf(ofl, ERR_FATAL,
2063                     MSG_INTL(reject[rej.rej_type]),
2064                     rej.rej_name ? rej.rej_name :
2065                     MSG_INTL(MSG_STR_UNKNOWN),
2066                     conv_reject_desc(&rej, &rej_buf,
2067                     ld_targ.t.m.m_mach));
2068             return (1);
2069         }
2070     }
2071     return (1);
2072 }

2074 uintptr_t
2075 ld_process_files(Ofl_desc *ofl, int argc, char **argv)
2076 {
2077     DBG_CALL(DBG_basic_files(ofl->ofl_lml));

2079     /*
2080      * Process command line files (taking into account any applicable
2081      * preceding flags). Return if any fatal errors have occurred.
2082      */
2083     opterr = 0;
2084     optind = 1;
2085     if (process_files_com(ofl, argc, argv) == S_ERROR)
2086         return (S_ERROR);
2087     if (ofl->ofl_flags & FLG_OF_FATAL)
2088         return (1);

2090     /*
2091      * Guidance: Use -B direct/nodirect or -z direct/nodirect.
2092      */

```

```

2093  * This is a backstop for the case where the link had no dependencies.
2094  * Otherwise, it will get caught by ld_process_ifl(). We need both,
2095  * because -z direct is positional, and its value at the time where
2096  * the first dependency is seen might be different than it is now.
2097  */
2098  if ((ofl->ofl_flags & FLG_OF_DYNAMIC) &&
2099      OFL_GUIDANCE(ofl, FLG_OFG_NO_DB)) {
2100      ld_eprintf(ofl, ERR_GUIDANCE, MSG_INTL(MSG_GUIDE_DIRECT));
2101      ofl->ofl_guideflags |= FLG_OFG_NO_DB;
2102  }

2104  /*
2105  * Now that all command line files have been processed see if there are
2106  * any additional 'needed' shared object dependencies.
2107  */
2108  if (ofl->ofl_soneed)
2109      if (ld_finish_libs(ofl) == S_ERROR)
2110          return (S_ERROR);

2112  /*
2113  * If rescanning archives is enabled, do so now to determine whether
2114  * there might still be members extracted to satisfy references from any
2115  * explicit objects. Continue until no new objects are extracted. Note
2116  * that this pass is carried out *after* processing any implicit objects
2117  * (above) as they may already have resolved any undefined references
2118  * from any explicit dependencies.
2119  */
2120  if (ofl->ofl_flags1 & FLG_OF1_RESCAN) {
2121      if (ld_rescan_archives(ofl, 0, argc) == S_ERROR)
2122          return (S_ERROR);
2123      if (ofl->ofl_flags & FLG_OF_FATAL)
2124          return (1);
2125  }

2127  /*
2128  * If debugging, provide statistics on each archives extraction, or flag
2129  * any archive that has provided no members. Note that this could be a
2130  * nice place to free up much of the archive infrastructure, as we've
2131  * extracted any members we need. However, as we presently don't free
2132  * anything under ld(1) there's not much point in proceeding further.
2133  */
2134  DBG_CALL(DBG_statistics_ar(ofl));

2136  /*
2137  * If any version definitions have been established, either via input
2138  * from a mapfile or from the input relocatable objects, make sure any
2139  * version dependencies are satisfied, and version symbols created.
2140  */
2141  if (ofl->ofl_verdesc)
2142      if (ld_vers_check_defs(ofl) == S_ERROR)
2143          return (S_ERROR);

2145  /*
2146  * If input section ordering was specified within some segment
2147  * using a mapfile, verify that the expected sections were seen.
2148  */
2149  if (ofl->ofl_flags & FLG_OF_IS_ORDER)
2150      ld_ent_check(ofl);

2152  return (1);
2153  }

2155  uintptr_t
2156  ld_init_strings(Of1_desc *ofl)
2157  {
2158      uint_t stflags;

```

```

2160      if (ofl->ofl_flags1 & FLG_OF1_NCSTTAB)
2161          stflags = 0;
2162      else
2163          stflags = FLG_STNEW_COMPRESS;

2165      if (((ofl->ofl_shdrsttab = st_new(stflags)) == NULL) ||
2166          ((ofl->ofl_strtab = st_new(stflags)) == NULL) ||
2167          ((ofl->ofl_dynstrtab = st_new(stflags)) == NULL))
2168          return (S_ERROR);

2170      return (0);
2171  }

```



```

258 #
259 # TRANSLATION_NOTE -- End of USAGE message
260 #
261 @ MSG_GRP_INVALIDNDX      "file %s: group section [%u]s: entry %d: \
262                          invalid section index: %d"

264 # Relocation processing messages (some of these are required to satisfy
265 # do_reloc(), which is common code used by cmd/sgs/rtld - make sure both
266 # message files remain consistent).

268 @ MSG_REL_NOFIT           "relocation error: %s: file %s: symbol %s: \
269                          value 0x%llx does not fit"
270 @ MSG_REL_NONALIGN        "relocation error: %s: file %s: symbol %s: \
271                          offset 0x%llx is non-aligned"
272 @ MSG_REL_NULL            "relocation error: file %s: section [%u]s: \
273                          skipping null relocation record"
274 @ MSG_REL_NOTSUP          "relocation error: %s: file %s: section [%u]s: \
275                          relocation not currently supported"
276 @ MSG_REL_PICREDLOC       "relocation error: %s: file %s: symbol %s: \
277                          -z redlocsymb may not be used for pic code"
278 @ MSG_REL_TLSLE           "relocation error: %s: file %s: symbol %s: \
279                          relocation illegal when building a shared object"
280 @ MSG_REL_TLSBND          "relocation error: %s: file %s: symbol %s: \
281                          bound to: %s: relocation illegal when not bound \
282                          to object being created"
283 @ MSG_REL_TLSSTAT         "relocation error: %s: file %s: symbol %s: \
284                          relocation illegal when building a static object"
285 @ MSG_REL_TLSBADSYM       "relocation error: %s: file %s: symbol %s: \
286                          bad symbol type %s: symbol type must be TLS"
287 @ MSG_REL_BADTLS          "relocation error: %s: file %s: symbol %s: \
288                          relocation illegal for TLS symbol"
289 @ MSG_REL_BADGOTBASED     "relocation error: %s: file %s: symbol %s: a GOT \
290                          relative relocation must reference a local symbol"
291 @ MSG_REL_UNKNWSYM        "relocation error: %s: file %s: section [%u]s: \
292                          attempt to relocate with respect to unknown \
293                          symbol %s: offset 0x%llx, symbol index %d"
294 @ MSG_REL_UNSUPSZ         "relocation error: %s: file %s: symbol %s: \
295                          offset size (%d bytes) is not supported"
296 @ MSG_REL_INVALIDOFFSET   "relocation error: %s: file %s: section [%u]s: \
297                          invalid offset symbol '%s': offset 0x%llx"
298 @ MSG_REL_INVALIDRELT     "relocation error: file %s: section [%u]s: \
299                          invalid relocation type: 0x%x"
300 @ MSG_REL_EMPTYSEC        "relocation error: %s: file %s: symbol %s: \
301                          attempted against empty section [%u]s"
302 @ MSG_REL_EXTERNYSYM      "relocation error: %s: file %s: symbol %s: \
303                          external symbolic relocation against non-allocatable \
304                          section %s; cannot be processed at runtime: \
305                          relocation ignored"
306 @ MSG_REL_UNEXPREL        "relocation error: %s: file %s: symbol %s: \
307                          unexpected relocation; generic processing performed"
308 @ MSG_REL_UNEXPSYM        "relocation error: %s: file %s: symbol %s: \
309                          unexpected symbol referenced from file %s"
310 @ MSG_REL_SYMDISC         "relocation error: %s: file %s: section [%u]s: \
311                          symbol %s: symbol has been discarded with discarded \
312                          section: [%u]s"
313 @ MSG_REL_NOSYMBOL        "relocation error: %s: file %s: section: [%u]s: \
314                          offset: 0x%llx: relocation requires reference symbol"
315 @ MSG_REL_DISPREL1        "relocation error: %s: file %s: symbol %s: \
316                          displacement relocation applied to the symbol \
317                          %s at 0x%llx: symbol %s is a copy relocated symbol"
318 @ MSG_REL_UNSUPSIZE       "relocation error: %s: file %s: section [%u]s: \
319                          relocation against section symbol unsupported"

321 @ MSG_REL_DISPREL2        "relocation warning: %s: file %s: symbol %s: \
322                          may contain displacement relocation"

```

```

323 @ MSG_REL_DISPREL3        "relocation warning: %s: file %s: symbol %s: \
324                          displacement relocation applied to the symbol \
325                          %s: at 0x%llx: displacement relocation will not be \
326                          visible in output image"
327 @ MSG_REL_DISPREL4        "relocation warning: %s: file %s: symbol %s: \
328                          displacement relocation to be applied to the symbol \
329                          %s: at 0x%llx: displacement relocation will be \
330                          visible in output image"
331 @ MSG_REL_COPY            "relocation warning: %s: file %s: symbol %s: \
332                          relocation bound to a symbol with STV_PROTECTED \
333                          visibility"
334 @ MSG_RELINVSEC           "relocation warning: %s: file %s: section: [%u]s: \
335                          against suspicious section [%u]s; relocation ignored"
336 @ MSG_REL_TLSIE           "relocation warning: %s: file %s: symbol %s: \
337                          relocation has restricted use when building a shared \
338                          object"

340 @ MSG_REL_SLOPCDATNONAM   "relocation warning: %s: file %s: section [%u]s: \
341                          relocation against discarded COMDAT section [%u]s: \
342                          redirected to file %s"
343 @ MSG_REL_SLOPCDATNAM     "relocation warning: %s: file %s: section [%u]s: \
344                          symbol %s: relocation against discarded COMDAT \
345                          section [%u]s: redirected to file %s"
346 @ MSG_REL_SLOPCDATNOSYM   "relocation warning: %s: file %s: section [%u]s: \
347                          symbol %s: relocation against discarded COMDAT \
348                          section [%u]s: symbol not found, relocation ignored"

350 @ MSG_REL_NOREG           "relocation error: REGISTER relocation not supported \
351                          on target architecture"

353 #
354 # TRANSLATION_NOTE
355 #       The following 7 messages are the message to print the
356 #       following example messages.
357 #
358 #Text relocation remains          referenced
359 #   against symbol                offset   in file
360 #str                               0x14     main.o
361 #printf                            0x1c     main.o
362 #
363 #       The first two lines are the header, and the next msgid
364 #       is the format string for the header.
365 #       Tabs and spaces are used for alignment.
366 #       The first and third %s are for: "Text relocation remains against symbol"
367 #       The second %s and fourth %s are for: "referenced in file"
368 #       The third %s is for: "offset"
369 #
370 @ MSG_REL_REMAIN_FMT_1     "%-40s\t%s\n   %s\t\t   %s\t%s"
371 #
372 # TRANSLATION_NOTE
373 #       The next two msdid make a sentence. So translate:
374 #       "Text relocation remain against symbol"
375 #       And separate them into two msgstr considering the proper
376 #       alignment.
377 @ MSG_REL_RMN_ITM_11       "Text relocation remains"
378 @ MSG_REL_RMN_ITM_12       "against symbol"
379 @ MSG_REL_RMN_ITM_13       "warning: Text relocation remains"

381 @ MSG_REL_RMN_ITM_2        "offset"

383 #
384 # TRANSLATION_NOTE
385 #       The next two msdid make a sentence. So translate:
386 #       "referenced in file"
387 #       And separate them into two msgstr considering the proper
388 #       alignment.

```

```

389 @ MSG_REL_RMN_ITM_31      "referenced"
390 @ MSG_REL_RMN_ITM_32      "in file"
391 @ MSG_REL_REMAIN_2        "%-35s 0x%-8llx\t%s"
392 @ MSG_REL_REMAIN_3        "relocations remain against allocatable but \
393                             non-writable sections"

395 # Files processing messages

397 @ MSG_FIL_MULINC_1        "file %s: attempted multiple inclusion of file"
398 @ MSG_FIL_MULINC_2        "file %s: linked to %s: attempted multiple inclusion \
399                             of file"
400 @ MSG_FIL_SOINSTAT        "input of shared object '%s' in static mode"
401 @ MSG_FIL_INVALSEC        "file %s: section [%u]s has invalid type %s"
402 @ MSG_FIL_NOTFOUND        "file %s: required by %s, not found"
403 @ MSG_FIL_MALSTR          "file %s: section [%u]s: malformed string table, \
404                             initial or final byte"
405 @ MSG_FIL_PTHTOOLONG     "'%s/%s' pathname too long"
406 @ MSG_FIL_EXCLUDE         "file %s: section [%u]s contains both SHF_EXCLUDE and \
407                             SHF_ALLOC flags: SHF_EXCLUDE ignored"
408 @ MSG_FIL_INTERRUPT       "file %s: creation interrupted: %s"
409 @ MSG_FIL_INVRELOC1        "file %s: section [%u]s: relocations can not be \
410                             applied against section [%u]s"
411 @ MSG_FIL_INVSHINFO        "file %s: section [%u]s: has invalid sh_info: %lld"
412 @ MSG_FIL_INVSHLINK        "file %s: section [%u]s: has invalid sh_link: %lld"
413 @ MSG_FIL_INVSHENTSIZE     "file %s: section [%u]s: has invalid sh_entsize: %lld"
414 @ MSG_FIL_NOSTRTABLE      "file %s: section [%u]s: symbol[%d]: specifies string \
415                             table offset 0x%llx: no string table is available"
416 @ MSG_FIL_EXCSTRTABLE     "file %s: section [%u]s: symbol[%d]: specifies string \
417                             table offset 0x%llx: exceeds string table %s: \
418                             size 0x%llx"
419 @ MSG_FIL_NONAMESYM        "file %s: section [%u]s: symbol[%d]: global symbol has \
420                             no name"
421 @ MSG_FIL_UNKCAP           "file %s: section [%u]s: unknown capability tag: %d"
422 @ MSG_FIL_BADSF1          "file %s: section [%u]s: unknown software \
423                             capabilities: 0x%llx; ignored"
424 @ MSG_FIL_INADDR32SF1     "file %s: section [%u]s: software capability ADDR32: is \
425                             ineffective when building 32-bit object; ignored"
426 @ MSG_FIL_EXADDR32SF1     "file %s: section [%u]s: software capability ADDR32: \
427                             requires executable be built with ADDR32 capability"

429 @ MSG_FIL_BADORDREF       "file %s: section [%u]s: contains illegal reference \
430                             to discarded section: [%u]s"

432 # Recording name conflicts

434 @ MSG_REC_OPTCNFLT        "recording name conflict: file '%s' and %s provide \
435                             identical dependency names: %s"
436 @ MSG_REC_OBVCNFLT        "recording name conflict: file '%s' and file '%s' \
437                             provide identical dependency names: %s %s"
438 @ MSG_REC_CNFLTTHINT      "(possible multiple inclusion of the same file)"

440 # System call messages

442 @ MSG_SYS_OPEN            "file %s: open failed: %s"
443 @ MSG_SYS_UNLINK          "file %s: unlink failed: %s"
444 @ MSG_SYS_MMAPANON        "mmap anon failed: %s"
445 @ MSG_SYS_MALLOCC         "malloc failed: %s"

448 # Messages related to platform support

450 @ MSG_TARG_UNSUPPORTED    "unsupported ELF machine type: %s"

453 # ELF processing messages

```

```

455 @ MSG_ELF_LIBELF         "libelf: version not supported: %d"

457 @ MSG_ELF_ARMEM          "file %s: unable to locate archive member;\n\t\
458                             offset=%x, symbol=%s"

460 @ MSG_ELF_ARSYM          "file %s ignored: unable to locate archive symbol table"

462 @ MSG_ELF_VERSYM          "file %s: version symbol section entry mismatch:\n\t\
463                             (section [%u]s entries=%d; section [%u]s entries=%d)"

465 @ MSG_ELF_NOGROUPSECT    "file %s: section [%u]s: SHF_GROUP flag set, but no \
466                             corresponding SHT_GROUP section found"

468 # Section processing errors

470 @ MSG_SCN_NONALLOC        "%s: non-allocatable section '%s' directed to a \
471                             loadable segment: %s"

473 @ MSG_SCN_MULTICOMDAT     "file %s: section [%u]s: cannot be susceptible to multi \
474                             COMDAT mechanisms: %s"

476 @ MSG_SCN_DWFOVRFLW       "%s: section %s: encoded DWARF data exceeds \
477                             section size"
478 @ MSG_SCN_DWFBADENC       "%s: section %s: invalid DWARF encoding: %#x"

480 # Symbol processing errors

482 @ MSG_SYM_NOSECEDEF       "symbol '%s' in file %s has no section definition"
483 @ MSG_SYM_INVSEC          "symbol '%s' in file %s associated with invalid \
484                             section[%lld]"
485 @ MSG_SYM_TLS             "symbol '%s' in file %s (STT_TLS), is defined \
486                             in a non-SHF_TLS section"
487 @ MSG_SYM_BADADDR         "symbol '%s' in file %s: section [%u]s: size %lld: \
488                             symbol (address %lld, size %lld) lies outside \
489                             of containing section"
490 @ MSG_SYM_BADADDR_ROTXT    "symbol '%s' in file %s: readonly text section \
491                             [%u]s: size %lld: symbol (address %lld, \
492                             size %lld) lies outside of containing section"
493 @ MSG_SYM_MULDEF          "symbol '%s' is multiply-defined:"
494 @ MSG_SYM_CONFFVIS         "symbol '%s' has conflicting visibilities:"
495 @ MSG_SYM_DIFFTYPE         "symbol '%s' has differing types:"
496 @ MSG_SYM_DIFFATTR        "symbol '%s' has differing %s:\n\
497                             \t(file %s value=0x%llx; file %s value=0x%llx);"
498 @ MSG_SYM_FILETYPES        "\t(file %s type=%s; file %s type=%s);"
499 @ MSG_SYM_VISTYPES         "\t(file %s visibility=%s; file %s visibility=%s);"
500 @ MSG_SYM_DEFTAKEN         "\t%s definition taken"
501 @ MSG_SYM_DEFUPDATE        "\t%s definition taken and updated with larger size"
502 @ MSG_SYM_LARGER          "\tlargest value applied"
503 @ MSG_SYM_TENTERR          "\ttentative symbol cannot override defined symbol \
504                             of smaller size"

506 @ MSG_SYM_INVSHNDX        "symbol %s has invalid section index; \
507                             ignored:\n\t(file %s value=%s);"
508 @ MSG_SYM_NONGLOB         "global symbol %s has non-global binding:\n\
509                             \t(file %s value=%s);"
510 @ MSG_SYM_RESERVE         "reserved symbol '%s' already defined in file %s"
511 @ MSG_SYM_NOTNULL         "undefined symbol '%s' with non-zero value encountered \
512                             from file %s"
513 @ MSG_SYM_DUPSORTADDR     "section %s: symbol '%s' and symbol '%s' have the \
514                             same address: %lld: remove duplicate with \
515                             NOSORTSYM mapfile directive"

517 @ MSG_PSYM_INVMINFO1      "file %s: section [%u]s: entry[%d] has invalid m_info: \
518                             0x%llx for symbol index"
519 @ MSG_PSYM_INVMINFO2      "file %s: section [%u]s: entry[%d] has invalid m_info: \
520                             0x%llx for size"

```



```

521 @ MSG_PSYM_INVREPEAT "file %s: section [%u]%s: entry[%d] has invalid m_repeat
522 0x%llx"
523 @ MSG_PSYM_CANNOTEXPND "file %s: section [%u]%s: entry[%d] can not be expanded:
524 associated symbol size is unknown %s"
525 @ MSG_PSYM_NOSTATIC "and partial initialization cannot be deferred to \
526 a static object"
527 @ MSG_MOVE_OVERLAP "file %s: section [%u]%s: symbol '%s' overlapping move \
528 initialization: start=0x%llx, length=0x%llx: \
529 start=0x%llx, length=0x%llx"
530 @ MSG_PSYM_EXPREASON1 "output file is static object"
531 @ MSG_PSYM_EXPREASON2 "-z nopartial option in effect"
532 @ MSG_PSYM_EXPREASON3 "move infrastructure size is greater than move data"

534 #
535 # Support library failures
536 #
537 @ MSG_SUP_NOLOAD "dlopen() of support library (%s) failed with \
538 error: %s"
539 @ MSG_SUP_BADVERSION "initialization of support library (%s) failed with \
540 bad version. supported: %d returned: %d"

543 #
544 # TRANSLATION_NOTE
545 # The following 7 messages are the message to print the
546 # following example messages.
547 #
548 #Undefined first referenced
549 # symbol in file
550 #inquire halt_hold.o
551 #
552 @ MSG_SYM_FMT_UNDEF "%s\t\t\t%s\
553 \n %s \t\t\t %s"

555 #
556 # TRANSLATION_NOTE
557 # The next two msdid make a sentence. So translate:
558 # "Undefined symbol"
559 # And separate them into two msgstr considering the proper
560 # alignment.
561 @ MSG_SYM_UNDEF_ITM_11 "Undefined"
562 @ MSG_SYM_UNDEF_ITM_12 "symbol"
563 #
564 # TRANSLATION_NOTE
565 # The next two msdid make a sentence. So translate:
566 # "first referenced in file"
567 # And separate them into two msgstr considering the proper
568 # alignment.
569 @ MSG_SYM_UNDEF_ITM_21 "first referenced"
570 @ MSG_SYM_UNDEF_ITM_22 "in file"
571 #

573 @ MSG_SYM_UND_UNDEF "%-35s %s"
574 @ MSG_SYM_UND_NOVER "%-35s %s (symbol has no version assigned)"
575 @ MSG_SYM_UND_IMPL "%-35s %s (symbol belongs to implicit dependency %s)"
576 @ MSG_SYM_UND_NOTA "%-35s %s (symbol belongs to unavailable version %s \
577 (%s))"
578 @ MSG_SYM_UND_BNDLOCAL "%-35s %s (symbol scope specifies local binding)"

580 @ MSG_SYM_ENTRY "entry point"
581 @ MSG_SYM_UNDEF "%s symbol '%s' is undefined"
582 @ MSG_SYM_EXTERN "%s symbol '%s' is undefined (symbol belongs to \
583 dependency %s)"
584 @ MSG_SYM_NOCRT "symbol '%s' not found, but %s section exists - \
585 possible link-edit without using the compiler driver"

```

```

587 # Output file update messages

589 @ MSG_UPD_NOREADSEG "No read-only segments found. Setting '_etext' to 0"
590 @ MSG_UPD_NORDWRSEG "No read-write segments found. Setting '_edata' to 0"
591 @ MSG_UPD_NOSEG "Setting 'end' and '_end' to 0"

593 @ MSG_UPD_SEGOVERLAP "%s: segment address overlap;\n\
594 \tprevious segment ending at address 0x%llx overlaps\n\
595 \tuser defined segment '%s' starting at address 0x%llx"
596 @ MSG_UPD_LARGSIZE "%s: segment %s calculated size 0x%llx\n\
597 \tis larger than user-defined size 0x%llx"

599 @ MSG_UPD_NOBITS "NOBITS section found before end of initialized data"
600 @ MSG_SEG_FIRNOTLOAD "First segment has type %s, PT_LOAD required: %s"
601 @ MSG_UPD_MULEHFRAME "file %s; section [%u]%s and file %s; section [%u]%s \
602 have incompatible attributes and cannot \
603 be merged into a single output section"

606 # Version processing messages

608 @ MSG_VER_HIGHER "file %s: version revision %d is higher than \
609 expected %d"
610 @ MSG_VER_NOEXIST "file %s: version '%s' does not exist:\n\
611 \trequired by file %s"
612 @ MSG_VER_UNDEF "version '%s' undefined, referenced by version '%s':\n\
613 \trequired by file %s"
614 @ MSG_VER_UNAVAIL "file %s: version '%s' is unavailable:\n\
615 \trequired by file %s"
616 @ MSG_VER_DEFINED "version symbol '%s' already defined in file %s"
617 @ MSG_VER_INVALIDNDX "version symbol '%s' from file %s has an invalid \
618 version index (%d)"
619 @ MSG_VER_ADDVERS "unused $ADDVERS specification from file '%s' \
620 for object '%s'\nversion(s):"
621 @ MSG_VER_ADDVER "\t%s"
622 @ MSG_VER_CYCLIC "following versions generate cyclic dependency:"

624 # Capabilities messages

626 @ MSG_CAP_MULDEF "capabilities symbol '%s' has multiply-defined members:"
627 @ MSG_CAP_MULDEFSYMS "\t(file %s symbol '%s'; file %s symbol '%s');"
628 @ MSG_CAP_REDUNDANT "file %s: section [%u]%s: symbol capabilities \
629 redundant, as object capabilities are more restrictive"
630 @ MSG_CAP_NOSYMSFOUND "no global symbols have been found that are associated \
631 with capabilities identified relocatable objects: \
632 -z symbolcap has no effect"

634 @ MSG_CAPINFO_INVALSYM "file %s: capabilities info section [%u]%s: index %d: \
635 family member symbol '%s': invalid"
636 @ MSG_CAPINFO_INVALLEAD "file %s: capabilities info section [%u]%s: index %d: \
637 family lead symbol '%s': invalid symbol index %d"

639 # Basic strings

641 @ MSG_STR_ALIGNMENTS "alignments"
642 @ MSG_STR_COMMAND "(command line)"
643 @ MSG_STR_TLSREL "(internal TLS relocation requirement)"
644 @ MSG_STR_SIZES "sizes"
645 @ MSG_STR_UNKNOWN "<unknown>"
646 @ MSG_STR_SECTION "%s (section)"
647 @ MSG_STR_SECTION_MSTR "%s (merged string section)"

649 #
650 # TRANSLATION_NOTE
651 # The elf_function name represents a man page reference and should not
652 # be translated.

```

```

653 @ MSG_ELF_BEGIN      "file %s: elf_begin"
654 @ MSG_ELF_CNTL      "file %s: elf_cntl"
655 @ MSG_ELF_GETARHDR   "file %s: elf_getarhdr"
656 @ MSG_ELF_GETARSYM   "file %s: elf_getarsym"
657 @ MSG_ELF_GETDATA    "file %s: elf_getdata"
658 @ MSG_ELF_GETEHDR    "file %s: elf_getehdr"
659 @ MSG_ELF_GETPHDR    "file %s: elf_getphdr"
660 @ MSG_ELF_GETSCN     "file %s: elf_getscn: scnndx: %d"
661 @ MSG_ELF_GETSHDR    "file %s: elf_getshdr"
662 @ MSG_ELF_MEMORY     "file %s: elf_memory"
663 @ MSG_ELF_NDXSCN     "file %s: elf_ndxscn"
664 @ MSG_ELF_NEWDATA    "file %s: elf_newdata"
665 @ MSG_ELF_NEWEHDR    "file %s: elf_newehdr"
666 @ MSG_ELF_NEWSCN     "file %s: elf_newscn"
667 @ MSG_ELF_NEWPHDR    "file %s: elf_newphdr"
668 @ MSG_ELF_STRPTR     "file %s: elf_strptr"
669 @ MSG_ELF_UPDATE     "file %s: elf_update"
670 @ MSG_ELF_SWAP_WRIMAGE "file %s: _elf_swap_wrimage"

673 @ MSG_REJ_MACH      "file %s: wrong ELF machine type: %s"
674 @ MSG_REJ_CLASS     "file %s: wrong ELF class: %s"
675 @ MSG_REJ_DATA      "file %s: wrong ELF data format: %s"
676 @ MSG_REJ_TYPE      "file %s: bad ELF type: %s"
677 @ MSG_REJ_BADFLAG   "file %s: bad ELF flags value: %s"
678 @ MSG_REJ_MISFLAG   "file %s: mismatched ELF flags value: %s"
679 @ MSG_REJ_VERSION   "file %s: mismatched ELF/lib version: %s"
680 @ MSG_REJ_HAL       "file %s: HAL Rl extensions required"
681 @ MSG_REJ_US3       "file %s: Sun UltraSPARC III extensions required"
682 @ MSG_REJ_STR       "file %s: %s"
683 @ MSG_REJ_UNKFILE   "file %s: unknown file type"
684 @ MSG_REJ_UNKCAP    "file=%s; unknown capability: %d"
685 @ MSG_REJ_HWCAP_1   "file %s: hardware capability (CA_SUNW_HW_1) \
686   unsupported: %s"
687 @ MSG_REJ_SFCAP_1   "file %s: software capability (CA_SUNW_SF_1) \
688   unsupported: %s"
689 @ MSG_REJ_MACHCAP   "file %s: machine capability (CA_SUNW_MACH) \
690   unsupported: %s"
691 @ MSG_REJ_PLATCAP   "file %s: platform capability (CA_SUNW_PLAT) \
692   unsupported: %s"
693 @ MSG_REJ_HWCAP_2   "file %s: hardware capability (CA_SUNW_HW_2) \
694   unsupported: %s"
695 @ MSG_REJ_ARCHIVE   "file %s: invalid archive use"

697 # Guidance messages
698 @ MSG_GUIDE_SUMMARY "see ld(1) -z guidance for more information"
699 @ MSG_GUIDE_DEFS    "-z defs option recommended for shared objects"
700 @ MSG_GUIDE_DIRECT  "-B direct or -z direct option recommended before \
701   first dependency"
702 @ MSG_GUIDE_LAZYLOAD "-z lazyload option recommended before \
703   first dependency"
704 @ MSG_GUIDE_MAPFILE "version 2 mapfile syntax recommended: %s"
705 @ MSG_GUIDE_TEXT    "position independent (PIC) code recommended for \
706   shared objects"
707 @ MSG_GUIDE_UNUSED  "removal of unused dependency recommended: %s"

709 @ _END_

712 # The following strings represent reserved names. Reference to these strings
713 # is via the MSG_ORIG() macro, and thus translations are not required.

715 @ MSG_STR_EOF       "<eof>"
716 @ MSG_STR_ERROR     "<error>"
717 @ MSG_STR_EMPTY     ""
718 @ MSG_QSTR_BANG     "'!'"

```

```

719 @ MSG_STR_COLON     ":"
720 @ MSG_QSTR_COLON    "':"
721 @ MSG_QSTR_SEMICOLON "':"
722 @ MSG_QSTR_EQUAL    "'='
723 @ MSG_QSTR_PLUSEQ   "'+='
724 @ MSG_QSTR_MINUSEQ  "'-='
725 @ MSG_QSTR_ATSIGN   "'@'"
726 @ MSG_QSTR_DASH     "'-'
727 @ MSG_QSTR_LEFTBKT  "'{'
728 @ MSG_QSTR_RIGHTBKT "'}'
729 @ MSG_QSTR_PIPE     "'|'"
730 @ MSG_QSTR_STAR     "'*'"
731 @ MSG_STR_DOT       "."
732 @ MSG_STR_SLASH     "/"
733 @ MSG_STR_DYNAMIC   "(.dynamic)"
734 @ MSG_STR_ORIGIN    "$ORIGIN"
735 @ MSG_STR_MACHINE   "$MACHINE"
736 @ MSG_STR_PLATFORM "$PLATFORM"
737 @ MSG_STR_ISALIST   "$ISALIST"
738 @ MSG_STR_OSNAME    "$OSNAME"
739 @ MSG_STR_OSREL     "$OSREL"
740 @ MSG_STR_UU_REAL_U  "__real_"
741 @ MSG_STR_UU_WRAP_U  "__wrap_"
742 @ MSG_STR_UELF32    "_ELF32"
743 @ MSG_STR_UELF64    "_ELF64"
744 @ MSG_STR_USPARC    "_sparc"
745 @ MSG_STR_UX86     "_x86"
746 @ MSG_STR_TRUE      "true"

748 @ MSG_STR_CDIR_ADD  "$add"
749 @ MSG_STR_CDIR_CLEAR "$clear"
750 @ MSG_STR_CDIR_ERROR "$error"
751 @ MSG_STR_CDIR_MFVER "$mapfile_version"
752 @ MSG_STR_CDIR_IF   "$if"
753 @ MSG_STR_CDIR_ELIF "$elif"
754 @ MSG_STR_CDIR_ELSE "$else"
755 @ MSG_STR_CDIR_ENDIF "$endif"

757 @ MSG_STR_GROUP     "GROUP"
758 @ MSG_STR_SUNW_COMDAT "SUNW_COMDAT"

760 @ MSG_FMT_ARMEM     "%s(%s)"
761 @ MSG_FMT_COLPATH   "%s:%s"
762 @ MSG_FMT_SYMNAM    "%s'"
763 @ MSG_FMT_NULLSYMNAM "%s[%d]"
764 @ MSG_FMT_STRCAT    "%s%s"

766 @ MSG_PTH_RTLD     "/usr/lib/ld.so.1"

768 @ MSG_SUNW_OST_SGS "SUNW_OST_SGS"

771 # Section strings

773 @ MSG_SCN_BSS       ".bss"
774 @ MSG_SCN_DATA      ".data"
775 @ MSG_SCN_COMMENT   ".comment"
776 @ MSG_SCN_DEBUG     ".debug"
777 @ MSG_SCN_DEBUG_INFO ".debug_info"
778 @ MSG_SCN_DYNAMIC   ".dynamic"
779 @ MSG_SCN_DYNSYMSORT ".SUNW_dynsymsort"
780 @ MSG_SCN_DYNTLSSORT ".SUNW_dyntlssort"
781 @ MSG_SCN_DYNSTR    ".dynstr"
782 @ MSG_SCN_DYNSYM    ".dynsym"
783 @ MSG_SCN_DYNSYM_SHNDX ".dynsym_shndx"
784 @ MSG_SCN_LDYNSYM   ".SUNW_ldynsym"

```

```

785 @ MSG_SCN_LDYSYM_SHNDX ".SUNW_ldysym_shndx"
786 @ MSG_SCN_EX_SHARED ".ex_shared"
787 @ MSG_SCN_EX_RANGES ".exception_ranges"
788 @ MSG_SCN_EXCL ".excl"
789 @ MSG_SCN_FINI ".fini"
790 @ MSG_SCN_FINIARRAY ".fini_array"
791 @ MSG_SCN_GOT ".got"
792 @ MSG_SCN_GNU_LINKONCE ".gnu.linkonce."
793 @ MSG_SCN_HASH ".hash"
794 @ MSG_SCN_INDEX ".index"
795 @ MSG_SCN_INIT ".init"
796 @ MSG_SCN_INITARRAY ".init_array"
797 @ MSG_SCN_INTERP ".interp"
798 @ MSG_SCN_LBSS ".lbss"
799 @ MSG_SCN_LDATA ".ldata"
800 @ MSG_SCN_LINE ".line"
801 @ MSG_SCN_LRODATA ".lrodata"
802 @ MSG_SCN_PLT ".plt"
803 @ MSG_SCN_PREINITARRAY ".preinit_array"
804 @ MSG_SCN_REL ".rel"
805 @ MSG_SCN_RELA ".rela"
806 @ MSG_SCN_RODATA ".rodata"
807 @ MSG_SCN_SBSS ".sbss"
808 @ MSG_SCN_SBSS2 ".sbss2"
809 @ MSG_SCN_SDATA ".sdata"
810 @ MSG_SCN_SDATA2 ".sdata2"
811 @ MSG_SCN_SHSTRTAB ".shstrtab"
812 @ MSG_SCN_STAB ".stab"
813 @ MSG_SCN_STABEXCL ".stab.exclstr"
814 @ MSG_SCN_STRTAB ".strtab"
815 @ MSG_SCN_SUNWMOVE ".SUNW_move"
816 @ MSG_SCN_SUNWRELOC ".SUNW_reloc"
817 @ MSG_SCN_SUNWSYMINFO ".SUNW_syminfo"
818 @ MSG_SCN_SUNWVERSION ".SUNW_version"
819 @ MSG_SCN_SUNWVERSYM ".SUNW_versym"
820 @ MSG_SCN_SUNWCAP ".SUNW_cap"
821 @ MSG_SCN_SUNWCAPINFO ".SUNW_capinfo"
822 @ MSG_SCN_SUNWCAPCHAIN ".SUNW_capchain"
823 @ MSG_SCN_SYMTAB ".symtab"
824 @ MSG_SCN_SYMTAB_SHNDX ".symtab_shndx"
825 @ MSG_SCN_TBSS ".tbss"
826 @ MSG_SCN_TDATA ".tdata"
827 @ MSG_SCN_TEXT ".text"

829 @ MSG_SYM_FINIARRAY "finiarray"
830 @ MSG_SYM_INITARRAY "initarray"
831 @ MSG_SYM_PREINITARRAY "preinitarray"

833 #
834 # GNU section names
835 #
836 @ MSG_SCN_CTORS ".ctors"
837 @ MSG_SCN_DTORS ".dtors"
838 @ MSG_SCN_EHFRAME ".eh_frame"
839 @ MSG_SCN_EHFRAME_HDR ".eh_frame_hdr"
840 @ MSG_SCN_GCC_X_TBL ".gcc_except_table"
841 @ MSG_SCN_JCR ".jcr"

843 # Segment names for segments referenced by entrance criteria

845 @ MSG_ENT_BSS "bss"
846 @ MSG_ENT_DATA "data"
847 @ MSG_ENT_EXTRA "extra"
848 @ MSG_ENT_LDATA "ldata"
849 @ MSG_ENT_LRODATA "lrodata"
850 @ MSG_ENT_NOTE "note"

```

```

851 @ MSG_ENT_TEXT "text"

853 # Symbol names

855 @ MSG_SYM_START "_start"
856 @ MSG_SYM_MAIN "main"

858 @ MSG_SYM_FINI_U "_fini"
859 @ MSG_SYM_INIT_U "_init"
860 @ MSG_SYM_DYNAMIC "DYNAMIC"
861 @ MSG_SYM_DYNAMIC_U "_DYNAMIC"
862 @ MSG_SYM_EDATA "edata"
863 @ MSG_SYM_EDATA_U "_edata"
864 @ MSG_SYM_END "end"
865 @ MSG_SYM_END_U "_end"
866 @ MSG_SYM_ETEXT "etext"
867 @ MSG_SYM_ETEXT_U "_etext"
868 @ MSG_SYM_GOFITBL "GLOBAL_OFFSET_TABLE_"
869 @ MSG_SYM_GOFITBL_U "_GLOBAL_OFFSET_TABLE_"
870 @ MSG_SYM_PLKTBTL "PROCEDURE_LINKAGE_TABLE_"
871 @ MSG_SYM_PLKTBTL_U "_PROCEDURE_LINKAGE_TABLE_"
872 @ MSG_SYM_TLSETADDR_U "__tls_get_addr"
873 @ MSG_SYM_TLSETADDR_UU "___tls_get_addr"

875 @ MSG_SYM_L_END "END_"
876 @ MSG_SYM_L_END_U "_END_"
877 @ MSG_SYM_L_START "START_"
878 @ MSG_SYM_L_START_U "_START_"

880 # Support functions

882 @ MSG_SUP_VERSION "ld_version"
883 @ MSG_SUP_INPUT_DONE "ld_input_done"

885 @ MSG_SUP_START_64 "ld_start64"
886 @ MSG_SUP_ATEXIT_64 "ld_atexit64"
887 @ MSG_SUP_OPEN_64 "ld_open64"
888 @ MSG_SUP_FILE_64 "ld_file64"
889 @ MSG_SUP_INSEC_64 "ld_input_section64"
890 @ MSG_SUP_SEC_64 "ld_section64"

892 @ MSG_SUP_START "ld_start"
893 @ MSG_SUP_ATEXIT "ld_atexit"
894 @ MSG_SUP_OPEN "ld_open"
895 @ MSG_SUP_FILE "ld_file"
896 @ MSG_SUP_INSEC "ld_input_section"
897 @ MSG_SUP_SEC "ld_section"

899 #
900 # Message previously in 'ld'
901 #
902 #
903 @ _START_

905 # System error messages

907 @ MSG_SYS_STAT "file %s: stat failed: %s"
908 @ MSG_SYS_READ "file %s: read failed: %s"
909 @ MSG_SYS_NOTREG "file %s: is not a regular file"

911 # Argument processing messages

913 @ MSG_ARG_DY_INCOMP "%s option is incompatible with building a dynamic \
executable"
914
915 @ MSG_MARG_DY_INCOMP "%s is incompatible with building a dynamic \
executable"
916

```



```

1049 @ MSG_MAP_BADAUTORED "%s: %llu: auto-reduction ('*'') can only be used in \
1050 hidden/local, or eliminate scope"
1051 @ MSG_MAP_BADFLAG "%s: %llu: badly formed section flags '%s'"
1052 @ MSG_MAP_BADNAME "%s: %llu: basename cannot contain path \
1053 separator ('/'): %s"
1054 @ MSG_MAP_BADONAME "%s: %llu: object name cannot contain path \
1055 separator ('/'): %s"
1056 @ MSG_MAP_REDEFATT "%s: %llu: redefining %s attribute for '%s'"
1057 @ MSG_MAP_PREMEOF "%s: %llu: premature EOF"
1058 @ MSG_MAP_ILLCCHAR "%s: %llu: illegal character '\\%03o'"
1059 @ MSG_MAP_MALFORM "%s: %llu: malformed entry"
1060 @ MSG_MAP_NONLOAD "%s: %llu: %s not allowed on non-LOAD segments"
1061 @ MSG_MAP_NOSTACK1 "%s: %llu: %s not allowed on STACK segment"
1062 @ MSG_MAP_MOREONCE "%s: %llu: %s set more than once on same line"
1063 @ MSG_MAP_NOTERM "%s: %llu: unterminated quoted string: %s"
1064 @ MSG_MAP_SECINSEG "%s: %llu: section within segment ordering done on \
1065 a non-existent segment '%s'"
1066 @ MSG_MAP_UNEXINHERIT "%s: %llu: unnamed version cannot inherit from other \
1067 versions: %s"
1068 @ MSG_MAP_UNEXTOK "%s: %llu: unexpected occurrence of '%c' token"

1070 @ MSG_MAP_SEGEMPLOAD "%s: %llu: empty segment must be of type LOAD or NULL"
1071 @ MSG_MAP_SEGEMPEXE "%s: %llu: a LOAD empty segment definition is only \
1072 allowed when creating a dynamic executable"
1073 @ MSG_MAP_SEGEMPATT "%s: %llu: a LOAD empty segment must have an address \
1074 and size"
1075 @ MSG_MAP_SEGEMPNOAT "%s: %llu: a NULL empty segment must not have an \
1076 address or size"
1077 @ MSG_MAP_SEGEMPSEC "%s: %llu: empty segment can not have sections \
1078 assigned to it"
1079 @ MSG_MAP_SEGEMNOPERM "%s: %llu: empty segment must not have \
1080 p_flags set: 0x%x"

1082 @ MSG_MAP_CNTADDRORDER "%s: %llu: segment cannot have an explicit address \
1083 and also be in the SEGMENT_ORDER list: %s"
1084 @ MSG_MAP_CNTDISSEG "%s: %llu: segment cannot be disabled: %s"
1085 @ MSG_MAP_DUPNAMENT "%s: %llu: cannot redefine entrance criteria: %s"
1086 @ MSG_MAP_DUPORDSEG "%s: %llu: segment is already in %s list: %s"
1087 @ MSG_MAP_DUP_OS_ORD "%s: %llu: section is already in OS_ORDER list: %s"
1088 @ MSG_MAP_DUP_IS_ORD "%s: %llu: entrance criteria is already in \
1089 IS_ORDER list: %s"
1090 @ MSG_MAP_UNKENT "%s: %llu: unknown entrance criteria \
1091 (ASSIGN_SECTION): %s"
1092 @ MSG_MAP_UNKSEG "%s: %llu: unknown segment: %s"
1093 @ MSG_MAP_UNKSYMDEF "%s: %llu: unknown symbol definition: %s"
1094 @ MSG_MAP_UNKSEGTYPE "%s: %llu: unknown internal segment type %d"
1095 @ MSG_MAP_UNKSOTYP "%s: %llu: unknown shared object type: %s"
1096 @ MSG_MAP_UNKSEGATT "%s: %llu: unknown segment attribute: %s"
1097 @ MSG_MAP_UNKSEGFLG "%s: %llu: unknown segment flag: %c"
1098 @ MSG_MAP_UNKSECTYP "%s: %llu: unknown section type: %s"

1100 @ MSG_MAP_SEGSIZE "%s: %lld: existing segment size symbols cannot \
1101 be reset: %s"
1102 @ MSG_MAP_SEGADDR "%s: %llu: segment address or length '%s' %s"
1103 @ MSG_MAP_BADCAPVAL "%s: %llu: bad capability value: %s"
1104 @ MSG_MAP_UNKCAPATTR "%s: %llu: unknown capability attribute '%s'"
1105 @ MSG_MAP_EMPTYCAP "%s: %llu: empty capability definition; ignored"

1107 @ MSG_MAP_SYMDEF1 "%s: %llu: symbol '%s' is already defined in file: \
1108 %s"
1109 @ MSG_MAP_SYMDEF2 "%s: %llu: symbol '%s': %s"

1111 @ MSG_MAP_EXPSCOL "%s: %llu: expected a ';' "
1112 @ MSG_MAP_EXPEQU "%s: %llu: expected a '=', ':', '|', or '@' "
1113 @ MSG_MAP_EXPSEGATT "%s: %llu: expected one or more segment attributes \
1114 after an '=' "

```

```

1115 @ MSG_MAP_EXPSEGNAM "%s: %llu: expected a segment name at the beginning \
1116 of a line"
1117 @ MSG_MAP_EXPSEGTYPE "%s: %llu: %s segment cannot be used with %s \
1118 directive: %s"
1119 @ MSG_MAP_EXPSYM_1 "%s: %llu: expected a symbol name after '@' "
1120 @ MSG_MAP_EXPSYM_2 "%s: %llu: expected a symbol name after '{' "
1121 @ MSG_MAP_EXPSEC "%s: %llu: expected a section name after '|' "
1122 @ MSG_MAP_EXPSO "%s: %llu: expected a shared object definition \
1123 after '-' "
1124 @ MSG_MAP_MULTFILTEE "%s: %llu: multiple filtee definitions are unsupported"
1125 @ MSG_MAP_NOFILTER "%s: %llu: filtee definition required"
1126 @ MSG_MAP_BADSF1 "%s: %llu: unknown software capabilities: 0x%llx; \
1127 ignored"
1128 @ MSG_MAP_INADDR32SF1 "%s: %llu: software capability ADDR32: is ineffective \
1129 when building 32-bit object: ignored"
1130 @ MSG_MAP_NOINTPOSE "%s: %llu: interposition symbols can only be defined \
1131 when building a dynamic executable"
1132 @ MSG_MAP_NOEXVLSZ "%s: %llu: value and size attributes are incompatible \
1133 with extern or parent symbols"
1134 @ MSG_MAP_FLTR_ONLYAVL "%s: %llu: symbol filtering is only available when \
1135 building a shared object"

1137 @ MSG_MAP_SEGSAME "segments '%s' and '%s' have the same assigned \
1138 virtual address"
1139 @ MSG_MAP_EXCLIMIT "exceeds internal limit"
1140 @ MSG_MAP_NOBADFRM "number is badly formed"

1142 @ MSG_MAP_SEGTYPE "segment type"
1143 @ MSG_MAP_SEGVADDR "segment virtual address"
1144 @ MSG_MAP_SEGPHYS "segment physical address"
1145 @ MSG_MAP_SEGLEN "segment length"
1146 @ MSG_MAP_SEGFLAG "segment flags"
1147 @ MSG_MAP_SEGALIGN "segment alignment"
1148 @ MSG_MAP_SEGROUND "segment rounding"

1150 @ MSG_MAP_SECTYP "section type"
1151 @ MSG_MAP_SECFLAG "section flags"
1152 @ MSG_MAP_SECNAM "section name"

1154 @ MSG_MAP_SYMVAL "symbol value"
1155 @ MSG_MAP_SYMSIZE "symbol size"

1157 @ MSG_MAP_DIFF_SYMVAL "symbol values differ"
1158 @ MSG_MAP_DIFF_SYMSZ "symbol sizes differ"
1159 @ MSG_MAP_DIFF_SYMTYP "symbol types differ"
1160 @ MSG_MAP_DIFF_SYMNDX "symbol indexes differ"
1161 @ MSG_MAP_DIFF_SYMLCL "symbol scope conflict against local and non-local"
1162 @ MSG_MAP_DIFF_SYMGLOB "symbol scope conflict against singleton/exported"
1163 @ MSG_MAP_DIFF_SYMPROT "symbol scope conflict against protected"
1164 @ MSG_MAP_DIFF_SYMVER "symbol version conflict"
1165 @ MSG_MAP_DIFF_SYMMUL "symbol multiple definition"
1166 @ MSG_MAP_DIFF_SNGLDIR "singleton scope and direct declaration are \
1167 incompatible"
1168 @ MSG_MAP_DIFF_PROTNDIR "protected scope and no-direct declaration \
1169 are incompatible"

1172 @ MSG_MAP_SECORDER "section ordering requested, but no matching section \
1173 found: segment: %s section: %s"

1176 # Mapfile Directives

1178 @ MSG_MAP_EXP_ATTR "%s: %llu: expected attribute name (%s), or \
1179 terminator (';', '|'): %s"
1180 @ MSG_MAP_EXP_CAPMASK "%s: %llu: expected capability name, integer value, or \

```

```

1181 terminator (';', '}'): %s"
1182 @ MSG_MAP_EXP_CAPNAME "%s: %llu: expected name, or terminator (';', '}'): %s"
1183 @ MSG_MAP_EXP_CAPID "%s: %llu: expected name, or '{' following %s: %s"
1184 @ MSG_MAP_EXP_CAPHW "%s: %llu: expected hardware capability, or \
1185 terminator (';', '}'): %s"
1186 @ MSG_MAP_EXP_CAPSF "%s: %llu: expected software capability, or \
1187 terminator (';', '}'): %s"
1188 @ MSG_MAP_EXP_EQ "%s: %llu: expected '=' following %s: %s"
1189 @ MSG_MAP_EXP_EQ_ALL "%s: %llu: expected '=', '+=' , or '-=' following %s: %s"
1190 @ MSG_MAP_EXP_EQ_PEQ "%s: %llu: expected '=' following %s: %s"
1191 @ MSG_MAP_EXP_DIR "%s: %llu: expected mapfile directive (%s): %s"
1192 @ MSG_MAP_SFLG_EXBANG "%s: %llu: '!' appears without corresponding flag"
1193 @ MSG_MAP_EXP_FILNAM "%s: %llu: expected file name following %s: %s"
1194 @ MSG_MAP_EXP_FILPATH "%s: %llu: expected file path following %s: %s"
1195 @ MSG_MAP_EXP_INT "%s: %llu: expected integer value following %s: %s"
1196 @ MSG_MAP_EXP_LBKT "%s: %llu: expected '{' following %s: %s"
1197 @ MSG_MAP_EXP_OBJNAM "%s: %llu: expected object name following %s: %s"
1198 @ MSG_MAP_SFLG_ONEBANG "%s: %llu: '!' can only be specified once per flag"
1199 @ MSG_MAP_EXP_SECFLAG "%s: %llu: expected section flag (%s), '!', or \
1200 terminator (';', '}'): %s"
1201 @ MSG_MAP_EXP_SECNAM "%s: %llu: expected section name following %s: %s"
1202 @ MSG_MAP_EXP_SEGFLAG "%s: %llu: expected segment flag (%s), or \
1203 terminator (';', '}'): %s"
1204 @ MSG_MAP_EXP_ECNAM "%s: %llu: expected entrance criteria (ASSIGN_SECTION) \
1205 name, or terminator (';', '}'): %s"
1206 @ MSG_MAP_EXP_SEGNAM "%s: %llu: expected segment name following %s: %s"
1207 @ MSG_MAP_EXP_SEM "%s: %llu: expected ';' to terminate %s: %s"
1208 @ MSG_MAP_EXP_SEMLBKT "%s: %llu: expected ';' or '{' following %s: %s"
1209 @ MSG_MAP_EXP_SEMRBKT "%s: %llu: expected ';' or '}' to terminate %s: %s"
1210 @ MSG_MAP_EXP_SHTYPE "%s: %llu: expected section type: %s"
1211 @ MSG_MAP_EXP_SYM "%s: %llu: expected symbol name, symbol scope, \
1212 or '*': %s"
1213 @ MSG_MAP_EXP_SYMEND "%s: %llu: expected inherited version name, or \
1214 terminator (';'): %s"
1215 @ MSG_MAP_EXP_SYMDELIM "%s: %llu: expected one of ':', ';', or '{': %s"
1216 @ MSG_MAP_EXP_SYMFLAG "%s: %llu: expected symbol flag (%s), or \
1217 terminator (';', '}'): %s"
1218 @ MSG_MAP_EXP_SYMNAM "%s: %llu: expected symbol name following %s: %s"
1219 @ MSG_MAP_EXP_SYMSCOPE "%s: %llu: expected symbol scope (%s): %s"
1220 @ MSG_MAP_EXP_SYMTYPE "%s: %llu: expected symbol type (%s): %s"
1221 @ MSG_MAP_EXP_VERSION "%s: %llu: expected version name following %s: %s"
1222 @ MSG_MAP_BADEXTRA "%s: %llu: unexpected text found following %s directive"
1223 @ MSG_MAP_VALUELIMIT "%s: %llu: numeric value exceeds word size: %s"
1224 @ MSG_MAP_MALVALUE "%s: %llu: malformed numeric value: %s"
1225 @ MSG_MAP_BADVALUETAIL "%s: %llu: unexpected characters following numeric \
1226 constant: %s"
1227 @ MSG_MAP_WSNEEDED "%s: %llu: whitespace needed before token: %s"
1228 @ MSG_MAP_BADCHAR "%s: %llu: unexpected text: %s"
1229 @ MSG_MAP_BADKWQUOTE "%s: %llu: mapfile keywords should not be quoted: %s"
1230 @ MSG_MAP_CDIRE_NOTBOL "%s: %llu: mapfile control directive not at start of \
1231 line: %s"
1232 @ MSG_MAP_NOATTR "%s: %llu: %s specified no attributes (empty {})"
1233 @ MSG_MAP_NOVALUES "%s: %llu: %s specified without values"
1234 @ MSG_MAP_INTERR "<internal error>"
1235 @ MSG_MAP_ISORDVER "%s: %llu: version 0 mapfile ?O flag and version 1 \
1236 segment IS_ORDER attribute are mutually exclusive: %s"
1237 @ MSG_MAP_SYMATTR "symbol attributes";

```

1239 # Mapfile Control Directives

```

1241 @ MSG_MAP_CDIRE_BADVDIR "%s: %llu: $mapfile_version directive must specify \
1242 version 2 or higher: %d"
1243 @ MSG_MAP_CDIRE_BADVER "%s: %llu: unknown mapfile version: %d"
1244 @ MSG_MAP_CDIRE_REPVER "%s: %llu: $mapfile_version must be first directive \
1245 in file"
1246 @ MSG_MAP_CDIRE_REQARG "%s: %llu: %s directive requires an argument"

```

```

1247 @ MSG_MAP_CDIRE_REQNOARG "%s: %llu: %s directive does not accept arguments"
1248 @ MSG_MAP_CDIRE_BAD "%s: %llu: unrecognized mapfile control directive"
1249 @ MSG_MAP_CDIRE_NOIF "%s: %llu: %s directive used without opening $if"
1250 @ MSG_MAP_CDIRE_ELSE "%s: %llu: %s directive preceded by $else on line %d"
1251 @ MSG_MAP_CDIRE_NOEND "%s: %llu: EOF encountered without closing $endif \
1252 for $if on line %d"
1253 @ MSG_MAP_CDIRE_ERROR "%s: %llu: error: %s"

1256 # Mapfile Conditional Expressions

1258 @ MSG_MAP_CEXP_TOKERR "%s: %llu: syntax error in conditional expression at: %s"
1259 @ MSG_MAP_CEXP_SEMERR "%s: %llu: malformed conditional expression"
1260 @ MSG_MAP_CEXP_BADOPUSE "%s: %llu: invalid operator use in conditional \
1261 expression"
1262 @ MSG_MAP_CEXP_UNBALPAR "%s: %llu: unbalanced parenthesis in conditional \
1263 expression"
1264 @ MSG_MAP_BADCESC "%s: %llu: unrecognized escape in double quoted \
1265 token: \\c\n"

1267 # Generic error diagnostic labels

1269 @ MSG_STR_NULL "(null)"

1271 @ MSG_DBG_DFLT_FMT "debug: "
1272 @ MSG_DBG_AOUT_FMT "debug: a.out: "
1273 @ MSG_DBG_NAME_FMT "debug: %s: "

1275 # -z assert-deflib strings

1277 @ MSG_ARG_ASSDEFLIB_MALFORMED "library name malformed: %s"
1278 @ MSG_ARG_ASSDEFLIB_FOUND "dynamic library found on default search path \
1279 (%s): lib%s.so"

1281 @ _END_

1284 # Software identification. Note, the SGU strings is historic, and has
1285 # little relevance. It is preserved as applications have used this
1286 # string to identify the Solaris link-editor.

1288 @ MSG_SGS_ID "ld: Software Generation Utilities - \
1289 Solaris Link Editors: "

1291 # The following strings represent reserved words, files, pathnames and symbols.
1292 # Reference to this strings is via the MSG_ORIG() macro, and thus no message
1293 # translation is required.

1295 @ MSG_DBG_FOPEN_MODE "w"

1297 @ MSG_DBG_CLS32_FMT "32: "
1298 @ MSG_DBG_CLS64_FMT "64: "

1300 @ MSG_STR_PATHCHK ";"
1301 @ MSG_STR_AOUT "a.out"

1303 @ MSG_STR_LIB_A "%s/lib%s.a"
1304 @ MSG_STR_LIB_SO "%s/lib%s.so"
1305 @ MSG_STR_PATH "%s/%s"
1306 @ MSG_STR_STRNL "%s\n"
1307 @ MSG_STR_NL "\n"
1308 @ MSG_STR_CAPGROUPID "CAP_GROUP_%d"

1310 @ MSG_STR_LD_DYNAMIC "dynamic"
1311 @ MSG_STR_SYMBOLIC "symbolic"
1312 @ MSG_STR_ELIMINATE "eliminate"

```

```

1313 @ MSG_STR_LOCAL          "local"
1314 @ MSG_STR_PROGBITS       "progbits"
1315 @ MSG_STR_SYMTAB         "symtab"
1316 @ MSG_STR_DYNSYM         "dynsym"
1317 @ MSG_STR_REL            "rel"
1318 @ MSG_STR_RELA           "rela"
1319 @ MSG_STR_STRTAB         "strtab"
1320 @ MSG_STR_HASH           "hash"
1321 @ MSG_STR_LIB            "lib"
1322 @ MSG_STR_NOTE           "note"
1323 @ MSG_STR_NOBITS        "nobits"
1324 @ MSG_STR_HWCAP_1        "hwcap_1"
1325 @ MSG_STR_SFCAP_1        "sfcap_1"
1326 @ MSG_STR_SOEXT          ".so"

1328 @ MSG_STR_OPTIONS        "3:6:abc:d:e:f:h:il:mo:p:rstu:z:B:CD:F:GI:L:M:N:P:Q:R:\
1329                          S:VW:Y:?"

1331 # Argument processing strings

1333 @ MSG_ARG_3                "-3"
1334 @ MSG_ARG_6                "-6"
1335 @ MSG_ARG_A                "-a"
1336 @ MSG_ARG_B                "-b"
1337 @ MSG_ARG_CB               "-B"
1338 @ MSG_ARG_BDIRECT          "-Bdirect"
1339 @ MSG_ARG_BDYNAMIC         "-Bdynamic"
1340 @ MSG_ARG_BELIMINATE       "-Beliminate"
1341 @ MSG_ARG_BGROUP           "-Bgroup"
1342 @ MSG_ARG_BLOCAL           "-Blocal"
1343 @ MSG_ARG_BNODIRECT        "-Bnodirect"
1344 @ MSG_ARG_BSYMBOLIC        "-Bsymbolic"
1345 @ MSG_ARG_BTRANSLATOR      "-Btranslator"
1346 @ MSG_ARG_C                "-c"
1347 @ MSG_ARG_D                "-d"
1348 @ MSG_ARG_DY               "-dy"
1349 @ MSG_ARG_CI               "-I"
1350 @ MSG_ARG_CN               "-N"
1351 @ MSG_ARG_P                "-p"
1352 @ MSG_ARG_CP               "-P"
1353 @ MSG_ARG_CQ               "-Q"
1354 @ MSG_ARG_CY               "-Y"
1355 @ MSG_ARG_CYL              "-YL"
1356 @ MSG_ARG_CYP              "-YP"
1357 @ MSG_ARG_CYU              "-YU"
1358 @ MSG_ARG_Z                "-z"
1359 @ MSG_ARG_ZDEFNODEF        "-z[defs|nodefs]"
1360 @ MSG_ARG_ZASLR            "-zaslr"
1361 #endif /* ! codereview */
1362 @ MSG_ARG_ZGUIDE           "-zguidance"
1363 @ MSG_ARG_ZNODEF           "-znodefs"
1364 @ MSG_ARG_ZNOINTERP        "-znointerp"
1365 @ MSG_ARG_ZRELAXRELOC      "-zrelaxreloc"
1366 @ MSG_ARG_ZNORELAXRELOC    "-znorelaxreloc"
1367 @ MSG_ARG_ZTEXT            "-ztext"
1368 @ MSG_ARG_ZTEXTOFF         "-ztextoff"
1369 @ MSG_ARG_ZTEXTWARN        "-ztextwarn"
1370 @ MSG_ARG_ZTEXTALL         "-z[text|textwarn|textoff]"
1371 @ MSG_ARG_ZLOADFLTR         "-zloadfltr"
1372 @ MSG_ARG_ZCOMBRELOC        "-zcombreloc"
1373 @ MSG_ARG_ZSYMBOLCAP        "-zsymbolcap"
1374 @ MSG_ARG_ZFATWNOFATW      "-z[fatal-warnings|nofatalwarnings]"

1376 @ MSG_ARG_ABSEXEC          "absexec"
1377 @ MSG_ARG_ALTEXEC64        "altexec64"
1378 @ MSG_ARG_ASLR             "aslr"

```

```

1379 #endif /* ! codereview */
1380 @ MSG_ARG_NOCOMPSTRTAB     "nocompstrtab"
1381 @ MSG_ARG_GROUPEM         "grouperem"
1382 @ MSG_ARG_NOGROUPEM       "nogrouperem"
1383 @ MSG_ARG_LAZYLOAD         "lazyload"
1384 @ MSG_ARG_NOLAZYLOAD       "nolazyload"
1385 @ MSG_ARG_INTERPOSE        "interpose"
1386 @ MSG_ARG_DIRECT          "direct"
1387 @ MSG_ARG_NODIRECT         "nodirect"
1388 @ MSG_ARG_IGNORE           "ignore"
1389 @ MSG_ARG_RECORD           "record"
1390 @ MSG_ARG_INITFIRST        "initfirst"
1391 @ MSG_ARG_INITARRAY        "initarray="
1392 @ MSG_ARG_FINIARRAY        "finiarray="
1393 @ MSG_ARG_PREINITARRAY     "preinitarray="
1394 @ MSG_ARG_RTLDINFO         "rtldinfo="
1395 @ MSG_ARG_DTRACE           "dtrace="
1396 @ MSG_ARG_TRANSLATOR       "translator"
1397 @ MSG_ARG_NOOPEN           "nodlopen"
1398 @ MSG_ARG_NOW              "now"
1399 @ MSG_ARG_ORIGIN           "origin"
1400 @ MSG_ARG_DEFS              "defs"
1401 @ MSG_ARG_NODEFS           "nodefs"
1402 @ MSG_ARG_NODUMP           "nodump"
1403 @ MSG_ARG_NOVERSION        "noversion"
1404 @ MSG_ARG_TEXT              "text"
1405 @ MSG_ARG_TEXTOFF          "textoff"
1406 @ MSG_ARG_TEXTWARN         "textwarn"
1407 @ MSG_ARG_MULDEFS          "muldefs"
1408 @ MSG_ARG_NODELETE         "nodelete"
1409 @ MSG_ARG_NOINTERP         "nointerp"
1410 @ MSG_ARG_NOPARTIAL        "nopartial"
1411 @ MSG_ARG_NORELOC          "noreloc"
1412 @ MSG_ARG_REDLOCYSYM       "redlocsym"
1413 @ MSG_ARG_VERBOSE          "verbose"
1414 @ MSG_ARG_WEAKEXT          "weakextract"
1415 @ MSG_ARG_LOADFLTR         "loadfltr"
1416 @ MSG_ARG_ALLEXTRT         "allextract"
1417 @ MSG_ARG_DFLEXTRT         "defaultextract"
1418 @ MSG_ARG_COMBRELOC        "combreloc"
1419 @ MSG_ARG_NOCOMBRELOC      "nocombreloc"
1420 @ MSG_ARG_NODEFAULTLIB     "nodefaultlib"
1421 @ MSG_ARG_ENDFILTEE        "endfiltee"
1422 @ MSG_ARG_LD32             "ld32="
1423 @ MSG_ARG_LD64             "ld64="
1424 @ MSG_ARG_RESCAN           "rescan"
1425 @ MSG_ARG_RESCAN_NOW       "rescan-now"
1426 @ MSG_ARG_RESCAN_START     "rescan-start"
1427 @ MSG_ARG_RESCAN_END       "rescan-end"
1428 @ MSG_ARG_GUIDE            "guidance"
1429 @ MSG_ARG_NOLDYNSYM        "noldynsym"
1430 @ MSG_ARG_RELAXRELOC       "relaxreloc"
1431 @ MSG_ARG_NORELAXRELOC     "norelaxreloc"
1432 @ MSG_ARG_NOSIGHANDLER     "nosighandler"
1433 @ MSG_ARG_GLOBAUDIT        "globalaudit"
1434 @ MSG_ARG_TARGET           "target="
1435 @ MSG_ARG_WRAP              "wrap="
1436 @ MSG_ARG_FATWARN          "fatal-warnings"
1437 @ MSG_ARG_NOFATWARN        "nofatal-warnings"
1438 @ MSG_ARG_HELP             "help"
1439 @ MSG_ARG_GROUP            "group"
1440 @ MSG_ARG_REDUCE           "reduce"
1441 @ MSG_ARG_STATIC           "static"
1442 @ MSG_ARG_SYMBOLCAP        "symbolcap"
1443 @ MSG_ARG_DEFERRED         "deferred"
1444 @ MSG_ARG_NODEFERRED       "nodeferred"

```

```

1445 @ MSG_ARG_ASSDEFLIB      "assert-deflib"

1447 @ MSG_ARG_LCOM           "L,"
1448 @ MSG_ARG_PCOM           "P,"
1449 @ MSG_ARG_UCOM            "U,"

1451 @ MSG_ARG_T_RPATH        "rpath"
1452 @ MSG_ARG_T_SHARED        "shared"
1453 @ MSG_ARG_T_SONAME        "soname"
1454 @ MSG_ARG_T_WL            "l,-"

1456 @ MSG_ARG_T_AUXFLTR      "--auxiliary"
1457 @ MSG_ARG_T_MULDEFS      "-allow-multiple-definition"
1458 @ MSG_ARG_T_INTERP        "-dynamic-linker"
1459 @ MSG_ARG_T_ENDGROUP      "-end-group"
1460 @ MSG_ARG_T_ENTRY         "-entry"
1461 @ MSG_ARG_T_STDFLTR       "-filter"
1462 @ MSG_ARG_T_FATWARN       "-fatal-warnings"
1463 @ MSG_ARG_T_NOFATWARN     "-no-fatal-warnings"
1464 @ MSG_ARG_T_HELP          "-help"
1465 @ MSG_ARG_T_LIBRARY        "-library"
1466 @ MSG_ARG_T_LIBPATH        "-library-path"
1467 @ MSG_ARG_T_NOUNDEF       "-no-undefined"
1468 @ MSG_ARG_T_NOWHOLEARC    "-no-whole-archive"
1469 @ MSG_ARG_T_OUTPUT         "-output"
1470 @ MSG_ARG_T_RELOCATABLE   "--relocatable"
1471 @ MSG_ARG_T_STARTGROUP    "-start-group"
1472 @ MSG_ARG_T_STRIP         "-strip-all"
1473 @ MSG_ARG_T_UNDEF         "-undefined"
1474 @ MSG_ARG_T_VERSION        "-version"
1475 @ MSG_ARG_T_WHOLEARC       "-whole-archive"
1476 @ MSG_ARG_T_WRAP           "-wrap"
1477 @ MSG_ARG_T_OPAR          "("
1478 @ MSG_ARG_T_CPAR           ")"

1480 @ MSG_ARG_ENABLED         "enabled"
1481 @ MSG_ARG_DISABLED        "disabled"

1483 #endif /* ! codereview */
1484 # -z guidance=item strings
1485 @ MSG_ARG_GUIDE_DELIM      ",: \t"
1486 @ MSG_ARG_GUIDE_NO_ALL     "noall"
1487 @ MSG_ARG_GUIDE_NO_DEFS    "nodefs"
1488 @ MSG_ARG_GUIDE_NO_DIRECT  "nodirect"
1489 @ MSG_ARG_GUIDE_NO_LAZYLOAD "nolazyload"
1490 @ MSG_ARG_GUIDE_NO_MAPFILE "nomapfile"
1491 @ MSG_ARG_GUIDE_NO_TEXT    "notext"
1492 @ MSG_ARG_GUIDE_NO_UNUSED  "nounused"

1494 # Environment variable strings

1496 @ MSG_LD_RUN_PATH          "LD_RUN_PATH"
1497 @ MSG_LD_LIBPATH_32        "LD_LIBRARY_PATH_32"
1498 @ MSG_LD_LIBPATH_64        "LD_LIBRARY_PATH_64"
1499 @ MSG_LD_LIBPATH           "LD_LIBRARY_PATH"

1501 @ MSG_LD_NOVERSION_32      "LD_NOVERSION_32"
1502 @ MSG_LD_NOVERSION_64      "LD_NOVERSION_64"
1503 @ MSG_LD_NOVERSION         "LD_NOVERSION"

1505 @ MSG_SGS_SUPPORT_32       "SGS_SUPPORT_32"
1506 @ MSG_SGS_SUPPORT_64       "SGS_SUPPORT_64"
1507 @ MSG_SGS_SUPPORT          "SGS_SUPPORT"

1510 # Symbol names

```

```

1512 @ MSG_SYM_LIBVER_U       "_lib_version"

1515 # Mapfile tokens

1517 @ MSG_MAP_LOAD             "load"
1518 @ MSG_MAP_NOTE             "note"
1519 @ MSG_MAP_NULL             "null"
1520 @ MSG_MAP_STACK            "stack"
1521 @ MSG_MAP_ADDVERS          "addvers"
1522 @ MSG_MAP_FUNCTION         "function"
1523 @ MSG_MAP_DATA             "data"
1524 @ MSG_MAP_COMMON           "common"
1525 @ MSG_MAP_PARENT           "parent"
1526 @ MSG_MAP_EXTERN           "extern"
1527 @ MSG_MAP_DIRECT           "direct"
1528 @ MSG_MAP_NODIRECT         "nodirect"
1529 @ MSG_MAP_FILTER           "filter"
1530 @ MSG_MAP_AUXILIARY        "auxiliary"
1531 @ MSG_MAP_OVERRIDE          "override"
1532 @ MSG_MAP_INTERPOSE        "interpose"
1533 @ MSG_MAP_DYNSORT          "dynsort"
1534 @ MSG_MAP_NODYNSORT        "nodynsort"

1536 @ MSG_MAPKW_ALIGN          "ALIGN"
1537 @ MSG_MAPKW_ALLOC          "ALLOC"
1538 @ MSG_MAPKW_ALLOW          "ALLOW"
1539 @ MSG_MAPKW_AMD64_LARGE    "AMD64_LARGE"
1540 @ MSG_MAPKW_ASSIGN_SECTION "ASSIGN_SECTION"
1541 @ MSG_MAPKW_AUX            "AUXILIARY"
1542 @ MSG_MAPKW_CAPABILITY     "CAPABILITY"
1543 @ MSG_MAPKW_COMMON         "COMMON"
1544 @ MSG_MAPKW_DATA           "DATA"
1545 @ MSG_MAPKW_DEFAULT        "DEFAULT"
1546 @ MSG_MAPKW_DEPEND_VERSIONS "DEPEND_VERSIONS"
1547 @ MSG_MAPKW_DIRECT         "DIRECT"
1548 @ MSG_MAPKW_DISABLE        "DISABLE"
1549 @ MSG_MAPKW_DYNSORT        "DYNSORT"
1550 @ MSG_MAPKW_ELIMINATE      "ELIMINATE"
1551 @ MSG_MAPKW_EXECUTE        "EXECUTE"
1552 @ MSG_MAPKW_EXPORTED       "EXPORTED"
1553 @ MSG_MAPKW_EXTERN         "EXTERN"
1554 @ MSG_MAPKW_FILTER         "FILTER"
1555 @ MSG_MAPKW_FILE_BASENAME   "FILE_BASENAME"
1556 @ MSG_MAPKW_FILE_PATH      "FILE_PATH"
1557 @ MSG_MAPKW_FILE_OBJNAME    "FILE_OBJNAME"
1558 @ MSG_MAPKW_FUNCTION        "FUNCTION"
1559 @ MSG_MAPKW_FLAGS          "FLAGS"
1560 @ MSG_MAPKW_GLOBAL          "GLOBAL"
1561 @ MSG_MAPKW_INTERPOSE       "INTERPOSE"
1562 @ MSG_MAPKW_HIDDEN          "HIDDEN"
1563 @ MSG_MAPKW_HDR_NOALLOC     "HDR_NOALLOC"
1564 @ MSG_MAPKW_HW              "HW"
1565 @ MSG_MAPKW_HW_1            "HW_1"
1566 @ MSG_MAPKW_HW_2            "HW_2"
1567 @ MSG_MAPKW_IS_NAME         "IS_NAME"
1568 @ MSG_MAPKW_IS_ORDER        "IS_ORDER"
1569 @ MSG_MAPKW_LOAD_SEGMENT    "LOAD_SEGMENT"
1570 @ MSG_MAPKW_LOCAL           "LOCAL"
1571 @ MSG_MAPKW_MACHINE         "MACHINE"
1572 @ MSG_MAPKW_MAX_SIZE        "MAX_SIZE"
1573 @ MSG_MAPKW_NOHDR           "NOHDR"
1574 @ MSG_MAPKW_NODIRECT        "NODIRECT"
1575 @ MSG_MAPKW_NODYNSORT       "NODYNSORT"
1576 @ MSG_MAPKW_NOTE_SEGMENT    "NOTE_SEGMENT"

```



```
1577 @ MSG_MAPKW_NULL_SEGMENT      "NULL_SEGMENT"  
1578 @ MSG_MAPKW_OS_ORDER          "OS_ORDER"  
1579 @ MSG_MAPKW_PADDR             "PADDR"  
1580 @ MSG_MAPKW_PARENT            "PARENT"  
1581 @ MSG_MAPKW_PHDR_ADD_NULL     "PHDR_ADD_NULL"  
1582 @ MSG_MAPKW_PLATFORM          "PLATFORM"  
1583 @ MSG_MAPKW_PROTECTED         "PROTECTED"  
1584 @ MSG_MAPKW_READ              "READ"  
1585 @ MSG_MAPKW_ROUND             "ROUND"  
1586 @ MSG_MAPKW_REQUIRE           "REQUIRE"  
1587 @ MSG_MAPKW_SEGMENT_ORDER    "SEGMENT_ORDER"  
1588 @ MSG_MAPKW_SF                "SF"  
1589 @ MSG_MAPKW_SF_1              "SF_1"  
1590 @ MSG_MAPKW_SINGLETON         "SINGLETON"  
1591 @ MSG_MAPKW_SIZE              "SIZE"  
1592 @ MSG_MAPKW_SIZE_SYMBOL      "SIZE_SYMBOL"  
1593 @ MSG_MAPKW_STACK             "STACK"  
1594 @ MSG_MAPKW_SYMBOL_SCOPE      "SYMBOL_SCOPE"  
1595 @ MSG_MAPKW_SYMBOL_VERSION    "SYMBOL_VERSION"  
1596 @ MSG_MAPKW_SYMBOLIC         "SYMBOLIC"  
1597 @ MSG_MAPKW_TYPE              "TYPE"  
1598 @ MSG_MAPKW_VADDR             "VADDR"  
1599 @ MSG_MAPKW_VALUE            "VALUE"  
1600 @ MSG_MAPKW_WRITE            "WRITE"  
  
1603 @ MSG_STR_DTRACE             "PT_SUNWDTRACE"
```

```

*****
96450 Thu Jun 30 22:01:33 2016
new/usr/src/cmd/sgs/libld/common/sections.c
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
_____unchanged_portion_omitted_____

927 /*
928 * Make the dynamic section. Calculate the size of any strings referenced
929 * within this structure, they will be added to the global string table
930 * (.dynstr). This routine should be called before make_dynstr().
931 *
932 * This routine must be maintained in parallel with update_odynamic()
933 * in update.c
934 */
935 static uintptr_t
936 make_dynamic(Of1_desc *of1)
937 {
938     Shdr      *shdr;
939     Os_desc   *osp;
940     Elf_Data  *data;
941     Is_desc   *isec;
942     size_t    cnt = 0;
943     Aliste    idx;
944     Ifl_desc  *ifl;
945     Sym_desc  *sdp;
946     size_t    size;
947     Str_ttbl  *strtbl;
948     ofl_flag_t flags = ofl->ofl_flags;
949     int       not_relobj = !(flags & FLG_OF_RELOBJ);
950     int       unused = 0;

952     /*
953      * Select the required string table.
954      */
955     if (OFL_IS_STATIC_OBJ(of1))
956         strtbl = ofl->ofl_strtab;
957     else
958         strtbl = ofl->ofl_dynstrtab;

960     /*
961      * Only a limited subset of DT_entries apply to relocatable
962      * objects. See the comment at the head of update_odynamic() in
963      * update.c for details.
964      */
965     if (new_section(of1, SHT_DYNAMIC, MSG_ORIG(MSG_SCN_DYNAMIC), 0,
966                   &isec, &shdr, &data) == S_ERROR)
967         return (S_ERROR);

969     /*
970      * new_section() does not set SHF_ALLOC. If we're building anything
971      * besides a relocatable object, then the .dynamic section should
972      * reside in allocatable memory.
973      */
974     if (not_relobj)
975         shdr->sh_flags |= SHF_ALLOC;

977     /*
978      * new_section() does not set SHF_WRITE. If we're building an object
979      * that specifies an interpreter, then a DT_DEBUG entry is created,
980      * which is initialized to the applications link-map list at runtime.
981      */
982     if (ofl->ofl_osinterp)

```

```

983         shdr->sh_flags |= SHF_WRITE;

985     osp = ofl->ofl_osdynamic =
986         ld_place_section(ofl, isec, NULL, ld_targ.t_id.id_dynamic, NULL);

988     /*
989      * Reserve entries for any needed dependencies.
990      */
991     for (APLIST_TRAVERSE(ofl->ofl_sos, idx, ifl)) {
992         if (!(ifl->ifl_flags & (FLG_IF_NEEDED | FLG_IF_NEEDSTR)))
993             continue;

995         /*
996          * If this dependency didn't satisfy any symbol references,
997          * generate a debugging diagnostic (ld(1) -Dunused can be used
998          * to display these). If this is a standard needed dependency,
999          * and -z ignore is in effect, drop the dependency. Explicitly
1000          * defined dependencies (i.e., -N dep) don't get dropped, and
1001          * are flagged as being required to simplify update_odynamic()
1002          * processing.
1003          */
1004         if ((ifl->ifl_flags & FLG_IF_NEEDSTR) ||
1005             ((ifl->ifl_flags & FLG_IF_DEPREQD) == 0)) {
1006             if (unused++ == 0)
1007                 DBG_CALL(DBG_util_nl(ofl->ofl_lml, DBG_NL_STD));
1008             DBG_CALL(DBG_unused_file(ofl->ofl_lml, ifl->ifl_soname,
1009                                   (ifl->ifl_flags & FLG_IF_NEEDSTR), 0));

1011         /*
1012          * Guidance: Remove unused dependency.
1013          *
1014          * If -z ignore is in effect, this warning is not
1015          * needed because we will quietly remove the unused
1016          * dependency.
1017          */
1018         if (OFL_GUIDANCE(ofl, FLG_OFG_NO_UNUSED) &&
1019             ((ifl->ifl_flags & FLG_IF_IGNORE) == 0))
1020             ld_eprintf(ofl, ERR_GUIDANCE,
1021                       MSG_INTL(MSG_GUIDE_UNUSED),
1022                       ifl->ifl_soname);

1024         if (ifl->ifl_flags & FLG_IF_NEEDSTR)
1025             ifl->ifl_flags |= FLG_IF_DEPREQD;
1026         else if (ifl->ifl_flags & FLG_IF_IGNORE)
1027             continue;
1028     }

1030     /*
1031      * If this object requires a DT_POSFLAG_1 entry, reserve it.
1032      */
1033     if ((ifl->ifl_flags & MSK_IF_POSFLAG1) && not_relobj)
1034         cnt++;

1036     if (st_insert(strtbl, ifl->ifl_soname) == -1)
1037         return (S_ERROR);
1038     cnt++;

1040     /*
1041      * If the needed entry contains the $ORIGIN token make sure
1042      * the associated DT_1_FLAGS entry is created.
1043      */
1044     if (strstr(ifl->ifl_soname, MSG_ORIG(MSG_STR_ORIGIN))) {
1045         ofl->ofl_dtflags_1 |= DF_1_ORIGIN;
1046         ofl->ofl_dtflags |= DF_ORIGIN;
1047     }
1048 }

```

```

1050     if (unused)
1051         DBG_CALL(DBG_util_nl(ofl->ofl_lml, DBG_NL_STD));

1053     if (not_relobj) {
1054         /*
1055          * Reserve entries for any per-symbol auxiliary/filter strings.
1056          */
1057         cnt += alist_nitems(ofl->ofl_dtsfltrs);

1059         /*
1060          * Reserve entries for _init() and _fini() section addresses.
1061          */
1062         if (((sdp = ld_sym_find(MSG_ORIG(MSG_SYM_INIT_U),
1063             SYM_NOHASH, NULL, ofl)) != NULL) &&
1064             (sdp->sd_ref == REF_REL_NEED) &&
1065             (sdp->sd_sym->st_shndx != SHN_UNDEF)) {
1066             sdp->sd_flags |= FLG_SY_UPREQD;
1067             cnt++;
1068         }
1069         if (((sdp = ld_sym_find(MSG_ORIG(MSG_SYM_FINI_U),
1070             SYM_NOHASH, NULL, ofl)) != NULL) &&
1071             (sdp->sd_ref == REF_REL_NEED) &&
1072             (sdp->sd_sym->st_shndx != SHN_UNDEF)) {
1073             sdp->sd_flags |= FLG_SY_UPREQD;
1074             cnt++;
1075         }

1077         /*
1078          * Reserve entries for any soname, filter name (shared libs
1079          * only), run-path pointers, cache names and audit requirements.
1080          */
1081         if (ofl->ofl_soname) {
1082             cnt++;
1083             if (st_insert(strtbl, ofl->ofl_soname) == -1)
1084                 return (S_ERROR);
1085         }
1086         if (ofl->ofl_filtees) {
1087             cnt++;
1088             if (st_insert(strtbl, ofl->ofl_filtees) == -1)
1089                 return (S_ERROR);

1091             /*
1092              * If the filtees entry contains the $ORIGIN token
1093              * make sure the associated DT_1_FLAGS entry is created.
1094              */
1095             if (strstr(ofl->ofl_filtees,
1096                 MSG_ORIG(MSG_STR_ORIGIN)) {
1097                 ofl->ofl_dtflags_1 |= DF_1_ORIGIN;
1098                 ofl->ofl_dtflags |= DF_ORIGIN;
1099             }
1100         }
1101     }

1103     if (ofl->ofl_rpath) {
1104         cnt += 2;          /* DT_RPATH & DT_RUNPATH */
1105         if (st_insert(strtbl, ofl->ofl_rpath) == -1)
1106             return (S_ERROR);

1108         /*
1109          * If the rpath entry contains the $ORIGIN token make sure
1110          * the associated DT_1_FLAGS entry is created.
1111          */
1112         if (strstr(ofl->ofl_rpath, MSG_ORIG(MSG_STR_ORIGIN)) {
1113             ofl->ofl_dtflags_1 |= DF_1_ORIGIN;
1114             ofl->ofl_dtflags |= DF_ORIGIN;

```

```

1115     }
1116 }

1118     if (not_relobj) {
1119         Aliste idx;
1120         Sg_desc *sgp;

1122         if (ofl->ofl_config) {
1123             cnt++;
1124             if (st_insert(strtbl, ofl->ofl_config) == -1)
1125                 return (S_ERROR);

1127             /*
1128              * If the config entry contains the $ORIGIN token
1129              * make sure the associated DT_1_FLAGS entry is created.
1130              */
1131             if (strstr(ofl->ofl_config, MSG_ORIG(MSG_STR_ORIGIN)) {
1132                 ofl->ofl_dtflags_1 |= DF_1_ORIGIN;
1133                 ofl->ofl_dtflags |= DF_ORIGIN;
1134             }
1135         }
1136         if (ofl->ofl_depaudit) {
1137             cnt++;
1138             if (st_insert(strtbl, ofl->ofl_depaudit) == -1)
1139                 return (S_ERROR);
1140         }
1141         if (ofl->ofl_audit) {
1142             cnt++;
1143             if (st_insert(strtbl, ofl->ofl_audit) == -1)
1144                 return (S_ERROR);
1145         }

1147         /*
1148          * Reserve entries for the DT_HASH, DT_STRTAB, DT_STRSZ,
1149          * DT_SYMTAB, DT_SYMENT, and DT_CHECKSUM.
1150          */
1151         cnt += 6;

1153         /*
1154          * If we are including local functions at the head of
1155          * the dynsym, then also reserve entries for DT_SUNW_SYMTAB
1156          * and DT_SUNW_SYMSZ.
1157          */
1158         if (OFL_ALLOW_LDYNSYM(ofl))
1159             cnt += 2;

1161         if ((ofl->ofl_dynsymstcnt > 0) ||
1162             (ofl->ofl_dyntlssortcnt > 0))
1163             cnt++;          /* DT_SUNW_SORTENT */

1165         if (ofl->ofl_dynsymstcnt > 0)
1166             cnt += 2;      /* DT_SUNW_[SYMSORT|SYMSORTSZ] */

1168         if (ofl->ofl_dyntlssortcnt > 0)
1169             cnt += 2;      /* DT_SUNW_[TLSSORT|TLSSORTSZ] */

1171         if ((flags & (FLG_OF_VERDEF | FLG_OF_NOVERSEC)) ==
1172             FLG_OF_VERDEF)
1173             cnt += 2;      /* DT_VERDEF & DT_VERDEFNUM */

1175         if ((flags & (FLG_OF_VERNEED | FLG_OF_NOVERSEC)) ==
1176             FLG_OF_VERNEED)
1177             cnt += 2;      /* DT_VERNEED & DT_VERNEEDNUM */

1179         if ((flags & FLG_OF_COMREL) && ofl->ofl_relocrelcnt)
1180             cnt++;        /* DT_RELACOUNT */

```

```

1182         if (flags & FLG_OF_TEXTREL)      /* DT_TEXTREL */
1183             cnt++;
1185         if (ofl->ofl_osfiniarray)         /* DT_FINI_ARRAY */
1186             cnt += 2;                     /* DT_FINI_ARRAYSZ */
1188         if (ofl->ofl_osinitarray)         /* DT_INIT_ARRAY */
1189             cnt += 2;                     /* DT_INIT_ARRAYSZ */
1191         if (ofl->ofl_ospreinitarray)      /* DT_PREINIT_ARRAY & */
1192             cnt += 2;                     /* DT_PREINIT_ARRAYSZ */
1194         /*
1195          * If we have plt's reserve a DT_PLTRELSZ, DT_PLTREL and
1196          * DT_JMPREL.
1197          */
1198         if (ofl->ofl_pltcnt)
1199             cnt += 3;
1201         /*
1202          * If plt padding is needed (Sparcv9).
1203          */
1204         if (ofl->ofl_pltpad)
1205             cnt += 2;                     /* DT_PLTPAD & DT_PLTPADSZ */
1207         /*
1208          * If we have any relocations reserve a DT_REL, DT_RELSZ and
1209          * DT_RELENT entry.
1210          */
1211         if (ofl->ofl_relocsz)
1212             cnt += 3;
1214         /*
1215          * If a syminfo section is required create DT_SYMINFO,
1216          * DT_SYMINSZ, and DT_SYMINENT entries.
1217          */
1218         if (flags & FLG_OF_SYMINFO)
1219             cnt += 3;
1221         /*
1222          * If there are any partially initialized sections allocate
1223          * DT_MOVETAB, DT_MOVESZ and DT_MOVEENT.
1224          */
1225         if (ofl->ofl_osmove)
1226             cnt += 3;
1228         /*
1229          * Allocate one DT_REGISTER entry for every register symbol.
1230          */
1231         cnt += ofl->ofl_regsymcnt;
1233         /*
1234          * Reserve a entry for each '--zrtldinfo=...' specified
1235          * on the command line.
1236          */
1237         for (APLIST_TRAVERSE(ofl->ofl_rtlldinfo, idx, sdp))
1238             cnt++;
1240         /*
1241          * The following entry should only be placed in a segment that
1242          * is writable.
1243          */
1244         if (((sgp = osp->os_sgdesc) != NULL) &&
1245             (sgp->sg_phdr.p_flags & PF_W) && ofl->ofl_osinterp)
1246             cnt++;                       /* DT_DEBUG */

```

```

1248         /*
1249          * Capabilities require a .dynamic entry for the .SUNW_cap
1250          * section.
1251          */
1252         if (ofl->ofl_oscapi)
1253             cnt++;                       /* DT_SUNW_CAP */
1255         /*
1256          * Symbol capabilities require a .dynamic entry for the
1257          * .SUNW_capinfo section.
1258          */
1259         if (ofl->ofl_oscapiinfo)
1260             cnt++;                       /* DT_SUNW_CAPINFO */
1262         /*
1263          * Capabilities chain information requires a .SUNW_capchain
1264          * entry (DT_SUNW_CAPCHAIN), entry size (DT_SUNW_CAPCHAINENT),
1265          * and total size (DT_SUNW_CAPCHAINSZ).
1266          */
1267         if (ofl->ofl_oscapichain)
1268             cnt += 3;
1270         if (flags & FLG_OF_SYMBOLIC)
1271             cnt++;                       /* DT_SYMBOLIC */
1273         if (ofl->ofl_aslr != 0)           /* DT_SUNW_ASLR */
1274             cnt++;
1275     #endif /* ! codereview */
1276     }
1278     /*
1279      * Account for Architecture dependent .dynamic entries, and defaults.
1280      */
1281     (*ld_targ.t_mr.mr_mach_make_dynamic)(ofl, &cnt);
1283     /*
1284      * DT_FLAGS, DT_FLAGS_1, DT_SUNW_STRPAD, and DT_NULL. Also,
1285      * allow room for the unused extra DT_NULLs. These are included
1286      * to allow an ELF editor room to add items later.
1287      */
1288     cnt += 4 + DYNAMIC_EXTRAELTS;
1290     /*
1291      * DT_SUNW_LDMACH. Used to hold the ELF machine code of the
1292      * linker that produced the output object. This information
1293      * allows us to determine whether a given object was linked
1294      * natively, or by a linker running on a different type of
1295      * system. This information can be valuable if one suspects
1296      * that a problem might be due to alignment or byte order issues.
1297      */
1298     cnt++;
1300     /*
1301      * Determine the size of the section from the number of entries.
1302      */
1303     size = cnt * (size_t)shdr->sh_entsize;
1305     shdr->sh_size = (Xword)size;
1306     data->d_size = size;
1308     /*
1309      * There are several tags that are specific to the Solaris osabi
1310      * range which we unconditionally put into any dynamic section
1311      * we create (e.g. DT_SUNW_STRPAD or DT_SUNW_LDMACH). As such,
1312      * any Solaris object with a dynamic section should be tagged as

```

```

1313  * ELFOSABI_SOLARIS.
1314  */
1315  ofl->ofl_flags |= FLG_OF_OSABI;

1317  return ((uintptr_t)ofl->ofl_osdynamic);
1318 }

1320 /*
1321  * Build the GOT section and its associated relocation entries.
1322  */
1323 uintptr_t
1324 ld_make_got(Of1_desc *ofl)
1325 {
1326     Elf_Data      *data;
1327     Shdr          *shdr;
1328     Is_desc       *isec;
1329     size_t        size = (size_t)ofl->ofl_gotcnt * ld_targ.t_m.m_got_entsize;
1330     size_t        rsize = (size_t)ofl->ofl_relocgotsz;

1332     if (new_section(ofl, SHT_PROGBITS, MSG_ORIG(MSG_SCN_GOT), 0,
1333                 &isec, &shdr, &data) == S_ERROR)
1334         return (S_ERROR);

1336     data->d_size = size;

1338     shdr->sh_flags |= SHF_WRITE;
1339     shdr->sh_size = (Xword)size;
1340     shdr->sh_entsize = ld_targ.t_m.m_got_entsize;

1342     ofl->ofl_osgot = ld_place_section(ofl, isec, NULL,
1343                 ld_targ.t_id.id_got, NULL);
1344     if (ofl->ofl_osgot == (Os_desc *)S_ERROR)
1345         return (S_ERROR);

1347     ofl->ofl_osgot->os_szoutrels = (Xword)rsize;

1349     return (1);
1350 }

1352 /*
1353  * Build an interpreter section.
1354  */
1355 static uintptr_t
1356 make_interp(Of1_desc *ofl)
1357 {
1358     Shdr          *shdr;
1359     Elf_Data      *data;
1360     Is_desc       *isec;
1361     const char    *iname = ofl->ofl_interp;
1362     size_t        size;

1364     /*
1365     * If -z nointerp is in effect, don't create an interpreter section.
1366     */
1367     if (ofl->ofl_flags1 & FLG_OF1_NOINTRP)
1368         return (1);

1370     /*
1371     * An .interp section is always created for a dynamic executable.
1372     * A user can define the interpreter to use. This definition overrides
1373     * the default that would be recorded in an executable, and triggers
1374     * the creation of an .interp section in any other object. Presumably
1375     * the user knows what they are doing. Refer to the generic ELF ABI
1376     * section 5-4, and the ld(1) -I option.
1377     */
1378     if (((ofl->ofl_flags & (FLG_OF_DYNAMIC | FLG_OF_EXEC |

```

```

1379         FLG_OF_RELOBJ)) != (FLG_OF_DYNAMIC | FLG_OF_EXEC)) && !iname)
1380         return (1);

1382     /*
1383     * In the case of a dynamic executable, supply a default interpreter
1384     * if the user has not specified their own.
1385     */
1386     if (iname == NULL)
1387         iname = ofl->ofl_interp = ld_targ.t_m.m_def_interp;

1389     size = strlen(iname) + 1;

1391     if (new_section(ofl, SHT_PROGBITS, MSG_ORIG(MSG_SCN_INTERP), 0,
1392                 &isec, &shdr, &data) == S_ERROR)
1393         return (S_ERROR);

1395     data->d_size = size;
1396     shdr->sh_size = (Xword)size;
1397     data->d_align = shdr->sh_addralign = 1;

1399     ofl->ofl_osinterp =
1400         ld_place_section(ofl, isec, NULL, ld_targ.t_id.id_interp, NULL);
1401     return ((uintptr_t)ofl->ofl_osinterp);
1402 }

1404 /*
1405  * Common function used to build the SHT_SUNW_versym section, SHT_SUNW_syminfo
1406  * section, and SHT_SUNW_capinfo section. Each of these sections provide
1407  * additional symbol information, and their size parallels the associated
1408  * symbol table.
1409  */
1410 static Os_desc *
1411 make_sym_sec(Of1_desc *ofl, const char *sectname, Word stype, int ident)
1412 {
1413     Shdr          *shdr;
1414     Elf_Data      *data;
1415     Is_desc       *isec;

1417     /*
1418     * We don't know the size of this section yet, so set it to 0. The
1419     * size gets filled in after the associated symbol table is sized.
1420     */
1421     if (new_section(ofl, stype, sectname, 0, &isec, &shdr, &data) ==
1422         S_ERROR)
1423         return ((Os_desc *)S_ERROR);

1425     return (ld_place_section(ofl, isec, NULL, ident, NULL));
1426 }

1428 /*
1429  * Determine whether a symbol capability is redundant because the object
1430  * capabilities are more restrictive.
1431  */
1432 inline static int
1433 is_cap_redundant(Objcapset *ocapset, Objcapset *scapset)
1434 {
1435     Alist          *oalp, *salp;
1436     elfcap_mask_t  omsk, smsk;

1438     /*
1439     * Inspect any platform capabilities. If the object defines platform
1440     * capabilities, then the object will only be loaded for those
1441     * platforms. A symbol capability set that doesn't define the same
1442     * platforms is redundant, and a symbol capability that does not provide
1443     * at least one platform name that matches a platform name in the object
1444     * capabilities will never execute (as the object wouldn't have been

```

```

1445     * loaded).
1446     */
1447     oalp = ocapset->oc_plat.cl_val;
1448     salp = scapset->oc_plat.cl_val;
1449     if (oalp && ((salp == NULL) || cap_names_match(oalp, salp)))
1450         return (1);
1451
1452     /*
1453     * If the symbol capability set defines platforms, and the object
1454     * doesn't, then the symbol set is more restrictive.
1455     */
1456     if (salp && (oalp == NULL))
1457         return (0);
1458
1459     /*
1460     * Next, inspect any machine name capabilities. If the object defines
1461     * machine name capabilities, then the object will only be loaded for
1462     * those machines. A symbol capability set that doesn't define the same
1463     * machine names is redundant, and a symbol capability that does not
1464     * provide at least one machine name that matches a machine name in the
1465     * object capabilities will never execute (as the object wouldn't have
1466     * been loaded).
1467     */
1468     oalp = ocapset->oc_plat.cl_val;
1469     salp = scapset->oc_plat.cl_val;
1470     if (oalp && ((salp == NULL) || cap_names_match(oalp, salp)))
1471         return (1);
1472
1473     /*
1474     * If the symbol capability set defines machine names, and the object
1475     * doesn't, then the symbol set is more restrictive.
1476     */
1477     if (salp && (oalp == NULL))
1478         return (0);
1479
1480     /*
1481     * Next, inspect any hardware capabilities. If the objects hardware
1482     * capabilities are greater than or equal to that of the symbols
1483     * capabilities, then the symbol capability set is redundant. If the
1484     * symbols hardware capabilities are greater than the objects, then the
1485     * symbol set is more restrictive.
1486     *
1487     * Note that this is a somewhat arbitrary definition, as each capability
1488     * bit is independent of the others, and some of the higher order bits
1489     * could be considered to be less important than lower ones. However,
1490     * this is the only reasonable non-subjective definition.
1491     */
1492     omsk = ocapset->oc_hw_2.cm_val;
1493     smsk = scapset->oc_hw_2.cm_val;
1494     if ((omsk > smsk) || (omsk && (omsk == smsk)))
1495         return (1);
1496     if (omsk < smsk)
1497         return (0);
1498
1499     /*
1500     * Finally, inspect the remaining hardware capabilities.
1501     */
1502     omsk = ocapset->oc_hw_1.cm_val;
1503     smsk = scapset->oc_hw_1.cm_val;
1504     if ((omsk > smsk) || (omsk && (omsk == smsk)))
1505         return (1);
1506
1507     return (0);
1508 }
1509
1510 /*

```

```

1511     * Capabilities values might have been assigned excluded values. These
1512     * excluded values should be removed before calculating any capabilities
1513     * sections size.
1514     */
1515     static void
1516     capmask_value(Lm_list *lml, Word type, Capmask *capmask, int *title)
1517     {
1518         /*
1519         * First determine whether any bits should be excluded.
1520         */
1521         if ((capmask->cm_val & capmask->cm_exc) == 0)
1522             return;
1523
1524         DBG_CALL(DBG_cap_post_title(lml, title));
1525
1526         DBG_CALL(DBG_cap_val_entry(lml, DBG_STATE_CURRENT, type,
1527             capmask->cm_val, ld_targ.t_m.m_mach));
1528         DBG_CALL(DBG_cap_val_entry(lml, DBG_STATE_EXCLUDE, type,
1529             capmask->cm_exc, ld_targ.t_m.m_mach));
1530
1531         capmask->cm_val &= ~capmask->cm_exc;
1532
1533         DBG_CALL(DBG_cap_val_entry(lml, DBG_STATE_RESOLVED, type,
1534             capmask->cm_val, ld_targ.t_m.m_mach));
1535     }
1536
1537     static void
1538     capstr_value(Lm_list *lml, Word type, Caplist *caplist, int *title)
1539     {
1540         Aliste idx1, idx2;
1541         char *estr;
1542         Capstr *capstr;
1543         Boolean found = FALSE;
1544
1545         /*
1546         * First determine whether any strings should be excluded.
1547         */
1548         for (APLIST_TRAVERSE(caplist->cl_exc, idx1, estr)) {
1549             for (ALIST_TRAVERSE(caplist->cl_val, idx2, capstr)) {
1550                 if (strcmp(estr, capstr->cs_str) == 0) {
1551                     found = TRUE;
1552                     break;
1553                 }
1554             }
1555         }
1556
1557         if (found == FALSE)
1558             return;
1559
1560         /*
1561         * Traverse the current strings, then delete the excluded strings,
1562         * and finally display the resolved strings.
1563         */
1564         if (DBG_ENABLED) {
1565             Dbg_cap_post_title(lml, title);
1566             for (ALIST_TRAVERSE(caplist->cl_val, idx2, capstr)) {
1567                 Dbg_cap_ptr_entry(lml, DBG_STATE_CURRENT, type,
1568                     capstr->cs_str);
1569             }
1570         }
1571         for (APLIST_TRAVERSE(caplist->cl_exc, idx1, estr)) {
1572             for (ALIST_TRAVERSE(caplist->cl_val, idx2, capstr)) {
1573                 if (strcmp(estr, capstr->cs_str) == 0) {
1574                     DBG_CALL(DBG_cap_ptr_entry(lml,
1575                         DBG_STATE_EXCLUDE, type, capstr->cs_str));
1576                     alist_delete(caplist->cl_val, &idx2);

```

```

1577         break;
1578     }
1579 }
1580 }
1581 if (DBG_ENABLED) {
1582     for (ALIST_TRAVERSE(caplist->cl_val, idx2, capstr)) {
1583         Dbg_cap_ptr_entry(lml, DBG_STATE_RESOLVED, type,
1584             capstr->cs_str);
1585     }
1586 }
1587 }

1589 /*
1590  * Build a capabilities section.
1591  */
1592 #define CAP_UPDATE(cap, capndx, tag, val) \
1593     cap->c_tag = tag; \
1594     cap->c_un.c_val = val; \
1595     cap++, capndx++;

1597 static uintptr_t
1598 make_cap(Of1_desc *of1, Word shtype, const char *shname, int ident)
1599 {
1600     Shdr      *shdr;
1601     Elf_Data  *data;
1602     Is_desc   *isec;
1603     Cap       *cap;
1604     size_t    size = 0;
1605     Word      capndx = 0;
1606     Str_tbl   *strtbl;
1607     Objcapset *ocapset = &of1->o1_ocapset;
1608     Aliste    idx1;
1609     Capstr    *capstr;
1610     int       title = 0;

1612     /*
1613      * Determine which string table to use for any CA_SUNW_MACH,
1614      * CA_SUNW_PLAT, or CA_SUNW_ID strings.
1615      */
1616     if (OFL_IS_STATIC_OBJ(of1))
1617         strtbl = of1->o1_strtab;
1618     else
1619         strtbl = of1->o1_dynstrtab;

1621     /*
1622      * If symbol capabilities have been requested, but none have been
1623      * created, warn the user. This scenario can occur if none of the
1624      * input relocatable objects defined any object capabilities.
1625      */
1626     if ((of1->o1_flags & FLG_OF_OTOSCAP) && (of1->o1_capsymcnt == 0))
1627         ld_printf(of1, ERR_WARNING, MSG_INTL(MSG_CAP_NOSYMSFOUND));

1629     /*
1630      * If symbol capabilities have been collected, but no symbols are left
1631      * referencing these capabilities, promote the capability groups back
1632      * to an object capability definition.
1633      */
1634     if ((of1->o1_flags & FLG_OF_OTOSCAP) && of1->o1_capsymcnt &&
1635         (of1->o1_capfamilies == NULL)) {
1636         ld_printf(of1, ERR_WARNING, MSG_INTL(MSG_CAP_NOSYMSFOUND));
1637         ld_cap_move_syntobj(of1);
1638         of1->o1_capsymcnt = 0;
1639         of1->o1_capgroups = NULL;
1640         of1->o1_flags &= ~FLG_OF_OTOSCAP;
1641     }

```

```

1643     /*
1644      * Remove any excluded capabilities.
1645      */
1646     capstr_value(of1->o1_lml, CA_SUNW_PLAT, &ocapset->oc_plat, &title);
1647     capstr_value(of1->o1_lml, CA_SUNW_MACH, &ocapset->oc_mach, &title);
1648     capmask_value(of1->o1_lml, CA_SUNW_HW_2, &ocapset->oc_hw_2, &title);
1649     capmask_value(of1->o1_lml, CA_SUNW_HW_1, &ocapset->oc_hw_1, &title);
1650     capmask_value(of1->o1_lml, CA_SUNW_SF_1, &ocapset->oc_sf_1, &title);

1652     /*
1653      * Determine how many entries are required for any object capabilities.
1654      */
1655     size += alist_nitems(ocapset->oc_plat.cl_val);
1656     size += alist_nitems(ocapset->oc_mach.cl_val);
1657     if (ocapset->oc_hw_2.cm_val)
1658         size++;
1659     if (ocapset->oc_hw_1.cm_val)
1660         size++;
1661     if (ocapset->oc_sf_1.cm_val)
1662         size++;

1664     /*
1665      * Only identify a capabilities group if the group has content. If a
1666      * capabilities identifier exists, and no other capabilities have been
1667      * supplied, remove the identifier. This scenario could exist if a
1668      * user mistakenly defined a lone identifier, or if an identified group
1669      * was overridden so as to clear the existing capabilities and the
1670      * identifier was not also cleared.
1671      */
1672     if (ocapset->oc_id.cs_str) {
1673         if (size)
1674             size++;
1675         else
1676             ocapset->oc_id.cs_str = NULL;
1677     }
1678     if (size)
1679         size++; /* Add CA_SUNW_NULL */

1681     /*
1682      * Determine how many entries are required for any symbol capabilities.
1683      */
1684     if (of1->o1_capsymcnt) {
1685         /*
1686          * If there are no object capabilities, a CA_SUNW_NULL entry
1687          * is required before any symbol capabilities.
1688          */
1689         if (size == 0)
1690             size++;
1691         size += of1->o1_capsymcnt;
1692     }

1694     if (size == 0)
1695         return (NULL);

1697     if (new_section(of1, shtype, shname, size, &isec,
1698         &shdr, &data) == S_ERROR)
1699         return (S_ERROR);

1701     if ((data->d_buf = libld_malloc(shdr->sh_size)) == NULL)
1702         return (S_ERROR);

1704     cap = (Cap *)data->d_buf;

1706     /*
1707      * Fill in any object capabilities. If there is an identifier, then the
1708      * identifier comes first. The remaining items follow in precedence

```

```

1709     * order, although the order isn't important for runtime verification.
1710     */
1711     if (ocapset->oc_id.cs_str) {
1712         ofl->ofl_flags |= FLG_OF_CAPSTRS;
1713         if (st_insert(strtbl, ocapset->oc_id.cs_str) == -1)
1714             return (S_ERROR);
1715         ocapset->oc_id.cs_ndx = capndx;
1716         CAP_UPDATE(cap, capndx, CA_SUNW_ID, 0);
1717     }
1718     if (ocapset->oc_plat.cl_val) {
1719         ofl->ofl_flags |= (FLG_OF_PTCAP | FLG_OF_CAPSTRS);
1720
1721         /*
1722          * Insert any platform name strings in the appropriate string
1723          * table. The capability value can't be filled in yet, as the
1724          * final offset of the strings isn't known until later.
1725          */
1726         for (ALIST_TRAVERSE(ocapset->oc_plat.cl_val, idx1, capstr)) {
1727             if (st_insert(strtbl, capstr->cs_str) == -1)
1728                 return (S_ERROR);
1729             capstr->cs_ndx = capndx;
1730             CAP_UPDATE(cap, capndx, CA_SUNW_PLAT, 0);
1731         }
1732     }
1733     if (ocapset->oc_mach.cl_val) {
1734         ofl->ofl_flags |= (FLG_OF_PTCAP | FLG_OF_CAPSTRS);
1735
1736         /*
1737          * Insert the machine name strings in the appropriate string
1738          * table. The capability value can't be filled in yet, as the
1739          * final offset of the strings isn't known until later.
1740          */
1741         for (ALIST_TRAVERSE(ocapset->oc_mach.cl_val, idx1, capstr)) {
1742             if (st_insert(strtbl, capstr->cs_str) == -1)
1743                 return (S_ERROR);
1744             capstr->cs_ndx = capndx;
1745             CAP_UPDATE(cap, capndx, CA_SUNW_MACH, 0);
1746         }
1747     }
1748     if (ocapset->oc_hw_2.cm_val) {
1749         ofl->ofl_flags |= FLG_OF_PTCAP;
1750         CAP_UPDATE(cap, capndx, CA_SUNW_HW_2, ocapset->oc_hw_2.cm_val);
1751     }
1752     if (ocapset->oc_hw_1.cm_val) {
1753         ofl->ofl_flags |= FLG_OF_PTCAP;
1754         CAP_UPDATE(cap, capndx, CA_SUNW_HW_1, ocapset->oc_hw_1.cm_val);
1755     }
1756     if (ocapset->oc_sf_1.cm_val) {
1757         ofl->ofl_flags |= FLG_OF_PTCAP;
1758         CAP_UPDATE(cap, capndx, CA_SUNW_SF_1, ocapset->oc_sf_1.cm_val);
1759     }
1760     CAP_UPDATE(cap, capndx, CA_SUNW_NULL, 0);
1761
1762     /*
1763     * Fill in any symbol capabilities.
1764     */
1765     if (ofl->ofl_capgroups) {
1766         Cap_group *cgp;
1767
1768         for (APLIST_TRAVERSE(ofl->ofl_capgroups, idx1, cgp)) {
1769             Objcaset *scapset = &cgp->cg_set;
1770             Aliste idx2;
1771             Is_desc *isp;
1772
1773             cgp->cg_ndx = capndx;

```

```

1775         if (scapset->oc_id.cs_str) {
1776             ofl->ofl_flags |= FLG_OF_CAPSTRS;
1777             /*
1778              * Insert the identifier string in the
1779              * appropriate string table. The capability
1780              * value can't be filled in yet, as the final
1781              * offset of the string isn't known until later.
1782              */
1783             if (st_insert(strtbl,
1784                 scapset->oc_id.cs_str) == -1)
1785                 return (S_ERROR);
1786             scapset->oc_id.cs_ndx = capndx;
1787             CAP_UPDATE(cap, capndx, CA_SUNW_ID, 0);
1788         }
1789
1790         if (scapset->oc_plat.cl_val) {
1791             ofl->ofl_flags |= FLG_OF_CAPSTRS;
1792
1793             /*
1794              * Insert the platform name string in the
1795              * appropriate string table. The capability
1796              * value can't be filled in yet, as the final
1797              * offset of the string isn't known until later.
1798              */
1799             for (ALIST_TRAVERSE(scapset->oc_plat.cl_val,
1800                 idx2, capstr)) {
1801                 if (st_insert(strtbl,
1802                     capstr->cs_str) == -1)
1803                     return (S_ERROR);
1804                 capstr->cs_ndx = capndx;
1805                 CAP_UPDATE(cap, capndx,
1806                     CA_SUNW_PLAT, 0);
1807             }
1808         }
1809         if (scapset->oc_mach.cl_val) {
1810             ofl->ofl_flags |= FLG_OF_CAPSTRS;
1811
1812             /*
1813              * Insert the machine name string in the
1814              * appropriate string table. The capability
1815              * value can't be filled in yet, as the final
1816              * offset of the string isn't known until later.
1817              */
1818             for (ALIST_TRAVERSE(scapset->oc_mach.cl_val,
1819                 idx2, capstr)) {
1820                 if (st_insert(strtbl,
1821                     capstr->cs_str) == -1)
1822                     return (S_ERROR);
1823                 capstr->cs_ndx = capndx;
1824                 CAP_UPDATE(cap, capndx,
1825                     CA_SUNW_MACH, 0);
1826             }
1827         }
1828         if (scapset->oc_hw_2.cm_val) {
1829             CAP_UPDATE(cap, capndx, CA_SUNW_HW_2,
1830                 scapset->oc_hw_2.cm_val);
1831         }
1832         if (scapset->oc_hw_1.cm_val) {
1833             CAP_UPDATE(cap, capndx, CA_SUNW_HW_1,
1834                 scapset->oc_hw_1.cm_val);
1835         }
1836         if (scapset->oc_sf_1.cm_val) {
1837             CAP_UPDATE(cap, capndx, CA_SUNW_SF_1,
1838                 scapset->oc_sf_1.cm_val);
1839         }
1840         CAP_UPDATE(cap, capndx, CA_SUNW_NULL, 0);

```



```

1842         /*
1843          * If any object capabilities are available, determine
1844          * whether these symbol capabilities are less
1845          * restrictive, and hence redundant.
1846          */
1847         if (((ofl->ofl_flags & FLG_OF_PTCAP) == 0) ||
1848             (is_cap_redundant(ocapset, scapset) == 0))
1849             continue;

1851         /*
1852          * Indicate any files that provide redundant symbol
1853          * capabilities.
1854          */
1855         for (APLIST_TRAVERSE(cgp->cg_secs, idx2, isp)) {
1856             ld_eprintf(ofl, ERR_WARNING,
1857                 MSG_INTL(MSG_CAP_REDUNDANT),
1858                 isp->is_file->ifl_name,
1859                 EC_WORD(isp->is_scndx), isp->is_name);
1860         }
1861     }
1862 }

1864 /*
1865  * If capabilities strings are required, the sh_info field of the
1866  * section header will be set to the associated string table.
1867  */
1868 if (ofl->ofl_flags & FLG_OF_CAPSTRS)
1869     shdr->sh_flags |= SHF_INFO_LINK;

1871 /*
1872  * Place these capabilities in the output file.
1873  */
1874 if ((ofl->ofl_oscaps = ld_place_section(ofl, isec,
1875     NULL, ident, NULL)) == (Os_desc *)S_ERROR)
1876     return (S_ERROR);

1878 /*
1879  * If symbol capabilities are required, then a .SUNW_capinfo section is
1880  * also created. This table will eventually be sized to match the
1881  * associated symbol table.
1882  */
1883 if (ofl->ofl_capfamilies) {
1884     if ((ofl->ofl_oscaps = make_sym_sec(ofl,
1885         MSG_ORIG(MSG_SCN_SUNWCAPINFO), SHT_SUNW_capinfo,
1886         ld_targ.t_id.id_capinfo)) == (Os_desc *)S_ERROR)
1887         return (S_ERROR);

1889     /*
1890      * If we're generating a dynamic object, capabilities family
1891      * members are maintained in a .SUNW_capchain section.
1892      */
1893     if (ofl->ofl_capchaincnt &&
1894         ((ofl->ofl_flags & FLG_OF_RELOBJ) == 0)) {
1895         if (new_section(ofl, SHT_SUNW_capchain,
1896             MSG_ORIG(MSG_SCN_SUNWCAPCHAIN),
1897             ofl->ofl_capchaincnt, &isec, &shdr,
1898             &data) == S_ERROR)
1899             return (S_ERROR);

1901         ofl->ofl_oscapschain = ld_place_section(ofl, isec,
1902             NULL, ld_targ.t_id.id_capchain, NULL);
1903         if (ofl->ofl_oscapschain == (Os_desc *)S_ERROR)
1904             return (S_ERROR);
1905     }
1906 }

```

```

1907     }
1908     return (1);
1909 }
1910 #undef CAP_UPDATE

1912 /*
1913  * Build the PLT section and its associated relocation entries.
1914  */
1915 static uintptr_t
1916 make_plt(Of1_desc *ofl)
1917 {
1918     Shdr          *shdr;
1919     Elf_Data      *data;
1920     Is_desc       *isec;
1921     size_t        size = ld_targ.t_m.m_plt_reservsz +
1922         (((size_t)ofl->ofl_pltcnt + (size_t)ofl->ofl_pltpad) *
1923         ld_targ.t_m.m_plt_entsize);
1924     size_t        rsize = (size_t)ofl->ofl_relocpltsz;

1926     /*
1927      * On sparc, account for the NOP at the end of the plt.
1928      */
1929     if (ld_targ.t_m.m_mach == LD_TARG_BYCLASS(EM_SPARC, EM_SPARCV9))
1930         size += sizeof (Word);

1932     if (new_section(ofl, SHT_PROGBITS, MSG_ORIG(MSG_SCN_PLT), 0,
1933         &isec, &shdr, &data) == S_ERROR)
1934         return (S_ERROR);

1936     data->d_size = size;
1937     data->d_align = ld_targ.t_m.m_plt_align;

1939     shdr->sh_flags = ld_targ.t_m.m_plt_shf_flags;
1940     shdr->sh_size = (Xword)size;
1941     shdr->sh_addralign = ld_targ.t_m.m_plt_align;
1942     shdr->sh_entsize = ld_targ.t_m.m_plt_entsize;

1944     ofl->ofl_osplt = ld_place_section(ofl, isec, NULL,
1945         ld_targ.t_id.id_plt, NULL);
1946     if (ofl->ofl_osplt == (Os_desc *)S_ERROR)
1947         return (S_ERROR);

1949     ofl->ofl_osplt->os_szoutrels = (Xword)rsize;

1951     return (1);
1952 }

1954 /*
1955  * Make the hash table. Only built for dynamic executables and shared
1956  * libraries, and provides hashed lookup into the global symbol table
1957  * (.dynsym) for the run-time linker to resolve symbol lookups.
1958  */
1959 static uintptr_t
1960 make_hash(Of1_desc *ofl)
1961 {
1962     Shdr          *shdr;
1963     Elf_Data      *data;
1964     Is_desc       *isec;
1965     size_t        size;
1966     Word          nsyms = ofl->ofl_globcnt;
1967     size_t        cnt;

1969     /*
1970      * Allocate section header structures. We set entcnt to 0
1971      * because it's going to change after we place this section.
1972      */

```

```

1973     if (new_section(ofl, SHT_HASH, MSG_ORIG(MSG_SCN_HASH), 0,
1974                 &isec, &shdr, &data) == S_ERROR)
1975         return (S_ERROR);

1977     /*
1978     * Place the section first since it will affect the local symbol
1979     * count.
1980     */
1981     ofl->ofl_oshash =
1982         ld_place_section(ofl, isec, NULL, ld_targ.t_id.id_hash, NULL);
1983     if (ofl->ofl_oshash == (Os_desc *)S_ERROR)
1984         return (S_ERROR);

1986     /*
1987     * Calculate the number of output hash buckets.
1988     */
1989     ofl->ofl_hashbkts = findprime(nsyms);

1991     /*
1992     * The size of the hash table is determined by
1993     *
1994     *   i.    the initial nbucket and nchain entries (2)
1995     *   ii.   the number of buckets (calculated above)
1996     *   iii.  the number of chains (this is based on the number of
1997     *         symbols in the .dynsym array).
1998     */
1999     cnt = 2 + ofl->ofl_hashbkts + DYNYSM_ALL_CNT(ofl);
2000     size = cnt * shdr->sh_entsize;

2002     /*
2003     * Finalize the section header and data buffer initialization.
2004     */
2005     if ((data->d_buf = libld_calloc(size, 1)) == NULL)
2006         return (S_ERROR);
2007     data->d_size = size;
2008     shdr->sh_size = (Xword)size;

2010     return (1);
2011 }

2013 /*
2014 * Generate the standard symbol table. Contains all locals and globals,
2015 * and resides in a non-allocatable section (ie. it can be stripped).
2016 */
2017 static uintptr_t
2018 make_symtab(Of1_desc *ofl)
2019 {
2020     Shdr      *shdr;
2021     Elf_Data  *data;
2022     Is_desc   *isec;
2023     Is_desc   *xisec = 0;
2024     size_t    size;
2025     Word      symcnt;

2027     /*
2028     * Create the section headers. Note that we supply an ent_cnt
2029     * of 0. We won't know the count until the section has been placed.
2030     */
2031     if (new_section(ofl, SHT_SYMTAB, MSG_ORIG(MSG_SCN_SYMTAB), 0,
2032                 &isec, &shdr, &data) == S_ERROR)
2033         return (S_ERROR);

2035     /*
2036     * Place the section first since it will affect the local symbol
2037     * count.
2038     */

```

```

2039     if ((ofl->ofl_ossymtab = ld_place_section(ofl, isec, NULL,
2040         ld_targ.t_id.id_symtab, NULL)) == (Os_desc *)S_ERROR)
2041         return (S_ERROR);

2043     /*
2044     * At this point we've created all but the 'shstrtab' section.
2045     * Determine if we have to use 'Extended Sections'. If so - then
2046     * also create a SHT_SYMTAB_SHNDX section.
2047     */
2048     if ((ofl->ofl_shdrcont + 1) >= SHN_LORESERVE) {
2049         Shdr      *xshdr;
2050         Elf_Data  *xdata;

2052         if (new_section(ofl, SHT_SYMTAB_SHNDX,
2053             MSG_ORIG(MSG_SCN_SYMTAB_SHNDX), 0, &xisec,
2054             &xshdr, &xdata) == S_ERROR)
2055             return (S_ERROR);

2057         if ((ofl->ofl_ossymshndx = ld_place_section(ofl, xisec, NULL,
2058             ld_targ.t_id.id_symtab_ndx, NULL)) == (Os_desc *)S_ERROR)
2059             return (S_ERROR);
2060     }

2062     /*
2063     * Calculated number of symbols, which need to be augmented by
2064     * the (yet to be created) .shstrtab entry.
2065     */
2066     symcnt = (size_t)(1 + SYMTAB_ALL_CNT(ofl));
2067     size = symcnt * shdr->sh_entsize;

2069     /*
2070     * Finalize the section header and data buffer initialization.
2071     */
2072     data->d_size = size;
2073     shdr->sh_size = (Xword)size;

2075     /*
2076     * If we created a SHT_SYMTAB_SHNDX - then set it's sizes too.
2077     */
2078     if (xisec) {
2079         size_t  xsize = symcnt * sizeof (Word);

2081         xisec->is_indata->d_size = xsize;
2082         xisec->is_shdr->sh_size = (Xword)xsize;
2083     }

2085     return (1);
2086 }

2088 /*
2089 * Build a dynamic symbol table. These tables reside in the text
2090 * segment of a dynamic executable or shared library.
2091 *
2092 *   .SUNW_ldynsym contains local function symbols
2093 *   .dynsym contains only globals symbols
2094 *
2095 * The two tables are created adjacent to each other, with .SUNW_ldynsym
2096 * coming first.
2097 */
2098 static uintptr_t
2099 make_dynsym(Of1_desc *ofl)
2100 {
2101     Shdr      *shdr, *lshdr;
2102     Elf_Data  *data, *ldata;
2103     Is_desc   *isec, *lsec;
2104     size_t    size;

```

```

2105     Xword      cnt;
2106     int        allow_ldynsym;

2108 /*
2109  * Unless explicitly disabled, always produce a .SUNW_ldynsym section
2110  * when it is allowed by the file type, even if the resulting
2111  * table only ends up with a single STT_FILE in it. There are
2112  * two reasons: (1) It causes the generation of the DT_SUNW_SYMTAB
2113  * entry in the .dynamic section, which is something we would
2114  * like to encourage, and (2) Without it, we cannot generate
2115  * the associated .SUNW_dyn[sym|tls]sort sections, which are of
2116  * value to DTrace.
2117  *
2118  * In practice, it is extremely rare for an object not to have
2119  * local symbols for .SUNW_ldynsym, so 99% of the time, we'd be
2120  * doing it anyway.
2121  */
2122 allow_ldynsym = OFL_ALLOW_LDYNSYM(ofl);

2124 /*
2125  * Create the section headers. Note that we supply an ent_cnt
2126  * of 0. We won't know the count until the section has been placed.
2127  */
2128 if (allow_ldynsym && new_section(ofl, SHT_SUNW_LDYNSYM,
2129     MSG_ORIG(MSG_SCN_LDYNSYM), 0, &lsec, &lshdr, &data) == S_ERROR)
2130     return (S_ERROR);

2132 if (new_section(ofl, SHT_DYNSYM, MSG_ORIG(MSG_SCN_DYNSYM), 0,
2133     &lsec, &lshdr, &data) == S_ERROR)
2134     return (S_ERROR);

2136 /*
2137  * Place the section(s) first since it will affect the local symbol
2138  * count.
2139  */
2140 if (allow_ldynsym &&
2141     ((ofl->ofl_osldynsym = ld_place_section(ofl, lsec, NULL,
2142     ld_targ.t_id.id_ldynsym, NULL)) == (Os_desc *)S_ERROR))
2143     return (S_ERROR);
2144 ofl->ofl_osdynsym =
2145     ld_place_section(ofl, lsec, NULL, ld_targ.t_id.id_dynsym, NULL);
2146 if (ofl->ofl_osdynsym == (Os_desc *)S_ERROR)
2147     return (S_ERROR);

2149 cnt = DYNSYM_ALL_CNT(ofl);
2150 size = (size_t)cnt * shdr->sh_entsize;

2152 /*
2153  * Finalize the section header and data buffer initialization.
2154  */
2155 data->d_size = size;
2156 shdr->sh_size = (Xword)size;

2158 /*
2159  * An ldynsym contains local function symbols. It is not
2160  * used for linking, but if present, serves to allow better
2161  * stack traces to be generated in contexts where the symtab
2162  * is not available. (dladdr(), or stripped executable/library files).
2163  */
2164 if (allow_ldynsym) {
2165     cnt = 1 + ofl->ofl_dynlocscnt + ofl->ofl_dynscopecnt;
2166     size = (size_t)cnt * shdr->sh_entsize;

2168     ldata->d_size = size;
2169     lshdr->sh_size = (Xword)size;
2170 }

```

```

2172     return (1);
2173 }

2175 /*
2176  * Build .SUNW_dynsym sort and/or .SUNW_dyntlssort sections. These are
2177  * index sections for the .SUNW_ldynsym/.dynsym pair that present data
2178  * and function symbols sorted by address.
2179  */
2180 static uintptr_t
2181 make_dynsort(Of1_desc *ofl)
2182 {
2183     Shdr      *shdr;
2184     Elf_Data  *data;
2185     Is_desc   *lsec;

2187     /* Only do it if the .SUNW_ldynsym section is present */
2188     if (!OFL_ALLOW_LDYNSYM(ofl))
2189         return (1);

2191     /* .SUNW_dynsym sort */
2192     if (ofl->ofl_dynsym sortcnt > 0) {
2193         if (new_section(ofl, SHT_SUNW_SYMSORT,
2194             MSG_ORIG(MSG_SCN_DYNSYMSORT), ofl->ofl_dynsym sortcnt,
2195             &lsec, &lshdr, &data) == S_ERROR)
2196             return (S_ERROR);

2198         if ((ofl->ofl_osdynsym sort = ld_place_section(ofl, lsec, NULL,
2199             ld_targ.t_id.id_dynsort, NULL)) == (Os_desc *)S_ERROR)
2200             return (S_ERROR);
2201     }

2203     /* .SUNW_dyntlssort */
2204     if (ofl->ofl_dyntlssortcnt > 0) {
2205         if (new_section(ofl, SHT_SUNW_TLSSORT,
2206             MSG_ORIG(MSG_SCN_DYNTLSSORT),
2207             ofl->ofl_dyntlssortcnt, &lsec, &lshdr, &data) == S_ERROR)
2208             return (S_ERROR);

2210         if ((ofl->ofl_osdyntlssort = ld_place_section(ofl, lsec, NULL,
2211             ld_targ.t_id.id_dynsort, NULL)) == (Os_desc *)S_ERROR)
2212             return (S_ERROR);
2213     }

2215     return (1);
2216 }

2218 /*
2219  * Helper routine for make_dynsym_shndx. Builds a
2220  * a SHT_SYMTAB_SHNDX for .dynsym or .SUNW_ldynsym, without knowing
2221  * which one it is.
2222  */
2223 static uintptr_t
2224 make_dyn_shndx(Of1_desc *ofl, const char *shname, Os_desc *symtab,
2225     Os_desc **ret_os)
2226 {
2227     Is_desc   *lsec;
2228     Is_desc   *dynsymisp;
2229     Shdr      *shdr;
2230     Elf_Data  *data;

2232     dynsymisp = ld_os_first_isdesc(symtab);
2233     dynshdr = dynsymisp->is_shdr;

2235     if (new_section(ofl, SHT_SYMTAB_SHNDX, shname,
2236         (dynshdr->sh_size / dynshdr->sh_entsize),

```

```

2237     &isec, &shdr, &data) == S_ERROR)
2238         return (S_ERROR);

2240     if ((*ret_os = ld_place_section(ofl, isec, NULL,
2241         ld_targ.t_id.id_dynsym_ndx, NULL)) == (Os_desc *)S_ERROR)
2242         return (S_ERROR);

2244     assert(*ret_os);

2246     return (1);
2247 }

2249 /*
2250  * Build a SHT_SYMTAB_SHNDX for the .dynsym, and .SUNW_ldynsym
2251  */
2252 static uintptr_t
2253 make_dynsym_shndx(Of1_desc *ofl)
2254 {
2255     /*
2256      * If there is a .SUNW_ldynsym, generate a section for its extended
2257      * index section as well.
2258      */
2259     if (OFL_ALLOW_LDYNSYM(ofl)) {
2260         if (make_dyn_shndx(ofl, MSG_ORIG(MSG_SCN_LDYNSYM_SHNDX),
2261             ofl->ofl_osldynsym, &ofl->ofl_osldynshndx) == S_ERROR)
2262             return (S_ERROR);
2263     }

2265     /* The Generate a section for the dynsym */
2266     if (make_dyn_shndx(ofl, MSG_ORIG(MSG_SCN_DYNSYM_SHNDX),
2267         ofl->ofl_osdynsym, &ofl->ofl_osdynshndx) == S_ERROR)
2268         return (S_ERROR);

2270     return (1);
2271 }

2274 /*
2275  * Build a string table for the section headers.
2276  */
2277 static uintptr_t
2278 make_shstrtab(Of1_desc *ofl)
2279 {
2280     Shdr      *shdr;
2281     Elf_Data  *data;
2282     Is_desc   *isec;
2283     size_t    size;

2285     if (new_section(ofl, SHT_STRTAB, MSG_ORIG(MSG_SCN_SHSTRTAB),
2286         0, &isec, &shdr, &data) == S_ERROR)
2287         return (S_ERROR);

2289     /*
2290      * Place the section first, as it may effect the number of section
2291      * headers to account for.
2292      */
2293     ofl->ofl_osshstrtab =
2294         ld_place_section(ofl, isec, NULL, ld_targ.t_id.id_note, NULL);
2295     if (ofl->ofl_osshstrtab == (Os_desc *)S_ERROR)
2296         return (S_ERROR);

2298     size = st_getstrtab_sz(ofl->ofl_shdrsttab);
2299     assert(size > 0);

2301     data->d_size = size;
2302     shdr->sh_size = (Xword)size;

```

```

2304         return (1);
2305 }

2307 /*
2308  * Build a string section for the standard symbol table.
2309  */
2310 static uintptr_t
2311 make_strtab(Of1_desc *ofl)
2312 {
2313     Shdr      *shdr;
2314     Elf_Data  *data;
2315     Is_desc   *isec;
2316     size_t    size;

2318     /*
2319      * This string table consists of all the global and local symbols.
2320      * Account for null bytes at end of the file name and the beginning
2321      * of section.
2322      */
2323     if (st_insert(ofl->ofl_strtab, ofl->ofl_name) == -1)
2324         return (S_ERROR);

2326     size = st_getstrtab_sz(ofl->ofl_strtab);
2327     assert(size > 0);

2329     if (new_section(ofl, SHT_STRTAB, MSG_ORIG(MSG_SCN_STRTAB),
2330         0, &isec, &shdr, &data) == S_ERROR)
2331         return (S_ERROR);

2333     /* Set the size of the data area */
2334     data->d_size = size;
2335     shdr->sh_size = (Xword)size;

2337     ofl->ofl_osstrtab =
2338         ld_place_section(ofl, isec, NULL, ld_targ.t_id.id_strtab, NULL);
2339     return ((uintptr_t)ofl->ofl_osstrtab);
2340 }

2342 /*
2343  * Build a string table for the dynamic symbol table.
2344  */
2345 static uintptr_t
2346 make_dynstr(Of1_desc *ofl)
2347 {
2348     Shdr      *shdr;
2349     Elf_Data  *data;
2350     Is_desc   *isec;
2351     size_t    size;

2353     /*
2354      * If producing a .SUNW_ldynsym, account for the initial STT_FILE
2355      * symbol that precedes the scope reduced global symbols.
2356      */
2357     if (OFL_ALLOW_LDYNSYM(ofl)) {
2358         if (st_insert(ofl->ofl_dynstrtab, ofl->ofl_name) == -1)
2359             return (S_ERROR);
2360         ofl->ofl_dynscopecnt++;
2361     }

2363     /*
2364      * Account for any local, named register symbols. These locals are
2365      * required for reference from DT_REGISTER .dynamic entries.
2366      */
2367     if (ofl->ofl_regsyms) {
2368         int     ndx;

```

```

2370     for (ndx = 0; ndx < ofl->ofl_regssymsno; ndx++) {
2371         Sym_desc      *sdp;

2373         if ((sdp = ofl->ofl_regssyms[ndx]) == NULL)
2374             continue;

2376         if (!SYM_IS_HIDDEN(sdp) &&
2377             (ELF_ST_BIND(sdp->sd_sym->st_info) != STB_LOCAL))
2378             continue;

2380         if (sdp->sd_sym->st_name == NULL)
2381             continue;

2383         if (st_insert(ofl->ofl_dynstrtab, sdp->sd_name) == -1)
2384             return (S_ERROR);
2385     }
2386 }

2388 /*
2389  * Reserve entries for any per-symbol auxiliary/filter strings.
2390  */
2391 if (ofl->ofl_dtsfltrs != NULL) {
2392     Dfltr_desc      *dftp;
2393     Aliste          idx;

2395     for (ALIST_TRAVERSE(ofl->ofl_dtsfltrs, idx, dftp))
2396         if (st_insert(ofl->ofl_dynstrtab, dftp->dft_str) == -1)
2397             return (S_ERROR);
2398 }

2400 size = st_getstrtab_sz(ofl->ofl_dynstrtab);
2401 assert(size > 0);

2403 if (new_section(ofl, SHT_STRTAB, MSG_ORIG(MSG_SCN_DYNSTR),
2404                0, &isec, &shdr, &data) == S_ERROR)
2405     return (S_ERROR);

2407 /* Make it allocable if necessary */
2408 if (!(ofl->ofl_flags & FLG_OF_RELOBJ))
2409     shdr->sh_flags |= SHF_ALLOC;

2411 /* Set the size of the data area */
2412 data->d_size = size + DYNSTR_EXTRA_PAD;

2414 shdr->sh_size = (Xword)size;

2416 ofl->ofl_osdynstr =
2417     ld_place_section(ofl, isec, NULL, ld_targ.t_id.id_dynstr, NULL);
2418 return ((uintptr_t)ofl->ofl_osdynstr);
2419 }

2421 /*
2422  * Generate an output relocation section which will contain the relocation
2423  * information to be applied to the 'osp' section.
2424  */
2425 * If (osp == NULL) then we are creating the coalesced relocation section
2426 * for an executable and/or a shared object.
2427 */
2428 static uintptr_t
2429 make_reloc(Of1_desc *ofl, Os_desc *osp)
2430 {
2431     Shdr      *shdr;
2432     Elf_Data  *data;
2433     Is_desc   *isec;
2434     size_t    size;

```

```

2435     Xword      sh_flags;
2436     char       *sectname;
2437     Os_desc    *rosp;
2438     Word       relsize;
2439     const char *rel_prefix;

2441     /* LINTED */
2442     if (ld_targ.t_m.m_rel_sht_type == SHT_REL) {
2443         /* REL */
2444         relsize = sizeof (Rel);
2445         rel_prefix = MSG_ORIG(MSG_SCN_REL);
2446     } else {
2447         /* RELA */
2448         relsize = sizeof (Rela);
2449         rel_prefix = MSG_ORIG(MSG_SCN_RELA);
2450     }

2452     if (osp) {
2453         size = osp->os_szoutrels;
2454         sh_flags = osp->os_shdr->sh_flags;
2455         if ((sectname = libld_malloc(strlen(rel_prefix) +
2456                                     strlen(osp->os_name) + 1)) == 0)
2457             return (S_ERROR);
2458         (void) strcpy(sectname, rel_prefix);
2459         (void) strcat(sectname, osp->os_name);
2460     } else if (ofl->ofl_flags & FLG_OF_COMREL) {
2461         size = (ofl->ofl_relocct - ofl->ofl_relocctsub) * relsize;
2462         sh_flags = SHF_ALLOC;
2463         sectname = (char *)MSG_ORIG(MSG_SCN_SUNWRELOC);
2464     } else {
2465         size = ofl->ofl_relocrelsz;
2466         sh_flags = SHF_ALLOC;
2467         sectname = (char *)rel_prefix;
2468     }

2470     /*
2471      * Keep track of total size of 'output relocations' (to be stored
2472      * in .dynamic)
2473      */
2474     /* LINTED */
2475     ofl->ofl_relocsz += (Xword)size;

2477     if (new_section(ofl, ld_targ.t_m.m_rel_sht_type, sectname, 0, &isec,
2478                    &shdr, &data) == S_ERROR)
2479         return (S_ERROR);

2481     data->d_size = size;

2483     shdr->sh_size = (Xword)size;
2484     if (OFL_ALLOW_DYNSYM(ofl) && (sh_flags & SHF_ALLOC))
2485         shdr->sh_flags = SHF_ALLOC;

2487     if (osp) {
2488         /*
2489          * The sh_info field of the SHT_REL* sections points to the
2490          * section the relocations are to be applied to.
2491          */
2492         shdr->sh_info = (uintptr_t)ofl->ofl_relocctsub;
2493     }

2495     rosp = ld_place_section(ofl, isec, NULL, ld_targ.t_id.id_rel, NULL);
2496     if (rosp == (Os_desc *)S_ERROR)
2497         return (S_ERROR);

2499     /*
2500      * Associate this relocation section to the section its going to

```

```

2501     * relocate.
2502     */
2503     if (osp) {
2504         Aliste idx;
2505         Is_desc *risp;

2507         /*
2508          * This is used primarily so that we can update
2509          * SHT_GROUP[sect_no] entries to point to the
2510          * created output relocation sections.
2511          */
2512         for (APLIST_TRAVERSE(osp->os_relisdescs, idx, risp)) {
2513             risp->is_osdesc = rosp;

2515             /*
2516              * If the input relocation section had the SHF_GROUP
2517              * flag set - propagate it to the output relocation
2518              * section.
2519              */
2520             if (risp->is_shdr->sh_flags & SHF_GROUP) {
2521                 rosp->os_shdr->sh_flags |= SHF_GROUP;
2522                 break;
2523             }
2524             osp->os_relosdesc = rosp;
2525         } else
2526             ofl->ofl_osrel = rosp;
2527
2529         /*
2530          * If this is the first relocation section we've encountered save it
2531          * so that the .dynamic entry can be initialized accordingly.
2532          */
2533         if (ofl->ofl_osrelhead == (Os_desc *)0)
2534             ofl->ofl_osrelhead = rosp;

2536     return (1);
2537 }

2539 /*
2540 * Generate version needed section.
2541 */
2542 static uintptr_t
2543 make_verneed(Of1_desc *ofl)
2544 {
2545     Shdr          *shdr;
2546     Elf_Data      *data;
2547     Is_desc       *isec;

2549     /*
2550      * verneed sections do not have a constant element size, so the
2551      * value of ent_cnt specified here (0) is meaningless.
2552      */
2553     if (new_section(ofl, SHT_SUNW_verneed, MSG_ORIG(MSG_SCN_SUNWVERSION),
2554                    0, &isec, &shdr, &data) == S_ERROR)
2555         return (S_ERROR);

2557     /* During version processing we calculated the total size. */
2558     data->d_size = ofl->ofl_verneedsz;
2559     shdr->sh_size = (Xword)ofl->ofl_verneedsz;

2561     ofl->ofl_osverneed =
2562         ld_place_section(ofl, isec, NULL, ld_targ.t_id.id_version, NULL);
2563     return ((uintptr_t)ofl->ofl_osverneed);
2564 }

2566 /*

```

```

2567 * Generate a version definition section.
2568 *
2569 * o the SHT_SUNW_verdef section defines the versions that exist within this
2570 * image.
2571 */
2572 static uintptr_t
2573 make_verdef(Of1_desc *ofl)
2574 {
2575     Shdr          *shdr;
2576     Elf_Data      *data;
2577     Is_desc       *isec;
2578     Ver_desc      *vdp;
2579     Str_tbl       *strtab;

2581     /*
2582      * Reserve a string table entry for the base version dependency (other
2583      * dependencies have symbol representations, which will already be
2584      * accounted for during symbol processing).
2585      */
2586     vdp = (Ver_desc *)ofl->ofl_verdesc->apl_data[0];

2588     if (OFL_IS_STATIC_OBJ(ofl))
2589         strtab = ofl->ofl_strtab;
2590     else
2591         strtab = ofl->ofl_dynstrtab;

2593     if (st_insert(strtab, vdp->vd_name) == -1)
2594         return (S_ERROR);

2596     /*
2597      * verdef sections do not have a constant element size, so the
2598      * value of ent_cnt specified here (0) is meaningless.
2599      */
2600     if (new_section(ofl, SHT_SUNW_verdef, MSG_ORIG(MSG_SCN_SUNWVERSION),
2601                    0, &isec, &shdr, &data) == S_ERROR)
2602         return (S_ERROR);

2604     /* During version processing we calculated the total size. */
2605     data->d_size = ofl->ofl_verdefsz;
2606     shdr->sh_size = (Xword)ofl->ofl_verdefsz;

2608     ofl->ofl_osverdef =
2609         ld_place_section(ofl, isec, NULL, ld_targ.t_id.id_version, NULL);
2610     return ((uintptr_t)ofl->ofl_osverdef);
2611 }

2613 /*
2614 * This routine is called when -z nopartial is in effect.
2615 */
2616 uintptr_t
2617 ld_make_parexp_data(Of1_desc *ofl, size_t size, Xword align)
2618 {
2619     Shdr          *shdr;
2620     Elf_Data      *data;
2621     Is_desc       *isec;
2622     Os_desc       *osp;

2624     if (new_section(ofl, SHT_PROGBITS, MSG_ORIG(MSG_SCN_DATA), 0,
2625                    &isec, &shdr, &data) == S_ERROR)
2626         return (S_ERROR);

2628     shdr->sh_flags |= SHF_WRITE;
2629     data->d_size = size;
2630     shdr->sh_size = (Xword)size;
2631     if (align != 0) {
2632         data->d_align = align;

```

```

2633     shdr->sh_addralign = align;
2634 }

2636 if ((data->d_buf = libld_calloc(size, 1)) == NULL)
2637     return (S_ERROR);

2639 /*
2640  * Retain handle to this .data input section. Variables using move
2641  * sections (partial initialization) will be redirected here when
2642  * such global references are added and '-z nopartial' is in effect.
2643  */
2644 ofl->ofl_isparexpn = isec;
2645 osp = ld_place_section(ofl, isec, NULL, ld_targ.t_id.id_data, NULL);
2646 if (osp == (Os_desc *)S_ERROR)
2647     return (S_ERROR);

2649 if (!(osp->os_flags & FLG_OS_OUTREL)) {
2650     ofl->ofl_dynshdrct++;
2651     osp->os_flags |= FLG_OS_OUTREL;
2652 }
2653 return (1);
2654 }

2656 /*
2657  * Make .sunwmove section
2658  */
2659 uintptr_t
2660 ld_make_sunwmove(Of1_desc *ofl, int mv_nums)
2661 {
2662     Shdr      *shdr;
2663     Elf_Data  *data;
2664     Is_desc   *isec;
2665     Aliste    idx;
2666     Sym_desc  *sdp;
2667     int       cnt = 1;

2670 if (new_section(ofl, SHT_SUNW_move, MSG_ORIG(MSG_SCN_SUNWMOVE),
2671 mv_nums, &isec, &shdr, &data) == S_ERROR)
2672     return (S_ERROR);

2674 if ((data->d_buf = libld_calloc(data->d_size, 1)) == NULL)
2675     return (S_ERROR);

2677 /*
2678  * Copy move entries
2679  */
2680 for (APLIST_TRAVERSE(ofl->ofl_parsyms, idx, sdp)) {
2681     Aliste    idx2;
2682     Mv_desc   *mdp;

2684     if (sdp->sd_flags & FLG_SY_PAREXPN)
2685         continue;

2687     for (ALIST_TRAVERSE(sdp->sd_move, idx2, mdp))
2688         mdp->md_oidx = cnt++;
2689 }

2691 if ((ofl->ofl_osmove = ld_place_section(ofl, isec, NULL, 0, NULL)) ==
2692 (Os_desc *)S_ERROR)
2693     return (S_ERROR);

2695 return (1);
2696 }

2698 /*

```

```

2699 * Given a relocation descriptor that references a string table
2700 * input section, locate the string referenced and return a pointer
2701 * to it.
2702 */
2703 static const char *
2704 strmerge_get_reloc_str(Of1_desc *ofl, Rel_desc *rsp)
2705 {
2706     Sym_desc *sdp = rsp->rel_sym;
2707     Xword    str_off;

2709     /*
2710     * In the case of an STT_SECTION symbol, the addend of the
2711     * relocation gives the offset into the string section. For
2712     * other symbol types, the symbol value is the offset.
2713     */

2715     if (ELF_ST_TYPE(sdp->sd_sym->st_info) != STT_SECTION) {
2716         str_off = sdp->sd_sym->st_value;
2717     } else if ((rsp->rel_flags & FLG_REL_RELA) == FLG_REL_RELA) {
2718         /*
2719          * For SHT_RELA, the addend value is found in the
2720          * rel_raddend field of the relocation.
2721          */
2722         str_off = rsp->rel_raddend;
2723     } else { /* REL and STT_SECTION */
2724         /*
2725          * For SHT_REL, the "addend" is not part of the relocation
2726          * record. Instead, it is found at the relocation target
2727          * address.
2728          */
2729         uchar_t *addr = (uchar_t *)((uintptr_t)rsp->rel_robfd +
2730 (uintptr_t)rsp->rel_isdesc->is_indata->d_buf);

2732         if (ld_reloc_targval_get(ofl, rsp, addr, &str_off) == 0)
2733             return (0);
2734     }

2736     return (str_off + (char *)sdp->sd_isc->is_indata->d_buf);
2737 }

2739 /*
2740 * First pass over the relocation records for string table merging.
2741 * Build lists of relocations and symbols that will need modification,
2742 * and insert the strings they reference into the mstrtab string table.
2743 */
2744 * entry:
2745 * ofl, osp - As passed to ld_make_strmerge().
2746 * mstrtab - String table to receive input strings. This table
2747 * must be in its first (initialization) pass and not
2748 * yet cooked (st_getstrtab_sz() not yet called).
2749 * rel_alpp - APlist to receive pointer to any relocation
2750 * descriptors with STT_SECTION symbols that reference
2751 * one of the input sections being merged.
2752 * sym_alpp - APlist to receive pointer to any symbols that reference
2753 * one of the input sections being merged.
2754 * rcp - Pointer to cache of relocation descriptors to examine.
2755 * Either &o1->o1_actrels (active relocations)
2756 * or &o1->o1_outrels (output relocations).
2757 *
2758 * exit:
2759 * On success, rel_alpp and sym_alpp are updated, and
2760 * any strings in the mergeable input sections referenced by
2761 * a relocation has been entered into mstrtab. True (1) is returned.
2762 *
2763 * On failure, False (0) is returned.
2764 */

```

```

2765 static int
2766 strmerge_pass1(Of1_desc *of1, Os_desc *osp, Str_tbl *mstrtab,
2767             APlist **rel_alpp, APlist **sym_alpp, Rel_cache *rcp)
2768 {
2769     Aliste      idx;
2770     Rel_cachebuf *rcbp;
2771     Sym_desc    *sdp;
2772     Sym_desc    *last_sdp = NULL;
2773     Rel_desc    *rsp;
2774     const char  *name;

2776     REL_CACHE_TRAVERSE(rcp, idx, rcbp, rsp) {
2777         sdp = rsp->rel_sym;
2778         if ((sdp->sd_isc == NULL) || ((sdp->sd_isc->is_flags &
2779             (FLG_IS_DISCARD | FLG_IS_INSTRMG)) != FLG_IS_INSTRMG) ||
2780             (sdp->sd_isc->is_osdesc != osp))
2781             continue;

2783         /*
2784          * Remember symbol for use in the third pass. There is no
2785          * reason to save a given symbol more than once, so we take
2786          * advantage of the fact that relocations to a given symbol
2787          * tend to cluster in the list. If this is the same symbol
2788          * we saved last time, don't bother.
2789          */
2790         if (last_sdp != sdp) {
2791             if (aplist_append(sym_alpp, sdp, AL_CNT_STRMRGSYM) ==
2792                 NULL)
2793                 return (0);
2794             last_sdp = sdp;
2795         }

2797         /* Enter the string into our new string table */
2798         name = strmerge_get_reloc_str(of1, rsp);
2799         if (st_insert(mstrtab, name) == -1)
2800             return (0);

2802         /*
2803          * If this is an STT_SECTION symbol, then the second pass
2804          * will need to modify this relocation, so hang on to it.
2805          */
2806         if ((ELF_ST_TYPE(sdp->sd_sym->st_info) == STT_SECTION) &&
2807             (aplist_append(rel_alpp, rsp, AL_CNT_STRMRGREL) == NULL))
2808             return (0);
2809     }

2811     return (1);
2812 }

2814 /*
2815  * If the output section has any SHF_MERGE|SHF_STRINGS input sections,
2816  * replace them with a single merged/compressed input section.
2817  *
2818  * entry:
2819  *   of1 - Output file descriptor
2820  *   osp - Output section descriptor
2821  *   rel_alpp, sym_alpp, - Address of 2 APlists, to be used
2822  *   for internal processing. On the initial call to
2823  *   ld_make_strmerge, these list pointers must be NULL.
2824  *   The caller is encouraged to pass the same lists back for
2825  *   successive calls to this function without freeing
2826  *   them in between calls. This causes a single pair of
2827  *   memory allocations to be reused multiple times.
2828  *
2829  * exit:
2830  *   If section merging is possible, it is done. If no errors are

```

```

2831  *   encountered, True (1) is returned. On error, S_ERROR.
2832  *
2833  *   The contents of rel_alpp and sym_alpp on exit are
2834  *   undefined. The caller can free them, or pass them back to a subsequent
2835  *   call to this routine, but should not examine their contents.
2836  */
2837 static uintptr_t
2838 ld_make_strmerge(Of1_desc *of1, Os_desc *osp, APlist **rel_alpp,
2839                 APlist **sym_alpp)
2840 {
2841     Str_tbl      *mstrtab;      /* string table for string merge secs */
2842     Is_desc      *mstrsec;      /* Generated string merge section */
2843     Is_desc      *isp;
2844     Shdr         *mstr_shdr;
2845     Elf_Data     *mstr_data;
2846     Sym_desc     *sdp;
2847     Rel_desc     *rsp;
2848     Aliste       idx;
2849     size_t       data_size;
2850     int          st_setstring_status;
2851     size_t       stoff;

2853     /* If string table compression is disabled, there's nothing to do */
2854     if ((of1->o1_flags1 & FLG_OF1_NCSTTAB) != 0)
2855         return (1);

2857     /*
2858      * Pass over the mergeable input sections, and if they haven't
2859      * all been discarded, create a string table.
2860      */
2861     mstrtab = NULL;
2862     for (APLIST_TRAVERSE(osp->os_mstrisdescs, idx, isp) {
2863         if (isdesc_discarded(isp))
2864             continue;

2866         /*
2867          * Input sections of 0 size are dubiously valid since they do
2868          * not even contain the NUL string. Ignore them.
2869          */
2870         if (isp->is_shdr->sh_size == 0)
2871             continue;

2873         /*
2874          * We have at least one non-discarded section.
2875          * Create a string table descriptor.
2876          */
2877         if ((mstrtab = st_new(FLG_STNEW_COMPRESS)) == NULL)
2878             return (S_ERROR);
2879         break;
2880     }

2882     /* If no string table was created, we have no mergeable sections */
2883     if (mstrtab == NULL)
2884         return (1);

2886     /*
2887      * This routine has to make 3 passes:
2888      *
2889      * 1) Examine all relocations, insert strings from relocations
2890      *    to the mergeable input sections into the string table.
2891      * 2) Modify the relocation values to be correct for the
2892      *    new merged section.
2893      * 3) Modify the symbols used by the relocations to reference
2894      *    the new section.
2895      *
2896      * These passes cannot be combined:

```



```

2897 * - The string table code works in two passes, and all
2898 * strings have to be loaded in pass one before the
2899 * offset of any strings can be determined.
2900 * - Multiple relocations reference a single symbol, so the
2901 * symbol cannot be modified until all relocations are
2902 * fixed.
2903 *
2904 * The number of relocations related to section merging is usually
2905 * a mere fraction of the overall active and output relocation lists,
2906 * and the number of symbols is usually a fraction of the number
2907 * of related relocations. We therefore build APlists for the
2908 * relocations and symbols in the first pass, and then use those
2909 * lists to accelerate the operation of pass 2 and 3.
2910 *
2911 * Reinitialize the lists to a completely empty state.
2912 */
2913 apolist_reset(*rel_alpp);
2914 apolist_reset(*sym_alpp);
2915
2916 /*
2917 * Pass 1:
2918 *
2919 * Every relocation related to this output section (and the input
2920 * sections that make it up) is found in either the active, or the
2921 * output relocation list, depending on whether the relocation is to
2922 * be processed by this invocation of the linker, or inserted into the
2923 * output object.
2924 *
2925 * Build lists of relocations and symbols that will need modification,
2926 * and insert the strings they reference into the mstrtab string table.
2927 */
2928 if (strmerge_pass1(ofl, osp, mstrtab, rel_alpp, sym_alpp,
2929 &ofl->oofl_actrels) == 0)
2930 goto return_s_error;
2931 if (strmerge_pass1(ofl, osp, mstrtab, rel_alpp, sym_alpp,
2932 &ofl->oofl_outrels) == 0)
2933 goto return_s_error;
2934
2935 /*
2936 * Get the size of the new input section. Requesting the
2937 * string table size "cooks" the table, and finalizes its contents.
2938 */
2939 data_size = st_getstrtab_sz(mstrtab);
2940
2941 /* Create a new input section to hold the merged strings */
2942 if (new_section_from_template(ofl, isp, data_size,
2943 &mstrsec, &mstr_shdr, &mstr_data) == S_ERROR)
2944 goto return_s_error;
2945 mstrsec->is_flags |= FLG_IS_GNSTRMRG;
2946
2947 /*
2948 * Allocate a data buffer for the new input section.
2949 * Then, associate the buffer with the string table descriptor.
2950 */
2951 if ((mstr_data->d_buf = libld_malloc(data_size)) == NULL)
2952 goto return_s_error;
2953 if (st_setstrbuf(mstrtab, mstr_data->d_buf, data_size) == -1)
2954 goto return_s_error;
2955
2956 /* Add the new section to the output image */
2957 if (ld_place_section(ofl, mstrsec, NULL, osp->os_idendndx, NULL) ==
2958 (Os_desc *)S_ERROR)
2959 goto return_s_error;
2960
2961 /*
2962 * Pass 2:

```

```

2963 *
2964 * Revisit the relocation descriptors with STT_SECTION symbols
2965 * that were saved by the first pass. Update each relocation
2966 * record so that the offset it contains is for the new section
2967 * instead of the original.
2968 */
2969 for (APLIST_TRAVERSE(*rel_alpp, idx, rsp)) {
2970     const char *name;
2971
2972     /* Put the string into the merged string table */
2973     name = strmerge_get_reloc_str(ofl, rsp);
2974     st_setstring_status = st_setstring(mstrtab, name, &stoff);
2975     if (st_setstring_status == -1) {
2976         /*
2977          * A failure to insert at this point means that
2978          * something is corrupt. This isn't a resource issue.
2979          */
2980         assert(st_setstring_status != -1);
2981         goto return_s_error;
2982     }
2983
2984     /*
2985     * Alter the relocation to access the string at the
2986     * new offset in our new string table.
2987     *
2988     * For SHT_RELA platforms, it suffices to simply
2989     * update the rel_raddend field of the relocation.
2990     *
2991     * For SHT_REL platforms, the new "addend" value
2992     * needs to be written at the address being relocated.
2993     * However, we can't alter the input sections which
2994     * are mapped readonly, and the output image has not
2995     * been created yet. So, we defer this operation,
2996     * using the rel_raddend field of the relocation
2997     * which is normally 0 on a REL platform, to pass the
2998     * new "addend" value to ld_perform_outreloc() or
2999     * ld_do_activerelocs(). The FLG_REL_NADDEND flag
3000     * tells them that this is the case.
3001     */
3002     if ((rsp->rel_flags & FLG_REL_RELA) == 0) /* REL */
3003         rsp->rel_flags |= FLG_REL_NADDEND;
3004     rsp->rel_raddend = (Sxword)stoff;
3005
3006     /*
3007     * Generate a symbol name string for STT_SECTION symbols
3008     * that might reference our merged section. This shows up
3009     * in debug output and helps show how the relocation has
3010     * changed from its original input section to our merged one.
3011     */
3012     if (ld_stt_section_sym_name(mstrsec) == NULL)
3013         goto return_s_error;
3014 }
3015
3016 /*
3017 * Pass 3:
3018 *
3019 * Modify the symbols referenced by the relocation descriptors
3020 * so that they reference the new input section containing the
3021 * merged strings instead of the original input sections.
3022 */
3023 for (APLIST_TRAVERSE(*sym_alpp, idx, sdp)) {
3024     /*
3025     * If we've already processed this symbol, don't do it
3026     * twice. strmerge_pass1() uses a heuristic (relocations to
3027     * the same symbol clump together) to avoid inserting a
3028     * given symbol more than once, but repeat symbols in

```

```

3029     * the list can occur.
3030     */
3031     if ((sdp->sd_isc->is_flags & FLG_IS_INSTRMRG) == 0)
3032         continue;

3034     if (ELF_ST_TYPE(sdp->sd_sym->st_info) != STT_SECTION) {
3035         /*
3036          * This is not an STT_SECTION symbol, so its
3037          * value is the offset of the string within the
3038          * input section. Update the address to reflect
3039          * the address in our new merged section.
3040          */
3041         const char *name = sdp->sd_sym->st_value +
3042             (char *)sdp->sd_isc->is_indata->d_buf;

3044         st_setstring_status =
3045             st_setstring(mstrtab, name, &stoff);
3046         if (st_setstring_status == -1) {
3047             /*
3048              * A failure to insert at this point means
3049              * something is corrupt. This isn't a
3050              * resource issue.
3051              */
3052             assert(st_setstring_status != -1);
3053             goto return_s_error;
3054         }

3056         if (ld_sym_copy(sdp) == S_ERROR)
3057             goto return_s_error;
3058         sdp->sd_sym->st_value = (Word)stoff;
3059     }

3061     /* Redirect the symbol to our new merged section */
3062     sdp->sd_isc = mstrsec;
3063 }

3065 /*
3066  * There are no references left to the original input string sections.
3067  * Mark them as discarded so they don't go into the output image.
3068  * At the same time, add up the sizes of the replaced sections.
3069  */
3070 data_size = 0;
3071 for (APLIST_TRAVERSE(osp->os_mstrisdescs, idx, isp) {
3072     if (isp->is_flags & (FLG_IS_DISCARD | FLG_IS_GNSTRMRG))
3073         continue;

3075     data_size += isp->is_indata->d_size;

3077     isp->is_flags |= FLG_IS_DISCARD;
3078     DBG_CALL(DBG_sec_discarded(ofl->ofl_lml, isp, mstrsec));
3079 }

3081 /* Report how much space we saved in the output section */
3082 DBG_CALL(DBG_sec_genstr_compress(ofl->ofl_lml, osp->os_name, data_size,
3083     mstr_data->d_size));

3085     st_destroy(mstrtab);
3086     return (1);

3088 return_s_error:
3089     st_destroy(mstrtab);
3090     return (S_ERROR);
3091 }

3093 /*
3094  * Update a data buffers size. A number of sections have to be created, and

```

```

3095     * the sections header contributes to the size of the eventual section. Thus,
3096     * a section may be created, and once all associated sections have been created,
3097     * we return to establish the required section size.
3098     */
3099     inline static void
3100     update_data_size(Os_desc *osp, ulong_t cnt)
3101     {
3102         Is_desc         *isec = ld_os_first_isdesc(osp);
3103         Elf_Data         *data = isec->is_indata;
3104         Shdr             *shdr = osp->os_shdr;
3105         size_t           size = cnt * shdr->sh_entsize;

3107         shdr->sh_size = (Xword)size;
3108         data->d_size = size;
3109     }

3111 /*
3112  * The following sections are built after all input file processing and symbol
3113  * validation has been carried out. The order is important (because the
3114  * addition of a section adds a new symbol there is a chicken and egg problem
3115  * of maintaining the appropriate counts). By maintaining a known order the
3116  * individual routines can compensate for later, known, additions.
3117  */
3118     uintptr_t
3119     ld_make_sections(Of1_desc *ofl)
3120     {
3121         ofl_flag_t       flags = ofl->ofl_flags;
3122         Sg_desc          *sgp;

3124         /*
3125          * Generate any special sections.
3126          */
3127         if (flags & FLG_OF_ADDVERS)
3128             if (make_comment(ofl) == S_ERROR)
3129                 return (S_ERROR);

3131         if (make_interp(ofl) == S_ERROR)
3132             return (S_ERROR);

3134         /*
3135          * Create a capabilities section if required.
3136          */
3137         if (make_cap(ofl, SHT_SUNW_cap, MSG_ORIG(MSG_SCN_SUNWCAP),
3138             ld_target_id.id_cap) == S_ERROR)
3139             return (S_ERROR);

3141         /*
3142          * Create any init/fini array sections.
3143          */
3144         if (make_array(ofl, SHT_INIT_ARRAY, MSG_ORIG(MSG_SCN_INITARRAY),
3145             ofl->ofl_initarray) == S_ERROR)
3146             return (S_ERROR);

3148         if (make_array(ofl, SHT_FINI_ARRAY, MSG_ORIG(MSG_SCN_FINIARRAY),
3149             ofl->ofl_finiarray) == S_ERROR)
3150             return (S_ERROR);

3152         if (make_array(ofl, SHT_PREINIT_ARRAY, MSG_ORIG(MSG_SCN_PREINITARRAY),
3153             ofl->ofl_preiarray) == S_ERROR)
3154             return (S_ERROR);

3156         /*
3157          * Make the .plt section. This occurs after any other relocation
3158          * sections are generated (see reloc_init()) to ensure that the
3159          * associated relocation section is after all the other relocation
3160          * sections.

```

```

3161  */
3162  if ((ofl->ofl_pltcnt) || (ofl->ofl_pltpad))
3163      if (make_plt(ofl) == S_ERROR)
3164          return (S_ERROR);

3166  /*
3167  * Determine whether any sections or files are not referenced. Under
3168  * -Dunused a diagnostic for any unused components is generated, under
3169  * -zignore the component is removed from the final output.
3170  */
3171  if (DBG_ENABLED || (ofl->ofl_flags1 & FLG_OF1_IGNPRC)) {
3172      if (ignore_section_processing(ofl) == S_ERROR)
3173          return (S_ERROR);
3174  }

3176  /*
3177  * If we have detected a situation in which previously placed
3178  * output sections may have been discarded, perform the necessary
3179  * readjustment.
3180  */
3181  if (ofl->ofl_flags & FLG_OF_ADJOSCNT)
3182      adjust_os_count(ofl);

3184  /*
3185  * Do any of the output sections contain input sections that
3186  * are candidates for string table merging? For each such case,
3187  * we create a replacement section, insert it, and discard the
3188  * originals.
3189  *
3190  * rel_alpp and sym_alpp are used by ld_make_strmerge()
3191  * for its internal processing. We are responsible for the
3192  * initialization and cleanup, and ld_make_strmerge() handles the rest.
3193  * This allows us to reuse a single pair of memory buffers, allocated
3194  * for this processing, for all the output sections.
3195  */
3196  if ((ofl->ofl_flags1 & FLG_OF1_NCSTTAB) == 0) {
3197      int     error_seen = 0;
3198      APLIST *rel_alpp = NULL;
3199      APLIST *sym_alpp = NULL;
3200      Aliste idx1;

3202      for (APLIST_TRAVERSE(ofl->ofl_segs, idx1, sgp)) {
3203          Os_desc *osp;
3204          Aliste idx2;

3206          for (APLIST_TRAVERSE(sgp->sg_osdescs, idx2, osp))
3207              if ((osp->os_mstrisdescs != NULL) &&
3208                  (ld_make_strmerge(ofl, osp,
3209                                &rel_alpp, &sym_alpp) ==
3210                   S_ERROR)) {
3211                  error_seen = 1;
3212                  break;
3213              }
3214      }
3215      if (rel_alpp != NULL)
3216          libld_free(rel_alpp);
3217      if (sym_alpp != NULL)
3218          libld_free(sym_alpp);
3219      if (error_seen != 0)
3220          return (S_ERROR);
3221  }

3223  /*
3224  * Add any necessary versioning information.
3225  */
3226  if (!(flags & FLG_OF_NOVERSEC)) {

```

```

3227      if ((flags & FLG_OF_VERNEED) &&
3228          (make_verneed(ofl) == S_ERROR))
3229          return (S_ERROR);
3230      if ((flags & FLG_OF_VERDEF) &&
3231          (make_verdef(ofl) == S_ERROR))
3232          return (S_ERROR);
3233      if ((flags & (FLG_OF_VERNEED | FLG_OF_VERDEF)) &&
3234          ((ofl->ofl_osversym = make_sym_sec(ofl,
3235                                           MSG_ORIG(MSG_SCN_SUNWVERSYM),
3236                                           SHT_SUNW_versym,
3237                                           ld_targ.t_id.id_version)) == (Os_desc*)S_ERROR))
3238          return (S_ERROR);
3239  }

3240  /*
3241  * Create a syminfo section if necessary.
3242  */
3243  if (flags & FLG_OF_SYMINFO) {
3244      if ((ofl->ofl_ossyminfo = make_sym_sec(ofl,
3245                                           MSG_ORIG(MSG_SCN_SUNWSYMINFO),
3246                                           SHT_SUNW_syminfo,
3247                                           ld_targ.t_id.id_syminfo)) == (Os_desc *)S_ERROR)
3248          return (S_ERROR);
3249  }

3250  if (flags & FLG_OF_COMREL) {
3251      /*
3252       * If -zcombreloc is enabled then all relocations (except for
3253       * the PLT's) are coalesced into a single relocation section.
3254       */
3255      if (ofl->ofl_relocnt) {
3256          if (make_reloc(ofl, NULL) == S_ERROR)
3257              return (S_ERROR);
3258      }
3259  } else {
3260      Aliste idx1;

3262      /*
3263       * Create the required output relocation sections. Note, new
3264       * sections may be added to the section list that is being
3265       * traversed. These insertions can move the elements of the
3266       * Alist such that a section descriptor is re-read. Recursion
3267       * is prevented by maintaining a previous section pointer and
3268       * insuring that this pointer isn't re-examined.
3269       */
3270      for (APLIST_TRAVERSE(ofl->ofl_segs, idx1, sgp)) {
3271          Os_desc *osp, *posp = 0;
3272          Aliste idx2;

3274          for (APLIST_TRAVERSE(sgp->sg_osdescs, idx2, osp)) {
3275              if ((osp != posp) && osp->os_szoutrels &&
3276                  (osp != ofl->ofl_osplt)) {
3277                  if (make_reloc(ofl, osp) == S_ERROR)
3278                      return (S_ERROR);
3279              }
3280              posp = osp;
3281          }
3282      }

3284      /*
3285       * If we're not building a combined relocation section, then
3286       * build a .rel[a] section as required.
3287       */
3288      if (ofl->ofl_relocrelsz) {
3289          if (make_reloc(ofl, NULL) == S_ERROR)
3290              return (S_ERROR);
3291      }
3292  }

```

```

3294  /*
3295  * The PLT relocations are always in their own section, and we try to
3296  * keep them at the end of the PLT table. We do this to keep the hot
3297  * "data" PLT's at the head of the table nearer the .dynsym & .hash.
3298  */
3299  if (ofl->ofl_osplt && ofl->ofl_relocpltsz) {
3300      if (make_reloc(ofl, ofl->ofl_osplt) == S_ERROR)
3301          return (S_ERROR);
3302  }

3304  /*
3305  * Finally build the symbol and section header sections.
3306  */
3307  if (flags & FLG_OF_DYNAMIC) {
3308      if (make_dynamic(ofl) == S_ERROR)
3309          return (S_ERROR);

3311      /*
3312      * A number of sections aren't necessary within a relocatable
3313      * object, even if -dy has been used.
3314      */
3315      if (!(flags & FLG_OF_RELOBJ)) {
3316          if (make_hash(ofl) == S_ERROR)
3317              return (S_ERROR);
3318          if (make_dynstr(ofl) == S_ERROR)
3319              return (S_ERROR);
3320          if (make_dynsym(ofl) == S_ERROR)
3321              return (S_ERROR);
3322          if (ld_unwind_make_hdr(ofl) == S_ERROR)
3323              return (S_ERROR);
3324          if (make_dynsort(ofl) == S_ERROR)
3325              return (S_ERROR);
3326      }
3327  }

3329  if (!(flags & FLG_OF_STRIP) || (flags & FLG_OF_RELOBJ) ||
3330      ((flags & FLG_OF_STATIC) && ofl->ofl_osversym)) {
3331      /*
3332      * Do we need to make a SHT_SYMTAB_SHNDX section
3333      * for the dynsym. If so - do it now.
3334      */
3335      if (ofl->ofl_osdynsym &&
3336          ((ofl->ofl_shdrCNT + 3) >= SHN_LORESERVE)) {
3337          if (make_dynsym_shndx(ofl) == S_ERROR)
3338              return (S_ERROR);
3339      }

3341      if (make_strtab(ofl) == S_ERROR)
3342          return (S_ERROR);
3343      if (make_symtab(ofl) == S_ERROR)
3344          return (S_ERROR);
3345  } else {
3346      /*
3347      * Do we need to make a SHT_SYMTAB_SHNDX section
3348      * for the dynsym. If so - do it now.
3349      */
3350      if (ofl->ofl_osdynsym &&
3351          ((ofl->ofl_shdrCNT + 1) >= SHN_LORESERVE)) {
3352          if (make_dynsym_shndx(ofl) == S_ERROR)
3353              return (S_ERROR);
3354      }
3355  }

3357  if (make_shstrtab(ofl) == S_ERROR)
3358      return (S_ERROR);

```

```

3360  /*
3361  * Now that we've created all output sections, adjust the size of the
3362  * SHT_SUNW_versym and SHT_SUNW_syminfo section, which are dependent on
3363  * the associated symbol table sizes.
3364  */
3365  if (ofl->ofl_osversym || ofl->ofl_ossyminfo) {
3366      ulong_t cnt;
3367      Is_desc *isp;
3368      Os_desc *osp;

3370      if (OFL_IS_STATIC_OBJ(ofl))
3371          osp = ofl->ofl_ossymtab;
3372      else
3373          osp = ofl->ofl_osdynsym;

3375      isp = ld_os_first_isdesc(osp);
3376      cnt = (isp->is_shdr->sh_size / isp->is_shdr->sh_entsize);

3378      if (ofl->ofl_osversym)
3379          update_data_size(ofl->ofl_osversym, cnt);

3381      if (ofl->ofl_ossyminfo)
3382          update_data_size(ofl->ofl_ossyminfo, cnt);
3383  }

3385  /*
3386  * Now that we've created all output sections, adjust the size of the
3387  * SHT_SUNW_capinfo, which is dependent on the associated symbol table
3388  * size.
3389  */
3390  if (ofl->ofl_oscapiinfo) {
3391      ulong_t cnt;

3393      /*
3394      * Symbol capabilities symbols are placed directly after the
3395      * STT_FILE symbol, section symbols, and any register symbols.
3396      * Effectively these are the first of any series of demoted
3397      * (scoped) symbols.
3398      */
3399      if (OFL_IS_STATIC_OBJ(ofl))
3400          cnt = SYMTAB_ALL_CNT(ofl);
3401      else
3402          cnt = DYNYSM_ALL_CNT(ofl);

3404      update_data_size(ofl->ofl_oscapiinfo, cnt);
3405  }
3406  return (1);
3407 }

3409 /*
3410 * Build an additional data section - used to back OBJT symbol definitions
3411 * added with a mapfile.
3412 */
3413 Is_desc *
3414 ld_make_data(Of1_desc *ofl, size_t size)
3415 {
3416     Shdr *shdr;
3417     Elf_Data *data;
3418     Is_desc *isec;

3420     if (new_section(ofl, SHT_PROGBITS, MSG_ORIG(MSG_SCN_DATA), 0,
3421         &isec, &shdr, &data) == S_ERROR)
3422         return ((Is_desc *)S_ERROR);

3424     data->d_size = size;

```

```

3425     shdr->sh_size = (Xword)size;
3426     shdr->sh_flags |= SHF_WRITE;

3428     if (aplist_append(&ofl->ofl_mapdata, isec, AL_CNT_OFL_MAPSECS) == NULL)
3429         return ((Is_desc *)S_ERROR);

3431     return (isec);
3432 }

3434 /*
3435  * Build an additional text section - used to back FUNC symbol definitions
3436  * added with a mapfile.
3437  */
3438 Is_desc *
3439 ld_make_text(Of1_desc *ofl, size_t size)
3440 {
3441     Shdr      *shdr;
3442     Elf_Data  *data;
3443     Is_desc   *isec;

3445     /*
3446      * Insure the size is sufficient to contain the minimum return
3447      * instruction.
3448      */
3449     if (size < ld_targ.t_nf.nf_size)
3450         size = ld_targ.t_nf.nf_size;

3452     if (new_section(ofl, SHT_PROGBITS, MSG_ORIG(MSG_SCN_TEXT), 0,
3453                    &isec, &shdr, &data) == S_ERROR)
3454         return ((Is_desc *)S_ERROR);

3456     data->d_size = size;
3457     shdr->sh_size = (Xword)size;
3458     shdr->sh_flags |= SHF_EXECINSTR;

3460     /*
3461      * Fill the buffer with the appropriate return instruction.
3462      * Note that there is no need to swap bytes on a non-native,
3463      * link, as the data being copied is given in bytes.
3464      */
3465     if ((data->d_buf = libld_calloc(size, 1)) == NULL)
3466         return ((Is_desc *)S_ERROR);
3467     (void) memcpy(data->d_buf, ld_targ.t_nf.nf_template,
3468                 ld_targ.t_nf.nf_size);

3470     /*
3471      * If size was larger than required, and the target supplies
3472      * a fill function, use it to fill the balance. If there is no
3473      * fill function, we accept the 0-fill supplied by libld_calloc().
3474      */
3475     if ((ld_targ.t_ff.ff_execfill != NULL) && (size > ld_targ.t_nf.nf_size))
3476         ld_targ.t_ff.ff_execfill(data->d_buf, ld_targ.t_nf.nf_size,
3477                                 size - ld_targ.t_nf.nf_size);

3479     if (aplist_append(&ofl->ofl_maptext, isec, AL_CNT_OFL_MAPSECS) == NULL)
3480         return ((Is_desc *)S_ERROR);

3482     return (isec);
3483 }

3485 void
3486 ld_comdat_validate(Of1_desc *ofl, Ifl_desc *ifl)
3487 {
3488     int i;

3490     for (i = 0; i < ifl->ifl_shnum; i++) {

```

```

3491     Is_desc *isp = ifl->ifl_isdesc[i];
3492     int types = 0;
3493     char buf[1024] = "";
3494     Group_desc *gr = NULL;

3496     if ((isp == NULL) || (isp->is_flags & FLG_IS_COMDAT) == 0)
3497         continue;

3499     if (isp->is_shdr->sh_type == SHT_SUNW_COMDAT) {
3500         types++;
3501         (void) strcpy(buf, MSG_ORIG(MSG_STR_SUNW_COMDAT),
3502                     sizeof (buf));
3503     }

3505     if (strcmp(MSG_ORIG(MSG_SCN_GNU_LINKONCE), isp->is_name,
3506              MSG_SCN_GNU_LINKONCE_SIZE) == 0) {
3507         types++;
3508         if (types > 1)
3509             (void) strcat(buf, ", ", sizeof (buf));
3510         (void) strcat(buf, MSG_ORIG(MSG_SCN_GNU_LINKONCE),
3511                     sizeof (buf));
3512     }

3514     if ((isp->is_shdr->sh_flags & SHF_GROUP) &&
3515         ((gr = ld_get_group(ofl, isp)) != NULL) &&
3516         (gr->gd_data[0] & GRP_COMDAT)) {
3517         types++;
3518         if (types > 1)
3519             (void) strcat(buf, ", ", sizeof (buf));
3520         (void) strcat(buf, MSG_ORIG(MSG_STR_GROUP),
3521                     sizeof (buf));
3522     }

3524     if (types > 1)
3525         ld_eprintf(ofl, ERR_FATAL,
3526                  MSG_INTL(MSG_SCN_MULTICOMDAT), ifl->ifl_name,
3527                  EC_WORD(isp->is_scndx), isp->is_name, buf);
3528 }
3529 }

```

```

*****
118329 Thu Jun 30 22:01:34 2016
new/usr/src/cmd/sgs/libld/common/update.c
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
_____unchanged_portion_omitted_____

2058 /*
2059 * Build the dynamic section.
2060 *
2061 * This routine must be maintained in parallel with make_dynamic()
2062 * in sections.c
2063 */
2064 static int
2065 update_odynamic(Of1_desc *of1)
2066 {
2067     Aliste      idx;
2068     Ifl_desc    *ifl;
2069     Sym_desc    *sdp;
2070     Shdr        *shdr;
2071     Dyn         *_dyn = (Dyn *)of1->ofl_osdynamic->os_outdata->d_buf;
2072     Dyn         *dyn;
2073     Os_desc     *symosp, *strospp;
2074     Str_tbl     *strtbl;
2075     size_t      stoff;
2076     ofl_flag_t  flags = of1->ofl_flags;
2077     int         not_relobj = !(flags & FLG_OF_RELOBJ);
2078     Word        cnt;

2080 /*
2081 * Relocatable objects can be built with -r and -dy to trigger the
2082 * creation of a .dynamic section. This model is used to create kernel
2083 * device drivers. The .dynamic section provides a subset of userland
2084 * .dynamic entries, typically entries such as DT_NEEDED and DT_RUNPATH.
2085 *
2086 * Within a dynamic object, any .dynamic string references are to the
2087 * .dynstr table. Within a relocatable object, these strings can reside
2088 * within the .strtab.
2089 */
2090 if (OFL_IS_STATIC_OBJ(of1)) {
2091     symosp = of1->ofl_ossymtab;
2092     strospp = of1->ofl_osstrtab;
2093     strtbl = of1->ofl_strtab;
2094 } else {
2095     symosp = of1->ofl_osdynsym;
2096     strospp = of1->ofl_osdynstr;
2097     strtbl = of1->ofl_dynstrtab;
2098 }

2100 /* LINTED */
2101 of1->ofl_osdynamic->os_shdr->sh_link = (Word)elf_ndxscn(strospp->os_scn);

2103 dyn = _dyn;

2105 for (APLIST_TRAVERSE(of1->ofl_sos, idx, ifl)) {
2106     if ((ifl->ifl_flags &
2107         (FLG_IF_IGNORE | FLG_IF_DEPREQD)) == FLG_IF_IGNORE)
2108         continue;

2110 /*
2111 * Create and set up the DT_POSFLAG_1 entry here if required.
2112 */
2113 if ((ifl->ifl_flags & MSK_IF_POSFLAG1) &&

```

```

2114     (ifl->ifl_flags & FLG_IF_NEEDED) && not_relobj) {
2115         dyn->d_tag = DT_POSFLAG_1;
2116         if (ifl->ifl_flags & FLG_IF_LAZYLD)
2117             dyn->d_un.d_val = DF_PL_LAZYLOAD;
2118         if (ifl->ifl_flags & FLG_IF_GRPFRM)
2119             dyn->d_un.d_val |= DF_PL_GROUPEM;
2120         if (ifl->ifl_flags & FLG_IF_DEFERRED)
2121             dyn->d_un.d_val |= DF_PL_DEFERRED;
2122         dyn++;
2123     }

2125 if (ifl->ifl_flags & (FLG_IF_NEEDED | FLG_IF_NEEDSTR))
2126     dyn->d_tag = DT_NEEDED;
2127 else
2128     continue;

2130 (void) st_setstring(strtbl, ifl->ifl_soname, &stoff);
2131 dyn->d_un.d_val = stoff;
2132 /* LINTED */
2133 ifl->ifl_neededndx = (Half)(((uintptr_t)dyn - (uintptr_t)_dyn) /
2134                             sizeof (Dyn));
2135 dyn++;
2136 }

2138 if (not_relobj) {
2139     if (of1->ofl_dtsfltrs != NULL) {
2140         Dfltr_desc *dftp;

2142         for (ALIST_TRAVERSE(of1->ofl_dtsfltrs, idx, dftp)) {
2143             if (dftp->dft_flag == FLG_SY_AUXFLTR)
2144                 dyn->d_tag = DT_SUNW_AUXILIARY;
2145             else
2146                 dyn->d_tag = DT_SUNW_FILTER;

2148             (void) st_setstring(strtbl, dftp->dft_str,
2149                               &stoff);
2150             dyn->d_un.d_val = stoff;
2151             dftp->dft_ndx = (Half)(((uintptr_t)dyn -
2152                                   (uintptr_t)_dyn) / sizeof (Dyn));
2153             dyn++;
2154         }
2155     }
2156 if (((sdp = ld_sym_find(MSG_ORIG(MSG_SYM_INIT_U),
2157                         SYM_NOHASH, 0, of1)) != NULL) &&
2158     (sdp->sd_ref == REF_REL_NEED) &&
2159     (sdp->sd_sym->st_shndx != SHN_UNDEF)) {
2160     dyn->d_tag = DT_INIT;
2161     dyn->d_un.d_ptr = sdp->sd_sym->st_value;
2162     dyn++;
2163 }
2164 if (((sdp = ld_sym_find(MSG_ORIG(MSG_SYM_FINI_U),
2165                         SYM_NOHASH, 0, of1)) != NULL) &&
2166     (sdp->sd_ref == REF_REL_NEED) &&
2167     (sdp->sd_sym->st_shndx != SHN_UNDEF)) {
2168     dyn->d_tag = DT_FINI;
2169     dyn->d_un.d_ptr = sdp->sd_sym->st_value;
2170     dyn++;
2171 }
2172 if (of1->ofl_soname) {
2173     dyn->d_tag = DT_SONAME;
2174     (void) st_setstring(strtbl, of1->ofl_soname, &stoff);
2175     dyn->d_un.d_val = stoff;
2176     dyn++;
2177 }
2178 if (of1->ofl_filtees) {
2179     if (flags & FLG_OF_AUX) {

```

```

2180         dyn->d_tag = DT_AUXILIARY;
2181     } else {
2182         dyn->d_tag = DT_FILTER;
2183     }
2184     (void) st_setstring(strtbl, ofl->ofl_filtees, &stoff);
2185     dyn->d_un.d_val = stoff;
2186     dyn++;
2187 }
2188
2190 if (ofl->ofl_rpath) {
2191     (void) st_setstring(strtbl, ofl->ofl_rpath, &stoff);
2192     dyn->d_tag = DT_RUNPATH;
2193     dyn->d_un.d_val = stoff;
2194     dyn++;
2195     dyn->d_tag = DT_RPATH;
2196     dyn->d_un.d_val = stoff;
2197     dyn++;
2198 }
2200 if (not_relobj) {
2201     Aliste idx;
2202     Sg_desc *sgp;
2203
2204     if (ofl->ofl_config) {
2205         dyn->d_tag = DT_CONFIG;
2206         (void) st_setstring(strtbl, ofl->ofl_config, &stoff);
2207         dyn->d_un.d_val = stoff;
2208         dyn++;
2209     }
2210     if (ofl->ofl_depaudit) {
2211         dyn->d_tag = DT_DEPAUDIT;
2212         (void) st_setstring(strtbl, ofl->ofl_depaudit, &stoff);
2213         dyn->d_un.d_val = stoff;
2214         dyn++;
2215     }
2216     if (ofl->ofl_audit) {
2217         dyn->d_tag = DT_AUDIT;
2218         (void) st_setstring(strtbl, ofl->ofl_audit, &stoff);
2219         dyn->d_un.d_val = stoff;
2220         dyn++;
2221     }
2222
2223     dyn->d_tag = DT_HASH;
2224     dyn->d_un.d_ptr = ofl->ofl_oshash->os_shdr->sh_addr;
2225     dyn++;
2226
2227     shdr = strosp->os_shdr;
2228     dyn->d_tag = DT_STARTAB;
2229     dyn->d_un.d_ptr = shdr->sh_addr;
2230     dyn++;
2231
2232     dyn->d_tag = DT_STRSZ;
2233     dyn->d_un.d_ptr = shdr->sh_size;
2234     dyn++;
2235
2236     /*
2237     * Note, the shdr is set and used in the ofl->ofl_osldynsym case
2238     * that follows.
2239     */
2240     shdr = symosp->os_shdr;
2241     dyn->d_tag = DT_SYMTAB;
2242     dyn->d_un.d_ptr = shdr->sh_addr;
2243     dyn++;
2244
2245     dyn->d_tag = DT_SYMENT;

```

```

2246     dyn->d_un.d_ptr = shdr->sh_entsize;
2247     dyn++;
2248
2249     if (ofl->ofl_osldynsym) {
2250         Shdr *lshdr = ofl->ofl_osldynsym->os_shdr;
2251
2252         /*
2253         * We have arranged for the .SUNW_ldynsym data to be
2254         * immediately in front of the .dynsym data.
2255         * This means that you could start at the top
2256         * of .SUNW_ldynsym and see the data for both tables
2257         * without a break. This is the view we want to
2258         * provide for DT_SUNW_SYMTAB, which is why we
2259         * add the lengths together.
2260         */
2261         dyn->d_tag = DT_SUNW_SYMTAB;
2262         dyn->d_un.d_ptr = lshdr->sh_addr;
2263         dyn++;
2264
2265         dyn->d_tag = DT_SUNW_SYMSZ;
2266         dyn->d_un.d_val = lshdr->sh_size + shdr->sh_size;
2267         dyn++;
2268     }
2269
2270     if (ofl->ofl_osdynsymsort || ofl->ofl_osdyntlssort) {
2271         dyn->d_tag = DT_SUNW_SORTENT;
2272         dyn->d_un.d_val = sizeof (Word);
2273         dyn++;
2274     }
2275
2276     if (ofl->ofl_osdynsymsort) {
2277         shdr = ofl->ofl_osdynsymsort->os_shdr;
2278
2279         dyn->d_tag = DT_SUNW_SYMSORT;
2280         dyn->d_un.d_ptr = shdr->sh_addr;
2281         dyn++;
2282
2283         dyn->d_tag = DT_SUNW_SYMSORTSZ;
2284         dyn->d_un.d_val = shdr->sh_size;
2285         dyn++;
2286     }
2287
2288     if (ofl->ofl_osdyntlssort) {
2289         shdr = ofl->ofl_osdyntlssort->os_shdr;
2290
2291         dyn->d_tag = DT_SUNW_TLSSORT;
2292         dyn->d_un.d_ptr = shdr->sh_addr;
2293         dyn++;
2294
2295         dyn->d_tag = DT_SUNW_TLSSORTSZ;
2296         dyn->d_un.d_val = shdr->sh_size;
2297         dyn++;
2298     }
2299
2300     /*
2301     * Reserve the DT_CHECKSUM entry. Its value will be filled in
2302     * after the complete image is built.
2303     */
2304     dyn->d_tag = DT_CHECKSUM;
2305     ofl->ofl_checksum = &dyn->d_un.d_val;
2306     dyn++;
2307
2308     /*
2309     * Versioning sections: DT_VERDEF and DT_VERNEED.
2310     *
2311     * The Solaris ld does not produce DT_VERSYM, but the GNU ld

```

```

2312     * does, in order to support their style of versioning, which
2313     * differs from ours:
2314     *
2315     *   - The top bit of the 16-bit Versym index is
2316     *     not part of the version, but is interpreted
2317     *     as a "hidden bit".
2318     *
2319     *   - External (SHN_UNDEF) symbols can have non-zero
2320     *     Versym values, which specify versions in
2321     *     referenced objects, via the Verneed section.
2322     *
2323     *   - The vna_other field of the Vernaux structures
2324     *     found in the Verneed section are not zero as
2325     *     with Solaris, but instead contain the version
2326     *     index to be used by Versym indices to reference
2327     *     the given external version.
2328     *
2329     * The Solaris ld, rtdld, and elfdump programs all interpret the
2330     * presence of DT_VERSYM as meaning that GNU versioning rules
2331     * apply to the given file. If DT_VERSYM is not present,
2332     * then Solaris versioning rules apply. If we should ever need
2333     * to change our ld so that it does issue DT_VERSYM, then
2334     * this rule for detecting GNU versioning will no longer work.
2335     * In that case, we will have to invent a way to explicitly
2336     * specify the style of versioning in use, perhaps via a
2337     * new dynamic entry named something like DT_SUNW_VERSIONSTYLE,
2338     * where the d_un.d_val value specifies which style is to be
2339     * used.
2340     */
2341     if ((flags & (FLG_OF_VERDEF | FLG_OF_NOVERSEC)) ==
2342         FLG_OF_VERDEF) {
2343         shdr = ofl->ofl_osverdef->os_shdr;
2344
2345         dyn->d_tag = DT_VERDEF;
2346         dyn->d_un.d_ptr = shdr->sh_addr;
2347         dyn++;
2348         dyn->d_tag = DT_VERDEFNUM;
2349         dyn->d_un.d_ptr = shdr->sh_info;
2350         dyn++;
2351     }
2352     if ((flags & (FLG_OF_VERNEED | FLG_OF_NOVERSEC)) ==
2353         FLG_OF_VERNEED) {
2354         shdr = ofl->ofl_osverneed->os_shdr;
2355
2356         dyn->d_tag = DT_VERNEED;
2357         dyn->d_un.d_ptr = shdr->sh_addr;
2358         dyn++;
2359         dyn->d_tag = DT_VERNEEDNUM;
2360         dyn->d_un.d_ptr = shdr->sh_info;
2361         dyn++;
2362     }
2363
2364     if ((flags & FLG_OF_COMREL) && ofl->ofl_relocrelcnt) {
2365         dyn->d_tag = ld_targ.t_m.m_rel_dt_count;
2366         dyn->d_un.d_val = ofl->ofl_relocrelcnt;
2367         dyn++;
2368     }
2369     if (flags & FLG_OF_TEXTREL) {
2370         /*
2371          * Only the presence of this entry is used in this
2372          * implementation, not the value stored.
2373          */
2374         dyn->d_tag = DT_TEXTREL;
2375         dyn->d_un.d_val = 0;
2376         dyn++;
2377     }

```

```

2379     if (ofl->ofl_osfiniarray) {
2380         shdr = ofl->ofl_osfiniarray->os_shdr;
2381
2382         dyn->d_tag = DT_FINI_ARRAY;
2383         dyn->d_un.d_ptr = shdr->sh_addr;
2384         dyn++;
2385
2386         dyn->d_tag = DT_FINI_ARRAYSZ;
2387         dyn->d_un.d_val = shdr->sh_size;
2388         dyn++;
2389     }
2390
2391     if (ofl->ofl_osinitarray) {
2392         shdr = ofl->ofl_osinitarray->os_shdr;
2393
2394         dyn->d_tag = DT_INIT_ARRAY;
2395         dyn->d_un.d_ptr = shdr->sh_addr;
2396         dyn++;
2397
2398         dyn->d_tag = DT_INIT_ARRAYSZ;
2399         dyn->d_un.d_val = shdr->sh_size;
2400         dyn++;
2401     }
2402
2403     if (ofl->ofl_ospreinitarray) {
2404         shdr = ofl->ofl_ospreinitarray->os_shdr;
2405
2406         dyn->d_tag = DT_PREINIT_ARRAY;
2407         dyn->d_un.d_ptr = shdr->sh_addr;
2408         dyn++;
2409
2410         dyn->d_tag = DT_PREINIT_ARRAYSZ;
2411         dyn->d_un.d_val = shdr->sh_size;
2412         dyn++;
2413     }
2414
2415     if (ofl->ofl_pltcnt) {
2416         shdr = ofl->ofl_osplt->os_relostdesc->os_shdr;
2417
2418         dyn->d_tag = DT_PLTRELSZ;
2419         dyn->d_un.d_ptr = shdr->sh_size;
2420         dyn++;
2421         dyn->d_tag = DT_PLTREL;
2422         dyn->d_un.d_ptr = ld_targ.t_m.m_rel_dt_type;
2423         dyn++;
2424         dyn->d_tag = DT_JMPREL;
2425         dyn->d_un.d_ptr = shdr->sh_addr;
2426         dyn++;
2427     }
2428     if (ofl->ofl_pltpad) {
2429         shdr = ofl->ofl_osplt->os_shdr;
2430
2431         dyn->d_tag = DT_PLTPAD;
2432         if (ofl->ofl_pltcnt) {
2433             dyn->d_un.d_ptr = shdr->sh_addr +
2434                 ld_targ.t_m.m_plt_reservsz +
2435                 ofl->ofl_pltcnt * ld_targ.t_m.m_plt_entsize;
2436         } else
2437             dyn->d_un.d_ptr = shdr->sh_addr;
2438         dyn++;
2439         dyn->d_tag = DT_PLTPADSZ;
2440         dyn->d_un.d_val = ofl->ofl_pltpad *
2441             ld_targ.t_m.m_plt_entsize;
2442         dyn++;
2443     }

```



```

2444     if (ofl->ofl_relocsz) {
2445         shdr = ofl->ofl_osrelhead->os_shdr;

2447         dyn->d_tag = ld_targ.t.m.m_rel_dt_type;
2448         dyn->d_un.d_ptr = shdr->sh_addr;
2449         dyn++;
2450         dyn->d_tag = ld_targ.t.m.m_rel_dt_size;
2451         dyn->d_un.d_ptr = ofl->ofl_relocsz;
2452         dyn++;
2453         dyn->d_tag = ld_targ.t.m.m_rel_dt_ent;
2454         if (shdr->sh_type == SHT_REL)
2455             dyn->d_un.d_ptr = sizeof (Rel);
2456         else
2457             dyn->d_un.d_ptr = sizeof (Rela);
2458         dyn++;
2459     }
2460     if (ofl->ofl_ossyminfo) {
2461         shdr = ofl->ofl_ossyminfo->os_shdr;

2463         dyn->d_tag = DT_SYMINFO;
2464         dyn->d_un.d_ptr = shdr->sh_addr;
2465         dyn++;
2466         dyn->d_tag = DT_SYMINSZ;
2467         dyn->d_un.d_val = shdr->sh_size;
2468         dyn++;
2469         dyn->d_tag = DT_SYMINENT;
2470         dyn->d_un.d_val = sizeof (Syminfo);
2471         dyn++;
2472     }
2473     if (ofl->ofl_osmove) {
2474         shdr = ofl->ofl_osmove->os_shdr;

2476         dyn->d_tag = DT_MOVETAB;
2477         dyn->d_un.d_val = shdr->sh_addr;
2478         dyn++;
2479         dyn->d_tag = DT_MOVESZ;
2480         dyn->d_un.d_val = shdr->sh_size;
2481         dyn++;
2482         dyn->d_tag = DT_MOVEENT;
2483         dyn->d_un.d_val = shdr->sh_entsize;
2484         dyn++;
2485     }
2486     if (ofl->ofl_regsymcnt) {
2487         int ndx;

2489         for (ndx = 0; ndx < ofl->ofl_regsysno; ndx++) {
2490             if ((sdp = ofl->ofl_regsys[ndx]) == NULL)
2491                 continue;

2493             dyn->d_tag = ld_targ.t.m.m_dt_register;
2494             dyn->d_un.d_val = sdp->sd_symndx;
2495             dyn++;
2496         }
2497     }

2499     for (APLIST_TRAVERSE(ofl->ofl_rtlldinfo, idx, sdp)) {
2500         dyn->d_tag = DT_SUNW_RTLDINF;
2501         dyn->d_un.d_ptr = sdp->sd_sym->st_value;
2502         dyn++;
2503     }

2505     if (((sgp = ofl->ofl_osdynamic->os_sgdesc) != NULL) &&
2506         (sgp->sg_phdr.p_flags & PF_W) && ofl->ofl_osinterp) {
2507         dyn->d_tag = DT_DEBUG;
2508         dyn->d_un.d_ptr = 0;
2509         dyn++;

```

```

2510     }

2512     if (ofl->ofl_oscapp) {
2513         dyn->d_tag = DT_SUNW_CAP;
2514         dyn->d_un.d_val = ofl->ofl_oscapp->os_shdr->sh_addr;
2515         dyn++;
2516     }
2517     if (ofl->ofl_oscappinfo) {
2518         dyn->d_tag = DT_SUNW_CAPINFO;
2519         dyn->d_un.d_val = ofl->ofl_oscappinfo->os_shdr->sh_addr;
2520         dyn++;
2521     }
2522     if (ofl->ofl_oscappchain) {
2523         shdr = ofl->ofl_oscappchain->os_shdr;

2525         dyn->d_tag = DT_SUNW_CAPCHAIN;
2526         dyn->d_un.d_val = shdr->sh_addr;
2527         dyn++;
2528         dyn->d_tag = DT_SUNW_CAPCHAINSZ;
2529         dyn->d_un.d_val = shdr->sh_size;
2530         dyn++;
2531         dyn->d_tag = DT_SUNW_CAPCHAINENT;
2532         dyn->d_un.d_val = shdr->sh_entsize;
2533         dyn++;
2534     }

2536     if (ofl->ofl_aslr != 0) {
2537         dyn->d_tag = DT_SUNW_ASLR;
2538         dyn->d_un.d_val = (ofl->ofl_aslr == 1);
2539         dyn++;
2540     }

2542 #endif /* ! codereview */
2543     if (flags & FLG_OF_SYMBOLIC) {
2544         dyn->d_tag = DT_SYMBOLIC;
2545         dyn->d_un.d_val = 0;
2546         dyn++;
2547     }
2548 }

2550     dyn->d_tag = DT_FLAGS;
2551     dyn->d_un.d_val = ofl->ofl_dtflags;
2552     dyn++;

2554     /*
2555     * If -Bdirect was specified, but some NODIRECT symbols were specified
2556     * via a mapfile, or -znodirect was used on the command line, then
2557     * clear the DF_1_DIRECT flag. The resultant object will use per-symbol
2558     * direct bindings rather than be enabled for global direct bindings.
2559     *
2560     * If any no-direct bindings exist within this object, set the
2561     * DF_1_NODIRECT flag. ld(1) recognizes this flag when processing
2562     * dependencies, and performs extra work to ensure that no direct
2563     * bindings are established to the no-direct symbols that exist
2564     * within these dependencies.
2565     */
2566     if (ofl->ofl_flags1 & FLG_OF1_NGLBDIR)
2567         ofl->ofl_dtflags_1 &= ~DF_1_DIRECT;
2568     if (ofl->ofl_flags1 & FLG_OF1_NDIRECT)
2569         ofl->ofl_dtflags_1 |= DF_1_NODIRECT;

2571     dyn->d_tag = DT_FLAGS_1;
2572     dyn->d_un.d_val = ofl->ofl_dtflags_1;
2573     dyn++;

2575     dyn->d_tag = DT_SUNW_STRPAD;

```

```

2576     dyn->d_un.d_val = DYNSTR_EXTRA_PAD;
2577     dyn++;

2579     dyn->d_tag = DT_SUNW_LDMACH;
2580     dyn->d_un.d_val = ld_sunw_ldmach();
2581     dyn++;

2583     (*ld_targ.t_mr.mr_mach_update_odynamic)(ofl, &dyn);

2585     for (cnt = 1 + DYNAMIC_EXTRAELTS; cnt--; dyn++) {
2586         dyn->d_tag = DT_NULL;
2587         dyn->d_un.d_val = 0;
2588     }

2590     /*
2591     * Ensure that we wrote the right number of entries. If not, we either
2592     * miscounted in make_dynamic(), or we did something wrong in this
2593     * function.
2594     */
2595     assert((ofl->ofl_osdynamic->os_shdr->sh_size /
2596            ofl->ofl_osdynamic->os_shdr->sh_entsize) ==
2597            ((uintptr_t)dyn - (uintptr_t)_dyn) / sizeof (*dyn));

2599     return (1);
2600 }

2602 /*
2603  * Build the version definition section
2604  */
2605 static int
2606 update_overdef(Of1_desc *ofl)
2607 {
2608     Aliste      idx1;
2609     Ver_desc    *vdp, *_vdp;
2610     Verdef      *vdf, *_vdf;
2611     int         num = 0;
2612     Os_desc     *stros;
2613     Str_tbl     *strtbl;

2615     /*
2616     * Determine which string table to use.
2617     */
2618     if (OFL_IS_STATIC_OBJ(ofl)) {
2619         strtbl = ofl->ofl_strtab;
2620         strosp = ofl->ofl_osstrtab;
2621     } else {
2622         strtbl = ofl->ofl_dynstrtab;
2623         strosp = ofl->ofl_osdynstr;
2624     }

2626     /*
2627     * Traverse the version descriptors and update the version structures
2628     * to point to the dynstr name in preparation for building the version
2629     * section structure.
2630     */
2631     for (APLIST_TRAVERSE(ofl->ofl_verdesc, idx1, vdp)) {
2632         Sym_desc  *sdp;

2634         if (vdp->vd_flags & VER_FLG_BASE) {
2635             const char *name = vdp->vd_name;
2636             size_t      stoff;

2638             /*
2639             * Create a new string table entry to represent the base
2640             * version name (there is no corresponding symbol for
2641             * this).

```

```

2642         /*
2643         (void) st_setstring(strtbl, name, &stoff);
2644         /* LINTED */
2645         vdp->vd_name = (const char *)stoff;
2646     } else {
2647         sdp = ld_sym_find(vdp->vd_name, vdp->vd_hash, 0, ofl);
2648         /* LINTED */
2649         vdp->vd_name = (const char *)
2650             (uintptr_t)sdp->sd_sym->st_name;
2651     }
2652 }

2654 _vdf = vdf = (Verdef *)ofl->ofl_osverdef->os_outdata->d_buf;

2656     /*
2657     * Traverse the version descriptors and update the version section to
2658     * reflect each version and its associated dependencies.
2659     */
2660     for (APLIST_TRAVERSE(ofl->ofl_verdesc, idx1, vdp)) {
2661         Aliste      idx2;
2662         Half        cnt = 1;
2663         Verdaux     *vdap, *_vdap;

2665         _vdap = vdap = (Verdaux *) (vdf + 1);

2667         vdf->vd_version = VER_DEF_CURRENT;
2668         vdf->vd_flags   = vdp->vd_flags & MSK_VER_USER;
2669         vdf->vd_ndx     = vdp->vd_ndx;
2670         vdf->vd_hash    = vdp->vd_hash;

2672         /* LINTED */
2673         vdap->vda_name = (uintptr_t)vdp->vd_name;
2674         vdap++;
2675         /* LINTED */
2676         _vdap->vda_next = (Word)((uintptr_t)vdap - (uintptr_t)_vdap);

2678         /*
2679         * Traverse this versions dependency list generating the
2680         * appropriate version dependency entries.
2681         */
2682         for (APLIST_TRAVERSE(vdp->vd_deps, idx2, _vdp)) {
2683             /* LINTED */
2684             vdap->vda_name = (uintptr_t)_vdp->vd_name;
2685             _vdap = vdap;
2686             vdap++, cnt++;
2687             /* LINTED */
2688             _vdap->vda_next = (Word)((uintptr_t)vdap -
2689                                   (uintptr_t)_vdap);
2690         }
2691         _vdap->vda_next = 0;

2693         /*
2694         * Record the versions auxiliary array offset and the associated
2695         * dependency count.
2696         */
2697         /* LINTED */
2698         vdf->vd_aux = (Word)((uintptr_t)(vdf + 1) - (uintptr_t)vdf);
2699         vdf->vd_cnt = cnt;

2701         /*
2702         * Record the next versions offset and update the version
2703         * pointer. Remember the previous version offset as the very
2704         * last structures next pointer should be null.
2705         */
2706         _vdf = vdf;
2707         vdf = (Verdef *)vdap, num++;

```

```

2708     /* LINTED */
2709     _vdf->vd_next = (Word)((uintptr_t)vdf - (uintptr_t)_vdf);
2710 }
2711 _vdf->vd_next = 0;

2713 /*
2714  * Record the string table association with the version definition
2715  * section, and the symbol table associated with the version symbol
2716  * table (the actual contents of the version symbol table are filled
2717  * in during symbol update).
2718  */
2719 /* LINTED */
2720 ofl->ofl_osverdef->os_shdr->sh_link = (Word)elf_ndxscn(strosp->os_scn);

2722 /*
2723  * The version definition sections 'info' field is used to indicate the
2724  * number of entries in this section.
2725  */
2726 ofl->ofl_osverdef->os_shdr->sh_info = num;

2728 return (1);
2729 }

2731 /*
2732  * Finish the version symbol index section
2733  */
2734 static void
2735 update_oversym(Of1_desc *ofl)
2736 {
2737     Os_desc      *osp;

2739     /*
2740      * Record the symbol table associated with the version symbol table.
2741      * The contents of the version symbol table are filled in during
2742      * symbol update.
2743      */
2744     if (OFL_IS_STATIC_OBJ(ofl))
2745         osp = ofl->ofl_ossymtab;
2746     else
2747         osp = ofl->ofl_osdynsym;

2749     /* LINTED */
2750     ofl->ofl_osversym->os_shdr->sh_link = (Word)elf_ndxscn(osp->os_scn);
2751 }

2753 /*
2754  * Build the version needed section
2755  */
2756 static int
2757 update_overneed(Of1_desc *ofl)
2758 {
2759     Aliste      idxl;
2760     Ifl_desc    *ifl;
2761     Verneed     *vnd, *_vnd;
2762     Os_desc     *strosp;
2763     Str_tbl     *strtbl;
2764     Word        num = 0;

2766     _vnd = vnd = (Verneed *)ofl->ofl_osverneed->os_outdata->d_buf;

2768     /*
2769      * Determine which string table is appropriate.
2770      */
2771     if (OFL_IS_STATIC_OBJ(ofl)) {
2772         strosp = ofl->ofl_osstrtab;
2773         strtbl = ofl->ofl_strtab;

```

```

2774     } else {
2775         strosp = ofl->ofl_osdynstr;
2776         strtbl = ofl->ofl_dynstrtab;
2777     }

2779     /*
2780      * Traverse the shared object list looking for dependencies that have
2781      * versions defined within them.
2782      */
2783     for (APLIST_TRAVERSE(ofl->ofl_sos, idxl, ifl)) {
2784         Half      _cnt;
2785         Word      cnt = 0;
2786         Vernaux   *vnap, *vnmap;
2787         size_t    stoff;

2789         if (!(ifl->ifl_flags & FLG_IF_VERNEED))
2790             continue;

2792         vnd->vn_version = VER_NEED_CURRENT;

2794         (void) st_setstring(strtbl, ifl->ifl_soname, &stoff);
2795         vnd->vn_file = stoff;

2797         _vnap = vnmap = (Vernaux *) (vnd + 1);

2799         /*
2800          * Traverse the version index list recording
2801          * each version as a needed dependency.
2802          */
2803         for (_cnt = 0; _cnt <= ifl->ifl_vercnt; _cnt++) {
2804             Ver_index  *vip = &ifl->ifl_verndx[_cnt];

2806             if (vip->vi_flags & FLG_VER_REFER) {
2807                 (void) st_setstring(strtbl, vip->vi_name,
2808                                     &stoff);
2809                 vnmap->vna_name = stoff;

2811                 if (vip->vi_desc) {
2812                     vnmap->vna_hash = vip->vi_desc->vd_hash;
2813                     vnmap->vna_flags =
2814                         vip->vi_desc->vd_flags;
2815                 } else {
2816                     vnmap->vna_hash = 0;
2817                     vnmap->vna_flags = 0;
2818                 }
2819                 vnmap->vna_other = vip->vi_overndx;

2821                 /*
2822                  * If version A inherits version B, then
2823                  * B is implicit in A. It suffices for ld.so.1
2824                  * to verify A at runtime and skip B. The
2825                  * version normalization process sets the INFO
2826                  * flag for the versions we want ld.so.1 to
2827                  * skip.
2828                  */
2829                 if (vip->vi_flags & VER_FLG_INFO)
2830                     vnmap->vna_flags |= VER_FLG_INFO;

2832                 _vnap = vnmap;
2833                 vnmap++, cnt++;
2834                 _vnmap->vna_next =
2835                     /* LINTED */
2836                     (Word)((uintptr_t)vnmap - (uintptr_t)_vnmap);
2837             }
2838         }

```

```

2840     _vnap->vna_next = 0;
2842     /*
2843      * Record the versions auxiliary array offset and
2844      * the associated dependency count.
2845      */
2846     /* LINTED */
2847     vnd->vn_aux = (Word)((uintptr_t)(vnd + 1) - (uintptr_t)vnd);
2848     /* LINTED */
2849     vnd->vn_cnt = (Half)cnt;
2851     /*
2852      * Record the next versions offset and update the version
2853      * pointer. Remember the previous version offset as the very
2854      * last structures next pointer should be null.
2855      */
2856     _vnd = vnd;
2857     vnd = (Verneed *)vnap, num++;
2858     /* LINTED */
2859     _vnd->vn_next = (Word)((uintptr_t)vnd - (uintptr_t)_vnd);
2860 }
2861 _vnd->vn_next = 0;
2863 /*
2864 * Use sh_link to record the associated string table section, and
2865 * sh_info to indicate the number of entries contained in the section.
2866 */
2867 /* LINTED */
2868 ofl->ofl_osverneed->os_shdr->sh_link = (Word)elf_ndxscn(strosp->os_scn);
2869 ofl->ofl_osverneed->os_shdr->sh_info = num;
2871 return (1);
2872 }
2874 /*
2875 * Update syminfo section.
2876 */
2877 static uintptr_t
2878 update_osyminfo(Of1_desc *ofl)
2879 {
2880     Os_desc      *symosp, *infosp = ofl->ofl_ossyminfo;
2881     Syminfo      *sip = infosp->os_outdata->d_buf;
2882     Shdr         *shdr = infosp->os_shdr;
2883     char         *strtab;
2884     Aliste       idx;
2885     Sym_desc     *sdp;
2886     Sfltr_desc   *sftp;
2888     if (ofl->ofl_flags & FLG_OF_RELOBJ) {
2889         symosp = ofl->ofl_ossymtab;
2890         strtab = ofl->ofl_osstrtab->os_outdata->d_buf;
2891     } else {
2892         symosp = ofl->ofl_osdynsym;
2893         strtab = ofl->ofl_osdynstr->os_outdata->d_buf;
2894     }
2896     /* LINTED */
2897     infosp->os_shdr->sh_link = (Word)elf_ndxscn(symosp->os_scn);
2898     if (ofl->ofl_osdynamic)
2899         infosp->os_shdr->sh_info =
2900             /* LINTED */
2901             (Word)elf_ndxscn(ofl->ofl_osdynamic->os_scn);
2903 /*
2904 * Update any references with the index into the dynamic table.
2905 */

```

```

2906     for (APLIST_TRAVERSE(ofl->ofl_symdent, idx, sdp))
2907         sip[sdp->sd_symndx].si_boundto = sdp->sd_file->ifl_neededndx;
2909     /*
2910      * Update any filtee references with the index into the dynamic table.
2911      */
2912     for (ALIST_TRAVERSE(ofl->ofl_symfltrs, idx, sftp)) {
2913         Dfltr_desc     *dftp;
2915         dftp = alist_item(ofl->ofl_dtsfltrs, sftp->sft_idx);
2916         sip[sftp->sft_sdp->sd_symndx].si_boundto = dftp->dft_ndx;
2917     }
2919     /*
2920      * Display debugging information about section.
2921      */
2922     DBG_CALL(Debug_syminfo_title(ofl->ofl_lml));
2923     if (DBG_ENABLED) {
2924         Word     _cnt, cnt = shdr->sh_size / shdr->sh_entsize;
2925         Sym       *symtab = symosp->os_outdata->d_buf;
2926         Dyn       *dyn;
2928         if (ofl->ofl_osdynamic)
2929             dyn = ofl->ofl_osdynamic->os_outdata->d_buf;
2930         else
2931             dyn = NULL;
2933         for (_cnt = 1; _cnt < cnt; _cnt++) {
2934             if (sip[_cnt].si_flags || sip[_cnt].si_boundto)
2935                 /* LINTED */
2936                 DBG_CALL(Debug_syminfo_entry(ofl->ofl_lml, _cnt,
2937                     &sis[_cnt], &symtab[_cnt], strtab, dyn));
2938         }
2939     }
2940     return (1);
2941 }
2943 /*
2944 * Build the output elf header.
2945 */
2946 static uintptr_t
2947 update_ohdr(Of1_desc *ofl)
2948 {
2949     Ehdr     *ehdr = ofl->ofl_nehdr;
2951     /*
2952      * If an entry point symbol has already been established (refer
2953      * sym_validate()) simply update the elf header entry point with the
2954      * symbols value. If no entry point is defined it will have been filled
2955      * with the start address of the first section within the text segment
2956      * (refer update_outfile()).
2957      */
2958     if (ofl->ofl_entry)
2959         ehdr->e_entry =
2960             ((Sym_desc *) (ofl->ofl_entry))->sd_sym->st_value;
2962     ehdr->e_ident[EI_DATA] = ld_targ.t_m.m_data;
2963     ehdr->e_version = ofl->ofl_dehdr->e_version;
2965     /*
2966      * When generating a relocatable object under -z symbolcap, set the
2967      * e_machine to be generic, and remove any e_flags. Input relocatable
2968      * objects may identify alternative e_machine (m.machplus) and e_flags
2969      * values. However, the functions within the created output object
2970      * are selected at runtime using the capabilities mechanism, which
2971      * supersedes the e-machine and e_flags information. Therefore,

```

```

2972     * e_machine and e_flag values are not propagated to the output object,
2973     * as these values might prevent the kernel from loading the object
2974     * before the runtime linker gets control.
2975     */
2976     if (ofl->ofl_flags & FLG_OF_OTOSCAP) {
2977         ehdr->e_machine = ld_targ.t_m.m_mach;
2978         ehdr->e_flags = 0;
2979     } else {
2980         /*
2981          * Note. it may be necessary to update the e_flags field in the
2982          * machine dependent section.
2983          */
2984         ehdr->e_machine = ofl->ofl_dehdr->e_machine;
2985         ehdr->e_flags = ofl->ofl_dehdr->e_flags;
2986
2987         if (ehdr->e_machine != ld_targ.t_m.m_mach) {
2988             if (ehdr->e_machine != ld_targ.t_m.m_machplus)
2989                 return (S_ERROR);
2990             if ((ehdr->e_flags & ld_targ.t_m.m_flagsplus) == 0)
2991                 return (S_ERROR);
2992         }
2993     }
2994
2995     if (ofl->ofl_flags & FLG_OF_SHAROBJ)
2996         ehdr->e_type = ET_DYN;
2997     else if (ofl->ofl_flags & FLG_OF_RELOBJ)
2998         ehdr->e_type = ET_REL;
2999     else
3000         ehdr->e_type = ET_EXEC;
3001
3002     return (1);
3003 }
3004
3005 /*
3006  * Perform move table expansion.
3007  */
3008 static void
3009 expand_move(Of1_desc *ofl, Sym_desc *sdp, Move *mvp)
3010 {
3011     Os_desc      *osp;
3012     uchar_t      *taddr, *taddr0;
3013     Sxword       offset;
3014     Half         cnt;
3015     uint_t       stride;
3016
3017     osp = ofl->ofl_ishparexp->is_osdesc;
3018     offset = sdp->sd_sym->st_value - osp->os_shdr->sh_addr;
3019
3020     taddr0 = taddr = osp->os_outdata->d_buf;
3021     taddr += offset;
3022     taddr = taddr + mvp->m_poffset;
3023
3024     for (cnt = 0; cnt < mvp->m_repeat; cnt++) {
3025         /* LINTED */
3026         DBG_CALL(DBG_move_expand(ofl->ofl_lml, mvp,
3027             (Addr)(taddr - taddr0)));
3028         stride = (uint_t)mvp->m_stride + 1;
3029
3030         /*
3031          * Update the target address based upon the move entry size.
3032          * This size was validated in ld_process_move().
3033          */
3034         /* LINTED */
3035         switch (ELF_M_SIZE(mvp->m_info)) {
3036         case 1:
3037             /* LINTED */

```

```

3038         *taddr = (uchar_t)mvp->m_value;
3039         taddr += stride;
3040         break;
3041     case 2:
3042         /* LINTED */
3043         *((Half *)taddr) = (Half)mvp->m_value;
3044         taddr += 2 * stride;
3045         break;
3046     case 4:
3047         /* LINTED */
3048         *((Word *)taddr) = (Word)mvp->m_value;
3049         taddr += 4 * stride;
3050         break;
3051     case 8:
3052         /* LINTED */
3053         *((u_longlong_t *)taddr) = mvp->m_value;
3054         taddr += 8 * stride;
3055         break;
3056     }
3057 }
3058 }
3059
3060 /*
3061  * Update Move sections.
3062  */
3063 static void
3064 update_move(Of1_desc *ofl)
3065 {
3066     Word          ndx = 0;
3067     ofl_flag_t    flags = ofl->ofl_flags;
3068     Move          *omvp;
3069     Aliste        idx1;
3070     Sym_desc      *sdp;
3071
3072     /*
3073      * Determine the index of the symbol table that will be referenced by
3074      * the Move section.
3075      */
3076     if (OFL_ALLOW_DYNSYM(ofl))
3077         /* LINTED */
3078         ndx = (Word) elf_ndxscn(ofl->ofl_osdynsym->os_scn);
3079     else if (!(flags & FLG_OF_STRIP) || (flags & FLG_OF_RELOBJ))
3080         /* LINTED */
3081         ndx = (Word) elf_ndxscn(ofl->ofl_ossymtab->os_scn);
3082
3083     /*
3084      * Update sh_link of the Move section, and point to the new Move data.
3085      */
3086     if (ofl->ofl_osmove) {
3087         ofl->ofl_osmove->os_shdr->sh_link = ndx;
3088         omvp = (Move *)ofl->ofl_osmove->os_outdata->d_buf;
3089     }
3090
3091     /*
3092      * Update symbol entry index
3093      */
3094     for (APLIST_TRAVERSE(ofl->ofl_parsyms, idx1, sdp)) {
3095         Aliste        idx2;
3096         Mv_desc       *mdp;
3097
3098         /*
3099          * Expand move table
3100          */
3101         if (sdp->sd_flags & FLG_SY_PAREXP) {
3102             const char *str;

```

```

3104         if (flags & FLG_OF_STATIC)
3105             str = MSG_INTL(MSG_PSYM_EXPREASON1);
3106         else if (ofl->ofl_flags1 & FLG_OF1_NOPARTI)
3107             str = MSG_INTL(MSG_PSYM_EXPREASON2);
3108         else
3109             str = MSG_INTL(MSG_PSYM_EXPREASON3);
3111
3112         DBG_CALL(Dbg_move_parexpn(ofl->ofl_lml,
3113                                 sdp->sd_name, str));
3114
3115         for (ALIST_TRAVERSE(sdp->sd_move, idx2, mdp)) {
3116             DBG_CALL(Dbg_move_entry1(ofl->ofl_lml, 0,
3117                                     mdp->md_move, sdp));
3118             expand_move(ofl, sdp, mdp->md_move);
3119         }
3120         continue;
3121     }
3122
3123     /*
3124     * Process move table
3125     */
3126     DBG_CALL(Dbg_move_outmove(ofl->ofl_lml, sdp->sd_name));
3127
3128     for (ALIST_TRAVERSE(sdp->sd_move, idx2, mdp)) {
3129         Move      *imvp;
3130         int       idx = 1;
3131         Sym       *sym;
3132
3133         imvp = mdp->md_move;
3134         sym = sdp->sd_sym;
3135
3136         DBG_CALL(Dbg_move_entry1(ofl->ofl_lml, 1, imvp, sdp));
3137
3138         *omvp = *imvp;
3139         if ((flags & FLG_OF_RELOBJ) == 0) {
3140             if (ELF_ST_BIND(sym->st_info) == STB_LOCAL) {
3141                 Os_desc *osp = sdp->sd_isc->is_osdesc;
3142                 Word    ndx = osp->os_identndx;
3143
3144                 omvp->m_info =
3145                     /* LINTED */
3146                     ELF_M_INFO(ndx, imvp->m_info);
3147
3148                 if (ELF_ST_TYPE(sym->st_info) !=
3149                     STT_SECTION) {
3150                     omvp->m_poffset =
3151                         sym->st_value -
3152                         osp->os_shdr->sh_addr +
3153                         imvp->m_poffset;
3154                 }
3155             } else {
3156                 omvp->m_info =
3157                     /* LINTED */
3158                     ELF_M_INFO(sdp->sd_symndx,
3159                                 imvp->m_info);
3160             }
3161         } else {
3162             Boolean    isredloc = FALSE;
3163
3164             if ((ELF_ST_BIND(sym->st_info) == STB_LOCAL) &&
3165                 (ofl->ofl_flags & FLG_OF_REDLSYM))
3166                 isredloc = TRUE;
3167
3168             if (isredloc && !(sdp->sd_move)) {
3169                 Os_desc *osp = sdp->sd_isc->is_osdesc;
3170                 Word    ndx = osp->os_identndx;

```

```

3171             omvp->m_info =
3172                 /* LINTED */
3173                 ELF_M_INFO(ndx, imvp->m_info);
3175
3176             omvp->m_poffset += sym->st_value;
3177         } else {
3178             if (isredloc)
3179                 DBG_CALL(Dbg_syms_reduce(ofl,
3180                                         DBG_SYM_REDUCE_RETAIN,
3181                                         sdp, idx,
3182                                         ofl->ofl_osmove->os_name));
3183
3184             omvp->m_info =
3185                 /* LINTED */
3186                 ELF_M_INFO(sdp->sd_symndx,
3187                             imvp->m_info);
3188         }
3189
3190         DBG_CALL(Dbg_move_entry1(ofl->ofl_lml, 0, omvp, sdp));
3191         omvp++;
3192         idx++;
3193     }
3194 }
3195
3197 /*
3198 * Scan through the SHT_GROUP output sections. Update their sh_link/sh_info
3199 * fields as well as the section contents.
3200 */
3201 static uintptr_t
3202 update_ogroup(Ofldesc *ofl)
3203 {
3204     Aliste      idx;
3205     Os_desc     *osp;
3206     uintptr_t   error = 0;
3207
3208     for (APLIST_TRAVERSE(ofl->ofl_osgroups, idx, osp)) {
3209         Is_desc  *isp;
3210         Ifl_desc *ifl;
3211         Shdr     *shdr = osp->os_shdr;
3212         Sym_desc *sdp;
3213         Xword    i, grpcnt;
3214         Word     *gdata;
3215
3216         /*
3217         * Since input GROUP sections always create unique
3218         * output GROUP sections - we know there is only one
3219         * item on the list.
3220         */
3221         isp = ld_os_first_isdesc(osp);
3222
3223         ifl = isp->is_file;
3224         sdp = ifl->ifl_olddndx[isp->is_shdr->sh_info];
3225         shdr->sh_link = (Word)elf_ndxscn(ofl->ofl_ossymtab->os_scn);
3226         shdr->sh_info = sdp->sd_symndx;
3227
3228         /*
3229         * Scan through the group data section and update
3230         * all of the links to new values.
3231         */
3232         grpcnt = shdr->sh_size / shdr->sh_entsize;
3233         gdata = (Word *)osp->os_outdata->d_buf;
3234
3235         for (i = 1; i < grpcnt; i++) {

```

```

3236     Os_desc *_osp;
3237     Is_desc *_isp = ifl->ifl_isdesc[gdata[i]];

3239     /*
3240     * If the referenced section didn't make it to the
3241     * output file - just zero out the entry.
3242     */
3243     if ((_osp = _isp->is_osdesc) == NULL)
3244         gdata[i] = 0;
3245     else
3246         gdata[i] = (Word)elf_ndxscn(_osp->os_scn);
3247     }
3248 }
3249 return (error);
3250 }

3252 static void
3253 update_ostrtab(Os_desc *_osp, Str_tbl *_stp, uint_t extra)
3254 {
3255     Elf_Data *_data;

3257     if (osp == NULL)
3258         return;

3260     data = osp->os_outdata;
3261     assert(data->d_size == (st_getstrtab_sz(stp) + extra));
3262     (void) st_setstrbuf(stp, data->d_buf, data->d_size - extra);
3263     /* If leaving an extra hole at the end, zero it */
3264     if (extra > 0)
3265         (void) memset((char *)data->d_buf + data->d_size - extra,
3266                     0x0, extra);
3267 }

3269 /*
3270 * Update capabilities information.
3271 *
3272 * If string table capabilities exist, then the associated string must be
3273 * translated into an offset into the string table.
3274 */
3275 static void
3276 update_osc(const Of1_desc *_ofl)
3277 {
3278     Os_desc *_stosp, *_cosp;
3279     Cap *_cap;
3280     Str_tbl *_strtbl;
3281     Capstr *_capstr;
3282     size_t stoff;
3283     Aliste idx1;

3285     /*
3286     * Determine which symbol table or string table is appropriate.
3287     */
3288     if (OFL_IS_STATIC_OBJ(ofl)) {
3289         _stosp = ofl->ofl_osstrtab;
3290         _strtbl = ofl->ofl_strtbl;
3291     } else {
3292         _stosp = ofl->ofl_osdynstr;
3293         _strtbl = ofl->ofl_dynstrtbl;
3294     }

3296     /*
3297     * If symbol capabilities exist, set the sh_link field of the .SUNW_cap
3298     * section to the .SUNW_capinfo section.
3299     */
3300     if (ofl->ofl_oscinfo) {
3301         _cosp = ofl->ofl_osc;

```

```

3302         _cosp->os_shdr->sh_link =
3303             (Word)elf_ndxscn(ofl->ofl_oscinfo->os_scn);
3304     }

3306     /*
3307     * If there are capability strings to process, set the sh_info
3308     * field of the .SUNW_cap section to the associated string table, and
3309     * proceed to process any CA_SUNW_PLAT entries.
3310     */
3311     if ((ofl->ofl_flags & FLG_OF_CAPSTRS) == 0)
3312         return;

3314     _cosp = ofl->ofl_osc;
3315     _cosp->os_shdr->sh_info = (Word)elf_ndxscn(stosp->os_scn);

3317     _cap = ofl->ofl_osc->os_outdata->d_buf;

3319     /*
3320     * Determine whether an object capability identifier, or object
3321     * machine/platform capabilities exists.
3322     */
3323     _capstr = &ofl->ofl_oscset.oc_id;
3324     if (_capstr->cs_str) {
3325         (void) st_setstring(_strtbl, _capstr->cs_str, &stoff);
3326         _cap[_capstr->cs_ndx].c_un.c_ptr = stoff;
3327     }
3328     for (ALIST_TRAVERSE(ofl->ofl_oscset.oc_plat.cl_val, idx1, _capstr)) {
3329         (void) st_setstring(_strtbl, _capstr->cs_str, &stoff);
3330         _cap[_capstr->cs_ndx].c_un.c_ptr = stoff;
3331     }
3332     for (ALIST_TRAVERSE(ofl->ofl_oscset.oc_mach.cl_val, idx1, _capstr)) {
3333         (void) st_setstring(_strtbl, _capstr->cs_str, &stoff);
3334         _cap[_capstr->cs_ndx].c_un.c_ptr = stoff;
3335     }

3337     /*
3338     * Determine any symbol capability identifiers, or machine/platform
3339     * capabilities.
3340     */
3341     if (ofl->ofl_capgroups) {
3342         Cap_group *_cgp;

3344         for (APLIST_TRAVERSE(ofl->ofl_capgroups, idx1, _cgp)) {
3345             Objcapset *_ocapset = &_cgp->cg_set;
3346             Aliste idx2;

3348             _capstr = &_ocapset->oc_id;
3349             if (_capstr->cs_str) {
3350                 (void) st_setstring(_strtbl, _capstr->cs_str,
3351                                     &stoff);
3352                 _cap[_capstr->cs_ndx].c_un.c_ptr = stoff;
3353             }
3354             for (ALIST_TRAVERSE(_ocapset->oc_plat.cl_val, idx2,
3355                                 _capstr)) {
3356                 (void) st_setstring(_strtbl, _capstr->cs_str,
3357                                     &stoff);
3358                 _cap[_capstr->cs_ndx].c_un.c_ptr = stoff;
3359             }
3360             for (ALIST_TRAVERSE(_ocapset->oc_mach.cl_val, idx2,
3361                                 _capstr)) {
3362                 (void) st_setstring(_strtbl, _capstr->cs_str,
3363                                     &stoff);
3364                 _cap[_capstr->cs_ndx].c_un.c_ptr = stoff;
3365             }
3366         }
3367     }

```

```

3368 }
3370 /*
3371  * Update the .SUNW_capinfo, and possibly the .SUNW_capchain sections.
3372  */
3373 static void
3374 update_oscapinfo(Of1_desc *of1)
3375 {
3376     Os_desc      *symosp, *ciosp, *ccosp = NULL;
3377     Capinfo      *ocapinfo;
3378     Capchain     *ocapchain;
3379     Cap_avlnode  *cav;
3380     Word         chainndx = 0;
3382     /*
3383      * Determine which symbol table is appropriate.
3384      */
3385     if (OFL_IS_STATIC_OBJ(of1))
3386         symosp = of1->ofl_ossymtab;
3387     else
3388         symosp = of1->ofl_osdynsym;
3390     /*
3391      * Update the .SUNW_capinfo sh_link to point to the appropriate symbol
3392      * table section. If we're creating a dynamic object, the
3393      * .SUNW_capinfo sh_info is updated to point to the .SUNW_capchain
3394      * section.
3395      */
3396     ciosp = of1->ofl_oscapinfo;
3397     ciosp->os_shdr->sh_link = (Word)elf_ndxscn(symosp->os_scn);
3399     if (OFL_IS_STATIC_OBJ(of1) == 0) {
3400         ccosp = of1->ofl_oscapchain;
3401         ciosp->os_shdr->sh_info = (Word)elf_ndxscn(ccosp->os_scn);
3402     }
3404     /*
3405      * Establish the data for each section. The first element of each
3406      * section defines the section's version number.
3407      */
3408     ocapinfo = ciosp->os_outdata->d_buf;
3409     ocapinfo[0] = CAPINFO_CURRENT;
3410     if (ccosp) {
3411         ocapchain = ccosp->os_outdata->d_buf;
3412         ocapchain[chainndx++] = CAPCHAIN_CURRENT;
3413     }
3415     /*
3416      * Traverse all capabilities families. Each member has a .SUNW_capinfo
3417      * assignment. The .SUNW_capinfo entry differs for relocatable objects
3418      * and dynamic objects.
3419      *
3420      * Relocatable objects:
3421      *
3422      *
3423      * Family lead:      CAPINFO_SUNW_GLOB      lead symbol index
3424      * Family lead alias: CAPINFO_SUNW_GLOB      lead symbol index
3425      * Family member:    .SUNW_cap index        lead symbol index
3426      *
3427      * Dynamic objects:
3428      *
3429      *
3430      * Family lead:      CAPINFO_SUNW_GLOB      .SUNW_capchain index
3431      * Family lead alias: CAPINFO_SUNW_GLOB      .SUNW_capchain index
3432      * Family member:    .SUNW_cap index        lead symbol index
3433      */

```

```

3434     * The ELF_C_GROUP field identifies a capabilities symbol. Lead
3435     * capability symbols, and lead capability aliases are identified by
3436     * a CAPINFO_SUNW_GLOB group identifier. For family members, the
3437     * ELF_C_GROUP provides an index to the associate capabilities group
3438     * (i.e, an index into the SUNW_cap section that defines a group).
3439     *
3440     * For relocatable objects, the ELF_C_SYM field identifies the lead
3441     * capability symbol. For the lead symbol itself, the .SUNW_capinfo
3442     * index is the same as the ELF_C_SYM value. For lead alias symbols,
3443     * the .SUNW_capinfo index differs from the ELF_C_SYM value. This
3444     * differentiation of CAPINFO_SUNW_GLOB symbols allows ld(1) to
3445     * identify, and propagate lead alias symbols. For example, the lead
3446     * capability symbol memcpy() would have the ELF_C_SYM for memcpy(),
3447     * and the lead alias _memcpy() would also have the ELF_C_SYM for
3448     * memcpy().
3449     *
3450     * For dynamic objects, both a lead capability symbol, and alias symbol
3451     * would have a ELF_C_SYM value that represents the same capability
3452     * chain index. The capability chain allows ld.so.1 to traverse a
3453     * family chain for a given lead symbol, and select the most appropriate
3454     * family member. The .SUNW_capchain array contains a series of symbol
3455     * indexes for each family member:
3456     *
3457     *   chaincap[n]  chaincap[n + 1]  chaincap[n + 2]  chaincap[n + x]
3458     *   foo() ndx   foo%x() ndx       foo%y() ndx       0
3459     *
3460     * For family members, the ELF_C_SYM value associates the capability
3461     * members with their family lead symbol. This association, although
3462     * unused within a dynamic object, allows ld(1) to identify, and
3463     * propagate family members when processing relocatable objects.
3464     */
3465     for (cav = avl_first(of1->ofl_capfamilies); cav;
3466          cav = AVL_NEXT(of1->ofl_capfamilies, cav)) {
3467         Cap_sym      *csp;
3468         Aliste       idx;
3469         Sym_desc     *asdp, *lsdp = cav->cn_symavlnode.sav_sdp;
3471         if (ccosp) {
3472             /*
3473              * For a dynamic object, identify this lead symbol, and
3474              * point it to the head of a capability chain. Set the
3475              * head of the capability chain to the same lead symbol.
3476              */
3477             ocapinfo[lsdp->sd_symndx] =
3478                 ELF_C_INFO(chainndx, CAPINFO_SUNW_GLOB);
3479             ocapchain[chainndx] = lsd->sd_symndx;
3480         } else {
3481             /*
3482              * For a relocatable object, identify this lead symbol,
3483              * and set the lead symbol index to itself.
3484              */
3485             ocapinfo[lsdp->sd_symndx] =
3486                 ELF_C_INFO(lsd->sd_symndx, CAPINFO_SUNW_GLOB);
3487         }
3489         /*
3490          * Gather any lead symbol aliases.
3491          */
3492         for (APLIST_TRAVERSE(cav->cn_aliases, idx, asdp)) {
3493             if (ccosp) {
3494                 /*
3495                  * For a dynamic object, identify this lead
3496                  * alias symbol, and point it to the same
3497                  * capability chain index as the lead symbol.
3498                  */
3499                 ocapinfo[asdp->sd_symndx] =

```



```

3500         ELF_C_INFO(chainndx, CAPINFO_SUNW_GLOB);
3501     } else {
3502         /*
3503          * For a relocatable object, identify this lead
3504          * alias symbol, and set the lead symbol index
3505          * to the lead symbol.
3506          */
3507         ocapinfo[asdp->sd_symndx] =
3508             ELF_C_INFO(ldsp->sd_symndx,
3509                 CAPINFO_SUNW_GLOB);
3510     }
3511 }

3513 chainndx++;

3515 /*
3516  * Gather the family members.
3517  */
3518 for (APLIST_TRAVERSE(cav->cn_members, idx, csp)) {
3519     Sym_desc      *msdp = csp->cs_sdp;

3521     /*
3522      * Identify the members capability group, and the lead
3523      * symbol of the family this symbol is a member of.
3524      */
3525     ocapinfo[msdp->sd_symndx] =
3526         ELF_C_INFO(ldsp->sd_symndx, csp->cs_group->cg_ndx);
3527     if (ccosp) {
3528         /*
3529          * For a dynamic object, set the next capability
3530          * chain to point to this family member.
3531          */
3532         ocapchain[chainndx++] = msdp->sd_symndx;
3533     }
3534 }

3536 /*
3537  * Any chain of family members is terminated with a 0 element.
3538  */
3539 if (ccosp)
3540     ocapchain[chainndx++] = 0;
3541 }
3542 }

3544 /*
3545  * Translate the shdr->sh_{link, info} from its input section value to that
3546  * of the corresponding shdr->sh_{link, info} output section value.
3547  */
3548 static Word
3549 translate_link(Of1_desc *of1, Os_desc *osp, Word link, const char *msg)
3550 {
3551     Is_desc      *isp;
3552     Ifl_desc     *ifl;

3554     /*
3555      * Don't translate the special section numbers.
3556      */
3557     if (link >= SHN_LORESERVE)
3558         return (link);

3560     /*
3561      * Does this output section translate back to an input file.  If not
3562      * then there is no translation to do.  In this case we will assume that
3563      * if sh_link has a value, it's the right value.
3564      */
3565     isp = ld_os_first_isdesc(osp);

```

```

3566     if ((ifl = isp->is_file) == NULL)
3567         return (link);

3569     /*
3570      * Sanity check to make sure that the sh_{link, info} value
3571      * is within range for the input file.
3572      */
3573     if (link >= ifl->ifl_shnum) {
3574         ld_eprintf(of1, ERR_WARNING, msg, ifl->ifl_name,
3575             EC_WORD(isp->is_scnndx), isp->is_name, EC_XWORD(link));
3576         return (link);
3577     }

3579     /*
3580      * Follow the link to the input section.
3581      */
3582     if ((isp = ifl->ifl_isdesc[link]) == NULL)
3583         return (0);
3584     if ((osp = isp->is_osdesc) == NULL)
3585         return (0);

3587     /* LINTED */
3588     return ((Word)elf_ndxscn(osp->os_scn));
3589 }

3591 /*
3592  * Having created all of the necessary sections, segments, and associated
3593  * headers, fill in the program headers and update any other data in the
3594  * output image.  Some general rules:
3595  *
3596  * - If an interpreter is required always generate a PT_PHDR entry as
3597  *   well.  It is this entry that triggers the kernel into passing the
3598  *   interpreter an aux vector instead of just a file descriptor.
3599  *
3600  * - When generating an image that will be interpreted (ie. a dynamic
3601  *   executable, a shared object, or a static executable that has been
3602  *   provided with an interpreter - weird, but possible), make the initial
3603  *   loadable segment include both the ehdr and phdr[].  Both of these
3604  *   tables are used by the interpreter therefore it seems more intuitive
3605  *   to explicitly defined them as part of the mapped image rather than
3606  *   relying on page rounding by the interpreter to allow their access.
3607  *
3608  * - When generating a static image that does not require an interpreter
3609  *   have the first loadable segment indicate the address of the first
3610  *   .section as the start address (things like /kernel/unix and ufsboot
3611  *   expect this behavior).
3612  */
3613 uintptr_t
3614 ld_update_outfile(Of1_desc *of1)
3615 {
3616     Addr          size, etext, vaddr;
3617     Sg_desc       *sgp;
3618     Sg_desc       *dtracesgp = NULL, *capsgp = NULL, *intpsgp = NULL;
3619     Os_desc       *osp;
3620     int           phdrndx = 0, segndx = -1, secndx, intppndx, intpsndx;
3621     int           dtracepndx, dtracesndx, cappndx, capsndx;
3622     Ehdr          *ehdr = of1->ofl_nehdr;
3623     Shdr          *hshdr;
3624     Phdr          *phdr = NULL;
3625     Word          phdrsz = (ehdr->e_phnum * ehdr->e_phentsize), hshcnndx;
3626     ofl_flag_t    flags = of1->ofl_flags;
3627     Word          ehdrsz = ehdr->e_ehsize;
3628     Boolean       nobits;
3629     Off           offset;
3630     Aliste        idxl;

```

```

3632 /*
3633  * Initialize the starting address for the first segment. Executables
3634  * have different starting addresses depending upon the target ABI,
3635  * where as shared objects have a starting address of 0. If this is
3636  * a 64-bit executable that is being constructed to run in a restricted
3637  * address space, use an alternative origin that will provide more free
3638  * address space for the the eventual process.
3639  */
3640 if (ofl->ofl_flags & FLG_OF_EXEC) {
3641 #if defined(_ELF64)
3642     if (ofl->ofl_ocapset.oc_sf_1.cm_val & SF1_SUNW_ADDR32)
3643         vaddr = ld_targ.t_m.m_seg_m_aorigin;
3644     else
3645         vaddr = ld_targ.t_m.m_seg_m_origin;
3646 #endif
3647 } else
3648     vaddr = 0;

3650 /*
3651  * Loop through the segment descriptors and pick out what we need.
3652  */
3653 DBG_CALL(Dbg_seg_title(ofl->ofl_lml));
3654 for (APLIST_TRAVERSE(ofl->ofl_segs, idx1, sgp)) {
3655     Phdr *phdr = &(sgp->sg_phdr);
3656     Xword p_align;
3657     Aliste idx2;
3658     Sym_desc *sdp;

3660     segndx++;

3662 /*
3663  * If an interpreter is required generate a PT_INTERP and
3664  * PT_PHDR program header entry. The PT_PHDR entry describes
3665  * the program header table itself. This information will be
3666  * passed via the aux vector to the interpreter (ld.so.1).
3667  * The program header is actually part of the first
3668  * loadable segment (and the PT_PHDR entry is the first entry),
3669  * therefore its virtual address isn't known until the first
3670  * loadable segment is processed.
3671  */
3672 if (phdr->p_type == PT_PHDR) {
3673     if (ofl->ofl_osinterp) {
3674         phdr->p_offset = ehdr->e_phoff;
3675         phdr->p_filesz = phdr->p_memsz = phdrsz;

3677         DBG_CALL(Dbg_seg_entry(ofl, segndx, sgp));
3678         ofl->ofl_phdr[phdrndx++] = *phdr;
3679     }
3680     continue;
3681 }
3682 if (phdr->p_type == PT_INTERP) {
3683     if (ofl->ofl_osinterp) {
3684         intpsgp = sgp;
3685         intpsndx = segndx;
3686         intppndx = phdrndx++;
3687     }
3688     continue;
3689 }

3691 /*
3692  * If we are creating a PT_SUNWTRACE segment, remember where
3693  * the program header is. The header values are assigned after
3694  * update_osym() has completed and the symbol table addresses
3695  * have been updated.
3696  */
3697 if (phdr->p_type == PT_SUNWTRACE) {

```

```

3698         if (ofl->ofl_dtracesym &&
3699             ((flags & FLG_OF_RELOBJ) == 0)) {
3700             dtracesgp = sgp;
3701             dtracesndx = segndx;
3702             dtracepndx = phdrndx++;
3703         }
3704         continue;
3705     }

3707 /*
3708  * If a hardware/software capabilities section is required,
3709  * generate the PT_SUNWCAP header. Note, as this comes before
3710  * the first loadable segment, we don't yet know its real
3711  * virtual address. This is updated later.
3712  */
3713 if (phdr->p_type == PT_SUNWCAP) {
3714     if (ofl->ofl_oscaps && (ofl->ofl_flags & FLG_OF_PTCAP) &&
3715         ((flags & FLG_OF_RELOBJ) == 0)) {
3716         capsqp = sgp;
3717         capsndx = segndx;
3718         cappndx = phdrndx++;
3719     }
3720     continue;
3721 }

3723 /*
3724  * As the dynamic program header occurs after the loadable
3725  * headers in the segment descriptor table, all the address
3726  * information for the .dynamic output section will have been
3727  * figured out by now.
3728  */
3729 if (phdr->p_type == PT_DYNAMIC) {
3730     if (OFL_ALLOW_DYNSYM(ofl)) {
3731         Shdr *shdr = ofl->ofl_osdynamic->os_shdr;

3733         phdr->p_vaddr = shdr->sh_addr;
3734         phdr->p_offset = shdr->sh_offset;
3735         phdr->p_filesz = shdr->sh_size;
3736         phdr->p_flags = ld_targ.t_m.m_dataseg_perm;

3738         DBG_CALL(Dbg_seg_entry(ofl, segndx, sgp));
3739         ofl->ofl_phdr[phdrndx++] = *phdr;
3740     }
3741     continue;
3742 }

3744 /*
3745  * As the unwind (.eh_frame_hdr) program header occurs after
3746  * the loadable headers in the segment descriptor table, all
3747  * the address information for the .eh_frame output section
3748  * will have been figured out by now.
3749  */
3750 if (phdr->p_type == PT_SUNW_UNWIND) {
3751     Shdr *shdr;

3753     if (ofl->ofl_unwindhdr == NULL)
3754         continue;

3756     shdr = ofl->ofl_unwindhdr->os_shdr;

3758     phdr->p_flags = PF_R;
3759     phdr->p_vaddr = shdr->sh_addr;
3760     phdr->p_memsz = shdr->sh_size;
3761     phdr->p_filesz = shdr->sh_size;
3762     phdr->p_offset = shdr->sh_offset;
3763     phdr->p_align = shdr->sh_addralign;

```

```

3764     phdr->p_paddr = 0;
3765     ofl->ofl_phdr[phdrndx++] = *phdr;
3766     continue;
3767 }
3769 /*
3770 * The sunwstack program is used to convey non-default
3771 * flags for the process stack. Only emit it if it would
3772 * change the default.
3773 */
3774 if (phdr->p_type == PT_SUNWSTACK) {
3775     if (((flags & FLG_OF_RELOBJ) == 0) &&
3776         ((sgp->sg_flags & FLG_SG_DISABLED) == 0))
3777         ofl->ofl_phdr[phdrndx++] = *phdr;
3778     continue;
3779 }
3781 /*
3782 * As the TLS program header occurs after the loadable
3783 * headers in the segment descriptor table, all the address
3784 * information for the .tls output section will have been
3785 * figured out by now.
3786 */
3787 if (phdr->p_type == PT_TLS) {
3788     Os_desc *tlsosp;
3789     Shdr *lastfileshdr = NULL;
3790     Shdr *firstshdr = NULL, *lastshdr;
3791     Aliste idx;
3793     if (ofl->ofl_ostlsseg == NULL)
3794         continue;
3796     /*
3797     * Scan the output sections that have contributed TLS.
3798     * Remember the first and last so as to determine the
3799     * TLS memory size requirement. Remember the last
3800     * progbits section to determine the TLS data
3801     * contribution, which determines the TLS program
3802     * header filesz.
3803     */
3804     for (APLIST_TRAVERSE(ofl->ofl_ostlsseg, idx, tlsosp)) {
3805         Shdr *tlsshdr = tlsosp->os_shdr;
3807         if (firstshdr == NULL)
3808             firstshdr = tlsshdr;
3809         if (tlsshdr->sh_type != SHT_NOBITS)
3810             lastfileshdr = tlsshdr;
3811         lastshdr = tlsshdr;
3812     }
3814     phdr->p_flags = PF_R | PF_W;
3815     phdr->p_vaddr = firstshdr->sh_addr;
3816     phdr->p_offset = firstshdr->sh_offset;
3817     phdr->p_align = firstshdr->sh_addralign;
3819     /*
3820     * Determine the initialized TLS data size. This
3821     * address range is from the start of the TLS segment
3822     * to the end of the last piece of initialized data.
3823     */
3824     if (lastfileshdr)
3825         phdr->p_filesz = lastfileshdr->sh_offset +
3826             lastfileshdr->sh_size - phdr->p_offset;
3827     else
3828         phdr->p_filesz = 0;

```

```

3830     /*
3831     * Determine the total TLS memory size. This includes
3832     * all TLS data and TLS uninitialized data. This
3833     * address range is from the start of the TLS segment
3834     * to the memory address of the last piece of
3835     * uninitialized data.
3836     */
3837     phdr->p_memsz = lastshdr->sh_addr +
3838         lastshdr->sh_size - phdr->p_vaddr;
3840     DBG_CALL(Dbg_seg_entry(ofl, segndx, sgp));
3841     ofl->ofl_phdr[phdrndx] = *phdr;
3842     ofl->ofl_tlsphdr = &ofl->ofl_phdr[phdrndx++];
3843     continue;
3844 }
3846 /*
3847 * If this is an empty segment declaration, it will occur after
3848 * all other loadable segments. As empty segments can be
3849 * defined with fixed addresses, make sure that no loadable
3850 * segments overlap. This might occur as the object evolves
3851 * and the loadable segments grow, thus encroaching upon an
3852 * existing segment reservation.
3853 * Segments are only created for dynamic objects, thus this
3854 * checking can be skipped when building a relocatable object.
3855 */
3856 if (!(flags & FLG_OF_RELOBJ) &&
3857     (sgp->sg_flags & FLG_SG_EMPTY)) {
3858     int i;
3859     Addr v_e;
3862     vaddr = phdr->p_vaddr;
3863     phdr->p_memsz = sgp->sg_length;
3864     DBG_CALL(Dbg_seg_entry(ofl, segndx, sgp));
3865     ofl->ofl_phdr[phdrndx++] = *phdr;
3867     if (phdr->p_type != PT_LOAD)
3868         continue;
3870     v_e = vaddr + phdr->p_memsz;
3872     /*
3873     * Check overlaps
3874     */
3875     for (i = 0; i < phdrndx - 1; i++) {
3876         Addr p_s = (ofl->ofl_phdr[i]).p_vaddr;
3877         Addr p_e;
3879         if ((ofl->ofl_phdr[i]).p_type != PT_LOAD)
3880             continue;
3882         p_e = p_s + (ofl->ofl_phdr[i]).p_memsz;
3883         if (((p_s <= vaddr) && (p_e > vaddr)) ||
3884             ((vaddr <= p_s) && (v_e > p_s)))
3885             ld_eprintf(ofl, ERR_WARNING,
3886                 MSG_INTL(MSG_UPD_SEGOVERLAP),
3887                 ofl->ofl_name, EC_ADDR(p_e),
3888                 sgp->sg_name, EC_ADDR(vaddr));
3889     }
3890     continue;
3891 }
3893 /*
3894 * Having processed any of the special program headers any
3895 * remaining headers will be built to express individual

```

```

3896     * segments. Segments are only built if they have output
3897     * section descriptors associated with them (ie. some form of
3898     * input section has been matched to this segment).
3899     */
3900     if (sgp->sg_osdescs == NULL)
3901         continue;
3902
3903     /*
3904     * Determine the segments offset and size from the section
3905     * information provided from elf_update().
3906     * Allow for multiple NOBITS sections.
3907     */
3908     osp = sgp->sg_osdescs->apl_data[0];
3909     hshdr = osp->os_shdr;
3910
3911     phdr->p_filesz = 0;
3912     phdr->p_memsz = 0;
3913     phdr->p_offset = offset = hshdr->sh_offset;
3914
3915     nobits = ((hshdr->sh_type == SHT_NOBITS) &&
3916             ((sgp->sg_flags & FLG_SG_PHREQ) == 0));
3917
3918     for (APLIST_TRAVERSE(sgp->sg_osdescs, idx2, osp)) {
3919         shdr *shdr = osp->os_shdr;
3920
3921         p_align = 0;
3922         if (shdr->sh_addralign > p_align)
3923             p_align = shdr->sh_addralign;
3924
3925         offset = (Off)S_ROUND(offset, shdr->sh_addralign);
3926         offset += shdr->sh_size;
3927
3928         if (shdr->sh_type != SHT_NOBITS) {
3929             if (nobits) {
3930                 ld_printf(ofl, ERR_FATAL,
3931                     MSG_INTL(MSG_UPD_NOBITS));
3932                 return (S_ERROR);
3933             }
3934             phdr->p_filesz = offset - phdr->p_offset;
3935             } else if ((sgp->sg_flags & FLG_SG_PHREQ) == 0)
3936                 nobits = TRUE;
3937         }
3938     }
3939     phdr->p_memsz = offset - hshdr->sh_offset;
3940
3941     /*
3942     * If this is the first loadable segment of a dynamic object,
3943     * or an interpreter has been specified (a static object built
3944     * with an interpreter will still be given a PT_HDR entry), then
3945     * compensate for the elf header and program header array. Both
3946     * of these are actually part of the loadable segment as they
3947     * may be inspected by the interpreter. Adjust the segments
3948     * size and offset accordingly.
3949     */
3950     if ((phdr == NULL) && (phdr->p_type == PT_LOAD) &&
3951         ((ofl->ofl_osinterp) || (flags & FLG_OF_DYNAMIC)) &&
3952         (!(ofl->ofl_dtflags_1 & DF_1_NOHDR))) {
3953         size = (Addr)S_ROUND((phdrsz + ehdrsz),
3954             hshdr->sh_addralign);
3955         phdr->p_offset -= size;
3956         phdr->p_filesz += size;
3957         phdr->p_memsz += size;
3958     }
3959
3960     /*
3961     * If segment size symbols are required (specified via a
3962     * mapfile) update their value.

```

```

3962     */
3963     for (APLIST_TRAVERSE(sgp->sg_sizesym, idx2, sdp))
3964         sdp->sd_sym->st_value = phdr->p_memsz;
3965
3966     /*
3967     * If no file content has been assigned to this segment (it
3968     * only contains no-bits sections), then reset the offset for
3969     * consistency.
3970     */
3971     if (phdr->p_filesz == 0)
3972         phdr->p_offset = 0;
3973
3974     /*
3975     * If a virtual address has been specified for this segment
3976     * from a mapfile use it and make sure the previous segment
3977     * does not run into this segment.
3978     */
3979     if (phdr->p_type == PT_LOAD) {
3980         if ((sgp->sg_flags & FLG_SG_P_VADDR)) {
3981             if (_phdr && (vaddr > phdr->p_vaddr) &&
3982                 (phdr->p_type == PT_LOAD))
3983                 ld_printf(ofl, ERR_WARNING,
3984                     MSG_INTL(MSG_UPD_SEGOVERLAP),
3985                     ofl->ofl_name, EC_ADDR(vaddr),
3986                     sgp->sg_name,
3987                     EC_ADDR(phdr->p_vaddr));
3988             vaddr = phdr->p_vaddr;
3989             phdr->p_align = 0;
3990         } else {
3991             vaddr = phdr->p_vaddr =
3992                 (Addr)S_ROUND(vaddr, phdr->p_align);
3993         }
3994     }
3995
3996     /*
3997     * Adjust the address offset and p_align if needed.
3998     */
3999     if (((sgp->sg_flags & FLG_SG_P_VADDR) == 0) &&
4000         ((ofl->ofl_dtflags_1 & DF_1_NOHDR) == 0)) {
4001         if (phdr->p_align != 0)
4002             vaddr += phdr->p_offset % phdr->p_align;
4003         else
4004             vaddr += phdr->p_offset;
4005         phdr->p_vaddr = vaddr;
4006     }
4007
4008     /*
4009     * If an interpreter is required set the virtual address of the
4010     * PT_PHDR program header now that we know the virtual address
4011     * of the loadable segment that contains it. Update the
4012     * PT_SUNWCAP header similarly.
4013     */
4014     if ((phdr == NULL) && (phdr->p_type == PT_LOAD)) {
4015         _phdr = phdr;
4016
4017         if ((ofl->ofl_dtflags_1 & DF_1_NOHDR) == 0) {
4018             if (ofl->ofl_osinterp)
4019                 ofl->ofl_phdr[0].p_vaddr =
4020                     vaddr + ehdrsz;
4021         }
4022
4023         /*
4024         * Finally, if we're creating a dynamic object
4025         * (or a static object in which an interpreter
4026         * is specified) update the vaddr to reflect
4027         * the address of the first section within this
4028         * segment.

```

```

4028     */
4029     if ((ofl->ofl_osinterp) ||
4030         (flags & FLG_OF_DYNAMIC))
4031         vaddr += size;
4032     } else {
4033     /*
4034     * If the DF_1_NOHDR flag was set, and an
4035     * interpreter is being generated, the PT_PHDR
4036     * will not be part of any loadable segment.
4037     */
4038     if (ofl->ofl_osinterp) {
4039         ofl->ofl_phdr[0].p_vaddr = 0;
4040         ofl->ofl_phdr[0].p_memsz = 0;
4041         ofl->ofl_phdr[0].p_flags = 0;
4042     }
4043     }
4044 }
4045
4046 /*
4047 * Ensure the ELF entry point defaults to zero. Typically, this
4048 * value is overridden in update_oehdr() to one of the standard
4049 * entry points. Historically, this default was set to the
4050 * address of first executable section, but this has since been
4051 * found to be more confusing than it is helpful.
4052 */
4053 ehdr->e_entry = 0;
4054
4055 DBG_CALL(Dbg_seg_entry(ofl, segndx, sgp));
4056
4057 /*
4058 * Traverse the output section descriptors for this segment so
4059 * that we can update the section headers addresses. We've
4060 * calculated the virtual address of the initial section within
4061 * this segment, so each successive section can be calculated
4062 * based on their offsets from each other.
4063 */
4064 secndx = 0;
4065 hshdr = 0;
4066 for (APLIST_TRAVERSE(sgp->sg_osdescs, idx2, osp)) {
4067     Shdr *shdr = osp->os_shdr;
4068
4069     if (shdr->sh_link)
4070         shdr->sh_link = translate_link(ofl, osp,
4071                                     shdr->sh_link, MSG_INTL(MSG_FIL_INVSHLINK));
4072
4073     if (shdr->sh_info && (shdr->sh_flags & SHF_INFO_LINK))
4074         shdr->sh_info = translate_link(ofl, osp,
4075                                     shdr->sh_info, MSG_INTL(MSG_FIL_INVSHINFO));
4076
4077     if (!(flags & FLG_OF_RELOBJ) &&
4078         (phdr->p_type == PT_LOAD)) {
4079         if (hshdr)
4080             vaddr += (shdr->sh_offset -
4081                     hshdr->sh_offset);
4082
4083         shdr->sh_addr = vaddr;
4084         hshdr = shdr;
4085     }
4086
4087     DBG_CALL(Dbg_seg_os(ofl, osp, secndx));
4088     secndx++;
4089 }
4090
4091 /*
4092 * Establish the virtual address of the end of the last section
4093 * in this segment so that the next segments offset can be

```

```

4094     * calculated from this.
4095     */
4096     if (hshdr)
4097         vaddr += hshdr->sh_size;
4098
4099     /*
4100     * Output sections for this segment complete. Adjust the
4101     * virtual offset for the last sections size, and make sure we
4102     * haven't exceeded any maximum segment length specification.
4103     */
4104     if ((sgp->sg_length != 0) && (sgp->sg_length < phdr->p_memsz)) {
4105         ld_eprintf(ofl, ERR_FATAL, MSG_INTL(MSG_UPD_LARGSIZE),
4106                 ofl->ofl_name, sgp->sg_name,
4107                 EC_XWORD(phdr->p_memsz), EC_XWORD(sgp->sg_length));
4108         return (S_ERROR);
4109     }
4110
4111     if (phdr->p_type == PT_NOTE) {
4112         phdr->p_vaddr = 0;
4113         phdr->p_paddr = 0;
4114         phdr->p_align = 0;
4115         phdr->p_memsz = 0;
4116     }
4117
4118     if ((phdr->p_type != PT_NULL) && !(flags & FLG_OF_RELOBJ))
4119         ofl->ofl_phdr[phdrndx++] = *phdr;
4120 }
4121
4122 /*
4123 * Update any new output sections. When building the initial output
4124 * image, a number of sections were created but left uninitialized (eg.
4125 * .dynsym, .dynstr, .symtab, .symtab, etc.). Here we update these
4126 * sections with the appropriate data. Other sections may still be
4127 * modified via reloc_process().
4128 *
4129 * Copy the interpreter name into the .interp section.
4130 */
4131 if (ofl->ofl_interp)
4132     (void) strcpy((char *)ofl->ofl_osinterp->os_outdata->d_buf,
4133                 ofl->ofl_interp);
4134
4135 /*
4136 * Update the .shstrtab, .strtab and .dynstr sections.
4137 */
4138 update_osstrtab(ofl->ofl_osshstrtab, ofl->ofl_shdrsttab, 0);
4139 update_osstrtab(ofl->ofl_osstrtab, ofl->ofl_strtab, 0);
4140 update_osstrtab(ofl->ofl_osdynstr, ofl->ofl_dynstrtab, DYNSTR_EXTRA_PAD);
4141
4142 /*
4143 * Build any output symbol tables, the symbols information is copied
4144 * and updated into the new output image.
4145 */
4146 if ((etext = update_osym(ofl)) == (Addr)S_ERROR)
4147     return (S_ERROR);
4148
4149 /*
4150 * If we have an PT_INTERP phdr, update it now from the associated
4151 * section information.
4152 */
4153 if (intpsgp) {
4154     Phdr *phdr = &(intpsgp->sg_phdr);
4155     Shdr *shdr = ofl->ofl_osinterp->os_shdr;
4156
4157     phdr->p_vaddr = shdr->sh_addr;
4158     phdr->p_offset = shdr->sh_offset;
4159     phdr->p_memsz = phdr->p_filesz = shdr->sh_size;

```

```

4160     phdr->p_flags = PF_R;
4162     DBG_CALL(Dbg_seg_entry(ofl, intpsndx, intpsgp));
4163     ofl->ofl_phdr[intppndx] = *phdr;
4164 }
4166 /*
4167  * If we have a PT_SUNWDTTRACE phdr, update it now with the address of
4168  * the symbol. It's only now been updated via update_sym().
4169  */
4170 if (dtracesgp) {
4171     Phdr      *aphdr, *phdr = &(dtracesgp->sg_phdr);
4172     Sym_desc  *sdp = ofl->ofl_dtracesym;
4174     phdr->p_vaddr = sdp->sd_sym->st_value;
4175     phdr->p_memsz = sdp->sd_sym->st_size;
4177     /*
4178     * Take permissions from the segment that the symbol is
4179     * associated with.
4180     */
4181     aphdr = &sdp->sd_isc->is_osdesc->os_sgdesc->sg_phdr;
4182     assert(aphdr);
4183     phdr->p_flags = aphdr->p_flags;
4185     DBG_CALL(Dbg_seg_entry(ofl, dtracesndx, dtracesgp));
4186     ofl->ofl_phdr[dtracepndx] = *phdr;
4187 }
4189 /*
4190  * If we have a PT_SUNWCAP phdr, update it now from the associated
4191  * section information.
4192  */
4193 if (capsgp) {
4194     Phdr      *phdr = &(capsgp->sg_phdr);
4195     Shdr      *shdr = ofl->ofl_oscapp->os_shdr;
4197     phdr->p_vaddr = shdr->sh_addr;
4198     phdr->p_offset = shdr->sh_offset;
4199     phdr->p_memsz = phdr->p_filesz = shdr->sh_size;
4200     phdr->p_flags = PF_R;
4202     DBG_CALL(Dbg_seg_entry(ofl, capsndx, capsgp));
4203     ofl->ofl_phdr[capppndx] = *phdr;
4204 }
4206 /*
4207  * Update the GROUP sections.
4208  */
4209 if (update_ogroup(ofl) == S_ERROR)
4210     return (S_ERROR);
4212 /*
4213  * Update Move Table.
4214  */
4215 if (ofl->ofl_osmove || ofl->ofl_isparexpn)
4216     update_move(ofl);
4218 /*
4219  * Build any output headers, version information, dynamic structure and
4220  * syminfo structure.
4221  */
4222 if (update_oehdr(ofl) == S_ERROR)
4223     return (S_ERROR);
4224 if (!(flags & FLG_OF_NOVERSEC)) {
4225     if ((flags & FLG_OF_VERDEF) &&

```

```

4226         (update_overdef(ofl) == S_ERROR))
4227         return (S_ERROR);
4228     if ((flags & FLG_OF_VERNEED) &&
4229         (update_overneed(ofl) == S_ERROR))
4230         return (S_ERROR);
4231     if (flags & (FLG_OF_VERNEED | FLG_OF_VERDEF))
4232         update_oversym(ofl);
4233 }
4234 if (flags & FLG_OF_DYNAMIC) {
4235     if (update_odynamic(ofl) == S_ERROR)
4236         return (S_ERROR);
4237 }
4238 if (ofl->ofl_ossyminfo) {
4239     if (update_osyminfo(ofl) == S_ERROR)
4240         return (S_ERROR);
4241 }
4243 /*
4244  * Update capabilities information if required.
4245  */
4246 if (ofl->ofl_oscapp)
4247     update_oscapp(ofl);
4248 if (ofl->ofl_oscappinfo)
4249     update_oscappinfo(ofl);
4251 /*
4252  * Sanity test: the first and last data byte of a string table
4253  * must be NULL.
4254  */
4255 assert((ofl->ofl_osshstrtab == NULL) ||
4256        (((char *)ofl->ofl_osshstrtab->os_outdata->d_buf) == '\0'));
4257 assert((ofl->ofl_osstrtab == NULL) ||
4258        (((char *)ofl->ofl_osstrtab->os_outdata->d_buf) +
4259         ofl->ofl_osstrtab->os_outdata->d_size - 1) == '\0'));
4261 assert((ofl->ofl_osstrtab == NULL) ||
4262        (((char *)ofl->ofl_osstrtab->os_outdata->d_buf) == '\0'));
4263 assert((ofl->ofl_osstrtab == NULL) ||
4264        (((char *)ofl->ofl_osstrtab->os_outdata->d_buf) +
4265         ofl->ofl_osstrtab->os_outdata->d_size - 1) == '\0'));
4267 assert((ofl->ofl_osdynstr == NULL) ||
4268        (((char *)ofl->ofl_osdynstr->os_outdata->d_buf) == '\0'));
4269 assert((ofl->ofl_osdynstr == NULL) ||
4270        (((char *)ofl->ofl_osdynstr->os_outdata->d_buf) +
4271         ofl->ofl_osdynstr->os_outdata->d_size - DYNSTR_EXTRA_PAD - 1) ==
4272         '\0'));
4274 /*
4275  * Emit Strtab diagnostics.
4276  */
4277 DBG_CALL(Dbg_sec_strtab(ofl->ofl_lml, ofl->ofl_osshstrtab,
4278                        ofl->ofl_shdrsttab));
4279 DBG_CALL(Dbg_sec_strtab(ofl->ofl_lml, ofl->ofl_osstrtab,
4280                        ofl->ofl_strtab));
4281 DBG_CALL(Dbg_sec_strtab(ofl->ofl_lml, ofl->ofl_osdynstr,
4282                        ofl->ofl_dynstrtab));
4284 /*
4285  * Initialize the section headers string table index within the elf
4286  * header.
4287  */
4288 /* LINTED */
4289 if ((shscnndx = elf_ndxscn(ofl->ofl_osshstrtab->os_scn)) <
4290     SHN_LORESERVE) {
4291     ofl->ofl_nehdr->e_shstrndx =

```

```
4292             /* LINTED */
4293             (Half)shscndx;
4294     } else {
4295         /*
4296          * If the STRTAB section index doesn't fit into
4297          * e_shstrndx, then we store it in 'shdr[0].st_link'.
4298          */
4299         Elf_Scn *scn;
4300         Shdr *shdr0;
4301
4302         if ((scn = elf_getscn(ofl->ofl_elf, 0)) == NULL) {
4303             ld_eprintf(ofl, ERR_elf, MSG_INTL(MSG_elf_GETSCN),
4304                       ofl->ofl_name);
4305             return (S_ERROR);
4306         }
4307         if ((shdr0 = elf_getshdr(scn)) == NULL) {
4308             ld_eprintf(ofl, ERR_elf, MSG_INTL(MSG_elf_GETSHDR),
4309                       ofl->ofl_name);
4310             return (S_ERROR);
4311         }
4312         ofl->ofl_nehdr->e_shstrndx = SHN_XINDEX;
4313         shdr0->sh_link = shscndx;
4314     }
4315     return ((uintptr_t)etext);
4316 }
4317 }
```

new/usr/src/cmd/svc/dtd/service_bundle.dtd.1

1

```
*****
26796 Thu Jun 30 22:01:36 2016
new/usr/src/cmd/svc/dtd/service_bundle.dtd.1
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!--
3 Copyright (c) 2004, 2010, Oracle and/or its affiliates. All rights reserved.

5 CDDL HEADER START

7 The contents of this file are subject to the terms of the
8 Common Development and Distribution License (the "License").
9 You may not use this file except in compliance with the License.

11 You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
12 or http://www.opensolaris.org/os/licensing.
13 See the License for the specific language governing permissions
14 and limitations under the License.

16 When distributing Covered Code, include this CDDL HEADER in each
17 file and include the License file at usr/src/OPENSOLARIS.LICENSE.
18 If applicable, add the following below this CDDL HEADER, with the
19 fields enclosed by brackets "[]" replaced with your own identifying
20 information: Portions Copyright [yyyy] [name of copyright owner]

22 CDDL HEADER END
23 -->

25 <!--
26 Service description DTD

28 Most attributes are string values (or an individual string from a
29 restricted set), but attributes with a specific type requirement are
30 noted in the comment describing the element.
31 -->

33 <!--
34 XInclude support

36 A series of service bundles may be composed via the xi:include tag.
37 smf(5) tools enforce that all bundles be of the same type.
38 -->

40 <!--
41 These entities are used for the property, propval and property_group
42 elements, that require type attributes for manifest, while for profiles
43 the type attributes are only implied.
44 -->

46 <!ENTITY % profile "IGNORE">
47 <!ENTITY % manifest "INCLUDE">

49 <!ELEMENT xi:include
50 (xi:fallback)
51 >
52 <!ATTLIST xi:include
53 href CDATA #REQUIRED
54 parse (xml|text) "xml"
55 encoding CDATA #IMPLIED
56 xmlns:xi CDATA #FIXED "http://www.w3.org/2001/XInclude"
57 >
```

new/usr/src/cmd/svc/dtd/service_bundle.dtd.1

2

```
59 <!ELEMENT xi:fallback
60 ANY
61 >
62 <!ATTLIST xi:fallback
63 xmlns:xi CDATA #FIXED "http://www.w3.org/2001/XInclude"
64 >

66 <!--
67 stability

69 This element associates an SMI stability level with the parent
70 element. See attributes(5) for an explanation of interface
71 stability levels.

73 Its attribute is

75 value The stability level of the parent element.
76 -->

78 <!ELEMENT stability EMPTY>

80 <!ATTLIST stability
81 value ( Standard | Stable | Evolving | Unstable |
82 External | Obsolete ) #REQUIRED >

84 <!-- Property value lists -->

86 <!--
87 value_node

89 This element represents a single value within any of the typed
90 property value lists.

92 Its attribute is

94 value The value for this node in the list.
95 -->

97 <!ELEMENT value_node EMPTY>

99 <!ATTLIST value_node
100 value CDATA #REQUIRED>

102 <!--
103 count_list
104 integer_list
105 opaque_list
106 host_list
107 hostname_list
108 net_address_list
109 net_address_v4_list
110 net_address_v6_list
111 time_list
112 astring_list
113 ustring_list
114 boolean_list
115 fmri_list
116 uri_list

118 These elements represent the typed lists of values for a property.
119 Each contains one or more value_node elements representing each
120 value on the list.

122 None of these elements has attributes.
123 -->
```



```

125 <!ELEMENT count_list
126     ( value_node+ )>

128 <!ATTLIST count_list>

130 <!ELEMENT integer_list
131     ( value_node+ )>

133 <!ATTLIST integer_list>

135 <!ELEMENT opaque_list
136     ( value_node+ )>

138 <!ATTLIST opaque_list>

140 <!ELEMENT host_list
141     ( value_node+ )>

143 <!ATTLIST host_list>

145 <!ELEMENT hostname_list
146     ( value_node+ )>

148 <!ATTLIST hostname_list>

150 <!ELEMENT net_address_list
151     ( value_node+ )>

153 <!ATTLIST net_address_list>

155 <!ELEMENT net_address_v4_list
156     ( value_node+ )>

158 <!ATTLIST net_address_v4_list>

160 <!ELEMENT net_address_v6_list
161     ( value_node+ )>

163 <!ATTLIST net_address_v6_list>

165 <!ELEMENT time_list
166     ( value_node+ )>

168 <!ATTLIST time_list>

170 <!ELEMENT astring_list
171     ( value_node+ )>

173 <!ATTLIST astring_list>

175 <!ELEMENT ustring_list
176     ( value_node+ )>

178 <!ATTLIST ustring_list>

180 <!ELEMENT boolean_list
181     ( value_node+ )>

183 <!ATTLIST boolean_list>

185 <!ELEMENT fmri_list
186     ( value_node+ )>

188 <!ATTLIST fmri_list>

190 <!ELEMENT uri_list

```

```

191     ( value_node+ )>

193 <!ATTLIST uri_list>

195 <!-- Properties and property groups -->

197 <!--
198     property

200     This element is for a singly or multiply valued property within a
201     property group. It contains an appropriate value list element,
202     which is expected to be consistent with the type attribute.

204     Its attributes are

206         name     The name of this property.

208         type     The data type for this property.

210         override These values should replace values already in the
211                 repository.
212 -->

214 <![%profile;[
215 <!ELEMENT property
216     ( count_list | integer_list | opaque_list | host_list | hostname_list |
217     net_address_list | net_address_v4_list | net_address_v6_list |
218     time_list | astring_list | ustring_list | boolean_list | fmri_list |
219     uri_list )? >

221 <!ATTLIST property
222     name             CDATA #REQUIRED
223     type             ( count | integer | opaque | host | hostname |
224                     net_address | net_address_v4 | net_address_v6 | time |
225                     astring | ustring | boolean | fmri | uri ) #IMPLIED
226     override        ( true | false ) "false" >
227 ]]>
228
229 <![%manifest;[
230 <!ELEMENT property
231     ( count_list | integer_list | opaque_list | host_list | hostname_list |
232     net_address_list | net_address_v4_list | net_address_v6_list |
233     time_list | astring_list | ustring_list | boolean_list | fmri_list |
234     uri_list )? >

236 <!ATTLIST property
237     name             CDATA #REQUIRED
238     type             ( count | integer | opaque | host | hostname |
239                     net_address | net_address_v4 | net_address_v6 | time |
240                     astring | ustring | boolean | fmri | uri ) #REQUIRED
241     override        ( true | false ) "false" >
242 ]]>

244 <!--
245     propval

247     This element is for a singly valued property within a property
248     group. List-valued properties must use the property element above.

250     Its attributes are

252         name     The name of this property.

254         type     The data type for this property.

256         value    The value for this property. Must match type

```

```

257             restriction of type attribute.

259     override This value should replace any values already in the
260             repository.
261 -->

263 <![%profile;[
264 <!ELEMENT propval EMPTY>

266 <!ATTLIST propval
267     name          CDATA #REQUIRED
268     type          ( count | integer | opaque | host | hostname |
269                 net_address | net_address_v4 | net_address_v6 | time |
270                 astring | ustring | boolean | fmri | uri ) #IMPLIED
271     value         CDATA #REQUIRED
272     override      ( true | false ) "false" >
273 ]]>

275 <![%manifest;[
276 <!ELEMENT propval EMPTY>

278 <!ATTLIST propval
279     name          CDATA #REQUIRED
280     type          ( count | integer | opaque | host | hostname |
281                 net_address | net_address_v4 | net_address_v6 | time |
282                 astring | ustring | boolean | fmri | uri ) #REQUIRED
283     value         CDATA #REQUIRED
284     override      ( true | false ) "false" >
285 ]]>

287 <!--
288     property_group

290     This element is for a set of related properties on a service or
291     instance. It contains an optional stability element, as well as
292     zero or more property-containing elements.

294     Its attributes are

296     name      The name of this property group.

298     type      A category for this property group. Groups of type
299               "framework", "implementation" or "template" are primarily
300               of interest to the service management facility, while
301               groups of type "application" are expected to be only of
302               interest to the service to which this group is attached.
303               Other types may be introduced using the service symbol
304               namespace conventions.

306     delete   If in the repository, this property group should be removed.
307 -->

309 <![%profile;[
310 <!ELEMENT property_group
311     ( stability?, ( propval | property ) * ) >

313 <!ATTLIST property_group
314     name          CDATA #REQUIRED
315     type          CDATA #IMPLIED
316     delete        ( true | false ) "false" >
317 ]]>

319 <![%manifest;[
320 <!ELEMENT property_group
321     ( stability?, ( propval | property ) * ) >

```

```

323 <!ATTLIST property_group
324     name          CDATA #REQUIRED
325     type          CDATA #REQUIRED
326     delete        ( true | false ) "false" >
327 ]]>

329 <!--
330     service_fmri

332     This element defines a reference to a service FMRI (for either a
333     service or an instance).

335     Its attribute is

337     value      The FMRI.
338 -->

340 <!ELEMENT service_fmri EMPTY>

342 <!ATTLIST service_fmri
343     value         CDATA #REQUIRED>

345 <!-- Dependencies -->

347 <!--
348     dependency

350     This element identifies a group of FMRIs upon which the service is
351     in some sense dependent. Its interpretation is left to the
352     restarter to which a particular service instance is delegated. It
353     contains a group of service FMRIs, as well as a block of properties.

355     Its attributes are

357     name        The name of this dependency.

359     grouping    The relationship between the various FMRIs grouped
360                 here; "require_all" of the FMRIs to be online, "require_any"
361                 of the FMRIs to be online, or "exclude_all" of the FMRIs
362                 from being online or in maintenance for the dependency to
363                 be satisfied. "optional_all" dependencies are satisfied
364                 when all of the FMRIs are either online or unable to come
365                 online (because they are disabled, misconfigured, or one
366                 of their dependencies is unable to come online).

368     restart_on The type of events from the FMRIs that the service should
369                 be restarted for. "error" restarts the service if the
370                 dependency is restarted due to hardware fault. "restart"
371                 restarts the service if the dependency is restarted for
372                 any reason, including hardware fault. "refresh" restarts
373                 the service if the dependency is refreshed or restarted for
374                 any reason. "none" will never restart the service due to
375                 dependency state changes.

377     type        The type of dependency: on another service ('service'), on
378                 a filesystem path ('path'), or another dependency type.

380     delete      This dependency should be deleted.
381 -->

383 <!ELEMENT dependency
384     ( service_fmri*, stability?, ( propval | property ) * ) >

386 <!ATTLIST dependency
387     name          CDATA #REQUIRED
388     grouping      ( require_all | require_any | exclude_all |

```

```

389         optional_all ) #REQUIRED
390     restart_on    ( error | restart | refresh | none ) #REQUIRED
391     type          CDATA #REQUIRED
392     delete        ( true | false ) "false" >
394 <!-- Dependents -->
396 <!--
397     dependent
399     This element identifies a service which should depend on this service. It
400     corresponds to a dependency in the named service. The grouping and type
401     attributes of that dependency are implied to be "require_all" and
402     "service", respectively.
404     Its attributes are
406         name      The name of the dependency property group to create in the
407         dependent entity.
409         grouping  The grouping relationship of the dependency property
410         group to create in the dependent entity. See "grouping"
411         attribute on the dependency element.
413         restart_on The type of events from this service that the named service
414         should be restarted for.
416         delete    True if this dependent should be deleted.
418         override  Whether to replace an existing dependent of the same name.
420 -->
422 <!ELEMENT dependent
423     ( service_fmri, stability?, ( propval | property ) * ) >
425 <!ATTLIST dependent
426     name          CDATA #REQUIRED
427     grouping      ( require_all | require_any | exclude_all |
428     optional_all) #REQUIRED
429     restart_on    ( error | restart | refresh | none) #REQUIRED
430     delete        ( true | false ) "false"
431     override      ( true | false ) "false" >
433 <!-- Method execution context, security profile, and credential definitions -->
435 <!--
436     envvar
438     An environment variable. It has two attributes:
440         name      The name of the environment variable.
441         value     The value of the environment variable.
442 -->
444 <!ELEMENT envvar EMPTY>
446 <!ATTLIST envvar
447     name          CDATA #REQUIRED
448     value         CDATA #REQUIRED >
450 <!--
451     method_environment
453     This element defines the environment for a method. It has no
454     attributes, and one or more envvar child elements.

```

```

455 -->
457 <!ELEMENT method_environment ( envvar+ ) >
459 <!ATTLIST method_environment>
461 <!--
462     method_profile
464     This element indicates which exec_attr(5) profile applies to the
465     method context being defined.
467     Its attribute is
469         name      The name of the profile.
470 -->
472 <!ELEMENT method_profile EMPTY>
474 <!ATTLIST method_profile
475     name          CDATA #REQUIRED >
477 <!--
478     method_credential
480     This element specifies credential attributes for the execution
481     method to use.
483     Its attributes are
485         user      The user ID, in numeric or text form.
487         group     The group ID, in numeric or text form. If absent or
488         ":default", the group associated with the user in the
489         passwd database.
491         supp_groups Supplementary group IDs to be associated with the
492         method, separated by commas or spaces. If absent or
493         ":default", initgroups(3C) will be used.
495         privileges An optional string specifying the privilege set.
497         limit_privileges An optional string specifying the limit
498         privilege set.
499 -->
501 <!ELEMENT method_credential EMPTY>
503 <!ATTLIST method_credential
504     user          CDATA #REQUIRED
505     group         CDATA #IMPLIED
506     supp_groups  CDATA #IMPLIED
507     privileges   CDATA #IMPLIED
508     limit_privileges CDATA #IMPLIED >
510 <!--
511     method_context
513     This element combines credential and resource management attributes
514     for execution methods. It may contain a method_environment, or
515     a method_profile or method_credential element.
517     Its attributes are
519         working_directory The home directory to launch the method from.
520         ":default" can be used as a token to indicate use of the

```

```

521         user specified by the credential or profile specified.

523     project The project ID, in numeric or text form. ":default" can
524             be used as a token to indicate use of the project
525             identified by getdefaultproj(3PROJECT) for the non-root
526             user specified by the credential or profile specified.
527             If the user is root, ":default" designates the project
528             the restarter is running in.

530     resource_pool The resource pool name to launch the method on.
531                  ":default" can be used as a token to indicate use of the
532                  pool specified in the project(4) entry given in the
533                  "project" attribute above.
534 -->
535 <!ELEMENT method_context
536         ( (method_profile | method_credential)?, method_environment? ) >

538 <!ATTLIST method_context
539     security_flags          CDATA #IMPLIED
540 #endif /* ! codereview */
541     working_directory      CDATA #IMPLIED
542     project                 CDATA #IMPLIED
543     resource_pool          CDATA #IMPLIED >

545 <!-- Restarter delegation, methods, and monitors -->

547 <!--
548     exec_method

550     This element describes one of the methods used by the designated
551     restarter to act on the service instance. Its interpretation is
552     left to the restarter to which a particular service instance is
553     delegated. It contains a set of attributes, an optional method
554     context, and an optional stability element for the optional
555     properties that can be included.

557     Its attributes are

559     type      The type of method, either "method" or "monitor".

561     name      Name of this execution method. The method names are
562               usually a defined interface of the restarter to which an
563               instance of this service is delegated.

565     exec      The string identifying the action to take. For
566               svc.startd(1M), this is a string suitable to pass to
567               exec(2).

569     timeout_seconds [integer] Duration, in seconds, to wait for this
570               method to complete. A '0' or '-1' denotes an infinite
571               timeout.

573     delete    If in the repository, the property group for this method
574               should be removed.
575 -->

577 <!ELEMENT exec_method
578         ( method_context?, stability?, ( propval | property )* ) >

580 <!ATTLIST exec_method
581     type          ( method | monitor ) #REQUIRED
582     name          CDATA #REQUIRED
583     exec          CDATA #REQUIRED
584     timeout_seconds CDATA #REQUIRED
585     delete        ( true | false ) "false" >

```

```

587 <!--
588     restarter

590     A flag element identifying the restarter to which this service or
591     service instance is delegated. Contains the FMRI naming the
592     delegated restarter.

594     This element has no attributes.
595 -->

597 <!ELEMENT restarter
598         ( service_fmri ) >

600 <!ATTLIST restarter>

602 <!--
603     Templates
604 -->

606 <!--
607     doc_link

609     The doc_link relates a resource described by the given URI to the
610     service described by the containing template. The resource is
611     expected to be a documentation or elucidatory reference of some
612     kind.

614     Its attributes are

616     name      A label for this resource.

618     uri       A URI to the resource.
619 -->

621 <!ELEMENT doc_link EMPTY>

623 <!ATTLIST doc_link
624     name          CDATA #REQUIRED
625     uri           CDATA #REQUIRED >

627 <!--
628     manpage

630     The manpage element connects the reference manual page to the
631     template's service.

633     Its attributes are

635     title     The manual page title.

637     section   The manual page's section.

639     manpath   The MANPATH environment variable, as described in man(1)
640               that is required to reach the named manual page
641 -->

643 <!ELEMENT manpage EMPTY>

645 <!ATTLIST manpage
646     title          CDATA #REQUIRED
647     section        CDATA #REQUIRED
648     manpath        CDATA " :default" >

650 <!--
651     documentation

```

```

653 The documentation element groups an arbitrary number of doc_link
654 and manpage references.

656 It has no attributes.
657 -->

659 <!ELEMENT documentation
660 ( doc_link | manpage )* >

662 <!ATTLIST documentation>

664 <!--
665 loctext

667 The loctext element is a container for localized text.

669 Its sole attribute is

671 xml:lang The name of the locale, in the form accepted by LC_ALL,
672 etc. See locale(5).
673 -->
674 <!ELEMENT loctext
675 (#PCDATA) >

677 <!ATTLIST loctext
678 xml:lang CDATA #REQUIRED >

680 <!--
681 description

683 The description holds a set of potentially longer, localized strings that
684 consist of a short description of the service.

686 The description has no attributes.
687 -->
688 <!ELEMENT description
689 ( loctext+ ) >

691 <!ATTLIST description>

693 <!--
694 common_name

696 The common_name holds a set of short, localized strings that
697 represent a well-known name for the service in the given locale.

699 The common_name has no attributes.
700 -->
701 <!ELEMENT common_name
702 ( loctext+ ) >

704 <!ATTLIST common_name>

706 <!--
707 units

709 The units a numerical property is expressed in.
710 -->
712 <!ELEMENT units
713 ( loctext+ ) >

715 <!ATTLIST units>

717 <!--
718 visibility

```

```

720 Expresses how a property is typically accessed. This isn't
721 intended as access control, but as an indicator as to how a
722 property is used.

724 Its attributes are:

726 value 'hidden', 'readonly', or 'readwrite' indicating that
727 the property should be hidden from the user, shown but
728 read-only, or modifiable.
729 -->

731 <!ELEMENT visibility EMPTY>

733 <!ATTLIST visibility
734 value ( hidden | readonly | readwrite ) #REQUIRED >

736 <!--
737 value

739 Describes a legal value for a property value, and optionally contains a
740 human-readable name and description for the specified property
741 value.

743 Its attributes are:

745 name A string representation of the value.
746 -->

748 <!ELEMENT value
749 ( common_name?, description? ) >

751 <!ATTLIST value
752 name CDATA #REQUIRED >

754 <!--
755 values

757 Human-readable names and descriptions for valid values of a property.
758 -->

760 <!ELEMENT values
761 (value+ ) >

763 <!ATTLIST values>

765 <!--
766 cardinality

768 Places a constraint on the number of values the property can take
769 on.

771 Its attributes are:
772 min minimum number of values
773 max maximum number of values

775 Both attributes are optional. If min is not specified, it defaults to
776 0. If max is not specified it indicates an unlimited number of values.
777 If neither is specified this indicates 0 or more values.
778 -->

780 <!ELEMENT cardinality EMPTY>

782 <!ATTLIST cardinality
783 min CDATA "0"
784 max CDATA "18446744073709551615">

```

```

786 <!--
787   internal_separators

789   Indicates the separators used within a property's value used to
790   separate the actual values. Used in situations where multiple
791   values are packed into a single property value instead of using a
792   multi-valued property.
793 -->

795 <!ELEMENT internal_separators
796   (#PCDATA) >

798 <!ATTLIST internal_separators>

800 <!--
801   range

803   Indicates a range of possible integer values.

805   Its attributes are:

807       min      The minimum value of the range (inclusive).
808       max      The maximum value of the range (inclusive).
809 -->

811 <!ELEMENT range EMPTY>

813 <!ATTLIST range
814   min      CDATA #REQUIRED
815   max      CDATA #REQUIRED >

817 <!--
818   constraints

820   Provides a set of constraints on the values a property can take on.
821 -->

823 <!ELEMENT constraints
824   ( value*, range* ) >
825 <!ATTLIST constraints>

827 <!--
828   include_values

830   Includes an entire set of values in the choices block.

832   Its attributes are:

834       type     Either "constraints" or "values", indicating an
835               inclusion of all values allowed by the property's
836               constraints or all values for which there are
837               human-readable names and descriptions, respectively.
838 -->

840 <!ELEMENT include_values EMPTY>

842 <!ATTLIST include_values
843   type     ( constraints | values ) #REQUIRED >

845 <!--
846   choices

848   Provides a set of common choices for the values a property can take
849   on. Useful in those cases where the possibilities are unenumerable
850   or merely inconveniently legion, and a manageable subset is desired

```

```

851   for presentation in a user interface.
852 -->

854 <!ELEMENT choices
855   ( value*, range*, include_values* ) >

857 <!ATTLIST choices>

859 <!--
860   prop_pattern

863   The prop_pattern describes one property of the enclosing property group
864   pattern.

866   Its attributes are:

868       name     The property's name.
869       type     The property's type.
870       required  If the property group is present, this property is required.
871

873       type can be omitted if required is false.
874 -->

876 <!ELEMENT prop_pattern
877   ( common_name?, description?, units?, visibility?, cardinality?,
878     internal_separators?, values?, constraints?, choices? ) >

880 <!ATTLIST prop_pattern
881   name      CDATA #REQUIRED
882   type      ( count | integer | opaque | host | hostname |
883             net_address | net_address_v4 | net_address_v6 | time |
884             astring | ustring | boolean | fmri | uri ) #IMPLIED
885   required  ( true | false ) "false" >

887 <!--
888   pg_pattern

890   The pg_pattern describes one property group.
891   Depending on the element's attributes, these descriptions may apply
892   to just the enclosing service/instance, instances of the enclosing
893   service, delegates of the service (assuming it is a restarter), or
894   all services.

896   Its attributes are:

898       name     The property group's name. If not specified, it
899               matches all property groups with the specified type.
900       type     The property group's type. If not specified, it
901               matches all property groups with the specified name.
902       required  If the property group is required.
903       target   The scope of the pattern, which may be all, delegate,
904               instance, or this. 'all' is reserved for framework use
905               and applies the template to all services on the system.
906               'delegate' is reserved for restarters, and means the
907               template applies to all services which use the restarter.
908               'this' would refer to the defining service or instance.
909               'instance' can only be used in a service's template block,
910               and means the definition applies to all instances of this
911               service.
912

914 -->

916 <!ELEMENT pg_pattern

```

```

917     ( common_name?, description?, prop_pattern* ) >
919 <!ATTLIST pg_pattern
920     name          CDATA  ""
921     type          CDATA  ""
922     required      ( true | false )      "false"
923     target        ( this | instance | delegate | all )  "this" >
925 <!--
926     template
928     The template contains a collection of metadata about the service.
929     It contains a localizable string that serves as a common,
930     human-readable name for the service. (This name should be less than
931     60 characters in a single byte locale.) The template may optionally
932     contain a longer localizable description of the service, a
933     collection of links to documentation, either in the form of manual
934     pages or in the form of URI specifications to external documentation
935     sources (such as docs.sun.com).
937     The template has no attributes.
938 -->
939 <!ELEMENT template
940     ( common_name, description?, documentation?, pg_pattern* ) >
942 <!ATTLIST template>
944 <!-- Notification Parameters -->
946 <!ELEMENT paramval EMPTY>
948 <!ATTLIST paramval
949     name          CDATA #REQUIRED
950     value         CDATA #REQUIRED>
952 <!ELEMENT parameter
953     ( value_node* )>
955 <!ATTLIST parameter
956     name          CDATA #REQUIRED>
958 <!ELEMENT event EMPTY>
960 <!ATTLIST event
961     value         CDATA #REQUIRED>
963 <!ELEMENT type
964     ( ( parameter | paramval )* )>
966 <!ATTLIST type
967     name          CDATA #REQUIRED
968     active        ( true | false ) "true" >
970 <!--
971     notification parameters
973     This element sets the notification parameters for Software Events and
974     Fault Management problem lifecycle events.
975 -->
977 <!ELEMENT notification_parameters
978     ( event, type+ )>
980 <!ATTLIST notification_parameters>
982 <!-- Services and instances -->

```

```

984 <!--
985     create_default_instance
987     A flag element indicating that an otherwise empty default instance
988     of this service (named "default") should be created at install, with
989     its enabled property set as given.
991     Its attribute is
993         enabled [boolean] The initial value for the enabled state of
994         this instance.
995 -->
997 <!ELEMENT create_default_instance EMPTY >
999 <!ATTLIST create_default_instance
1000     enabled        ( true | false ) #REQUIRED >
1002 <!--
1003     single_instance
1005     A flag element stating that this service can only have a single
1006     instance on a particular system.
1007 -->
1009 <!ELEMENT single_instance EMPTY>
1011 <!ATTLIST single_instance>
1013 <!--
1014     instance
1016     The service instance is the object representing a software component
1017     that will run on the system if enabled. It contains an enabled
1018     element, a set of dependencies on other services, potentially
1019     customized methods or configuration data, an optional method
1020     context, and a pointer to its restarter. (If no restarter is
1021     specified, the master restarter, svc.startd(1M), is assumed to be
1022     responsible for the service.)
1024     Its attributes are
1026         name          The canonical name for this instance of the service.
1028         enabled [boolean] The initial value for the enabled state of
1029         this instance.
1030 -->
1032 <!ELEMENT instance
1033     ( restarter?, dependency*, dependent*, method_context?,
1034     exec_method*, notification_parameters*, property_group*,
1035     template? ) >
1037 <!ATTLIST instance
1038     name          CDATA #REQUIRED
1039     enabled        ( true | false ) #REQUIRED >
1041 <!--
1042     service
1044     The service contains the set of instances defined by default for
1045     this service, an optional method execution context, any default
1046     methods, the template, and various restrictions or advice applicable
1047     at installation. The method execution context and template elements
1048     are required for service_bundle documents with type "manifest", but

```

```
1049     are optional for "profile" or "archive" documents.
1051     Its attributes are
1053         name      The canonical name for the service.
1055         version [integer] The integer version for this service.
1057         type      Whether this service is a simple service, a delegated
1058                   restarter, or a milestone (a synthetic service that
1059                   collects a group of dependencies).
1060 -->
1062 <!ELEMENT service
1063     ( create_default_instance?, single_instance?, restarter?,
1064       dependency*, dependent*, method_context?, exec_method*,
1065       notification_parameters*, property_group*, instance*,
1066       stability?, template? ) >
1068 <!ATTLIST service
1069     name      CDATA #REQUIRED
1070     version   CDATA #REQUIRED
1071     type      ( service | restarter | milestone ) #REQUIRED >
1073 <!--
1074     service_bundle
1076     The bundle possesses two attributes:
1078         type      How this file is to be understood by the framework (or
1079                   used in a non-framework compliant way). Standard types
1080                   are 'archive', 'manifest', and 'profile'.
1081
1082         name      A name for the bundle. Manifests should be named after
1083                   the package which delivered them; profiles should be
1084                   named after the "feature set nickname" they intend to
1085                   enable.
1086 -->
1088 <!ELEMENT service_bundle
1089     ( service_bundle* | service* | xi:include* )>
1091 <!ATTLIST service_bundle
1092     type      CDATA #REQUIRED
1093     name      CDATA #REQUIRED>
```


new/usr/src/cmd/svc/milestone/Makefile

1

```
*****
3349 Thu Jun 30 22:01:37 2016
new/usr/src/cmd/svc/milestone/Makefile
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
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 (c) 2004, 2010, Oracle and/or its affiliates. All rights reserved.
23 #

25 include ../../Makefile.cmd

27 FILEMODE = 0444

29 BUILTXML= \
30     console-login.xml

32 FSSVCS= \
33     local-fs.xml \
34     minimal-fs.xml \
35     root-fs.xml \
36     usr-fs.xml

38 FSMANIFESTS= $(FSSVCS:%=$(ROOTSVCSYSTEMFILESYSYSTEM)/%)

40 NETSVCS= \
41     network-initial.xml \
42     network-install.xml \
43     network-iptun.xml \
44     network-ipqos.xml \
45     network-location.xml \
46     network-loopback.xml \
47     network-netcfg.xml \
48     network-netmask.xml \
49     network-netcfg.xml \
50     network-physical.xml \
51     network-routing-setup.xml \
52     network-service.xml

54 NETMANIFESTS= $(NETSVCS:%=$(ROOTSVCNETWORK)/%)

56 MAINMILESTONES= \
57     multi-user-server.xml \
58     multi-user.xml \
```

new/usr/src/cmd/svc/milestone/Makefile

2

```
59     name-services.xml \
60     network.xml \
61     single-user.xml \
62     sysconfig.xml

64 MAINMANIFESTS= $(MAINMILESTONES:%=$(ROOTSVCMILESTONE)/%)

66 SYSDESVCS= \
67     devices-local.xml \
68     devices-audio.xml

70 SYSDEVMANIFESTS= $(SYSDESVCS:%=$(ROOTSVCSYSTEMDEVICE)/%)

72 SYSTEMSVCS= \
73     boot-archive.xml \
74     console-login.xml \
75     early-manifest-import.xml \
76     identity.xml \
77     manifest-import.xml \
78     process-security.xml \
79 #endif /* ! codereview */
80     rmtmpfiles.xml \
81     vtdaemon.xml

83 SYSTEMMANIFESTS = $(SYSTEMSVCS:%=$(ROOTSVCSYSTEM)/%)

85 SYSTEMSVCSVCS= \
86     restarter.xml \
87     global.xml

89 SYSTEMSVCMANIFESTS= $(SYSTEMSVCSVCS:%=$(ROOTSVCSYSTEM)/svc/%)

91 MISCFILES= \
92     README.share

94 SYSTEMMISCFILES = $(MISCFILES:%.share=$(ROOT)/lib/svc/share/%)

96 #
97 # MANIFEST is used solely in the construction of the check target.
98 #
99 MANIFEST= $(FSSVCS) $(NETSVCS) $(MAINMILESTONES) $(SYSTEMSVCS) \
100     $(SYSDESVCS) $(SYSTEMSVCSVCS)

102 SVCMETHOD=\
103     boot-archive \
104     console-login \
105     devices-audio \
106     devices-local \
107     fs-local \
108     fs-minimal \
109     fs-root \
110     fs-usr \
111     identity-domain \
112     identity-node \
113     manifest-import \
114     net-loc \
115     net-loopback \
116     net-init \
117     net-install \
118     net-iptun \
119     net-ipqos \
120     net-netmask \
121     net-nwam \
122     net-physical \
123     net-routing-setup \
124     net-svc \
```

```
125     rmtmpfiles \
126     vtdaemon

128 $(ROOTSVCMETHOD) := FILEMODE = 0555

130 all: $(BUILTXML)

132 install: $(FSMANIFESTS) $(MAINMANIFESTS) $(NETMANIFESTS) $(SYSTEMMANIFESTS) \
133           $(ROOTSVCMETHOD) $(SYSDEVMANIFESTS) $(SYSTEMSVCMANIFESTS) \
134           $(SYSTEMMISCFILES)

136 check: $(CHKMANIFEST)

138 console-login.xml: make-console-login-xml
139     $(SH) ./make-console-login-xml

141 clobber: clean
142     -$(RM) $(BUILTXML)

144 $(ROOTSVC MILESTONE)/%: %
145     $(INS.file)

147 $(ROOTSVC NETWORK)/%: %
148     $(INS.file)

150 $(ROOTSVC SYSTEM)/%: %
151     $(INS.file)

153 $(ROOTSVC SYSTEM DEVICE)/%: %
154     $(INS.file)

156 $(ROOTSVC SYSTEM FILESYSTEM)/%: %
157     $(INS.file)

159 $(ROOTSVC SYSTEM)/svc/%: %
160     $(INS.file)

162 $(ROOT)/lib/svc/share/%: %.share
163     $(INS.rename)

165 clean lint _msg:
```

new/usr/src/cmd/svc/milestone/global.xml

1

```
*****
28885 Thu Jun 30 22:01:38 2016
new/usr/src/cmd/svc/milestone/global.xml
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
1 <?xml version="1.0"?>
2 <!DOCTYPE service_bundle SYSTEM "/usr/share/lib/xml/dtd/service_bundle.dtd.1">
3 <!--
4 Copyright (c) 2008, 2010, Oracle and/or its affiliates. All rights reserved.
5 Copyright 2016 Hans Rosenfeld <rosenfeld@grumpf.hope-2000.org>

7 CDDL HEADER START

9 The contents of this file are subject to the terms of the
10 Common Development and Distribution License (the "License").
11 You may not use this file except in compliance with the License.

13 You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
14 or http://www.opensolaris.org/os/licensing.
15 See the License for the specific language governing permissions
16 and limitations under the License.

18 When distributing Covered Code, include this CDDL HEADER in each
19 file and include the License file at usr/src/OPENSOLARIS.LICENSE.
20 If applicable, add the following below this CDDL HEADER, with the
21 fields enclosed by brackets "[]" replaced with your own identifying
22 information: Portions Copyright [yyyy] [name of copyright owner]

24 CDDL HEADER END

26 NOTE: This service manifest is not editable; its contents will
27 be overwritten by package or patch operations, including
28 operating system upgrade. Make customizations in a different
29 file.
30 -->
32 <service_bundle type='manifest' name='SUNWcsr:global'>
34 <service
35 name='system/svc/global'
36 type='service'
37 version='1'>
39 <!--
40 There's no running configuration to manage here. However,
41 this service stores the system-wide definitions for
42 templates.
43 -->
45 <create_default_instance enabled='false' />
47 <single_instance/>
49 <exec_method
50 type='method'
51 name='start'
52 exec=':true'
53 timeout_seconds='0' />
55 <exec_method
56 type='method'
57 name='stop'
58 exec=':true'
```

new/usr/src/cmd/svc/milestone/global.xml

2

```
59 timeout_seconds='0' />
61 <stability value='Unstable' />
63 <template>
64 <common_name>
65 <loctext xml:lang='C'>
66 system-wide configuration definitions
67 </loctext>
68 </common_name>
69 <documentation>
70 <manpage title='smf' section='5'
71 manpath='/usr/share/man' />
72 <manpage title='smf_template' section='5'
73 manpath='/usr/share/man' />
74 </documentation>
76 <pg_pattern name='general' type='framework'
77 target='all' required='true'>
78 <description>
79 <loctext xml:lang='C'>
80 Basic information about a service instance which is supplied by the service auth
81 </loctext>
82 </description>
83 <prop_pattern name='enabled' type='boolean'
84 required='true'>
85 <description>
86 <loctext xml:lang='C'>
87 The service instance is expected to be started once all of its dependencies are
88 </loctext>
89 </description>
90 <cardinality min='1' max='1' />
91 </prop_pattern>
92 <prop_pattern name='restarter' type='fmri'
93 required='false'>
94 <description>
95 <loctext xml:lang='C'>
96 The restarter responsible for managing this service instance. If the property i
97 </loctext>
98 </description>
99 <cardinality min='1' max='1' />
100 </prop_pattern>
101 <prop_pattern name='single_instance' type='boolean'
102 required='false'>
103 <description>
104 <loctext xml:lang='C'>
105 Only one instance of this service may be run. This property is currently unenfo
106 </loctext>
107 </description>
108 <cardinality min='1' max='1' />
109 </prop_pattern>
110 </pg_pattern>
112 <pg_pattern type='dependency' target='all' required='false'>
113 <description>
114 <loctext xml:lang='C'>
115 A dependency declares a required condition for a service instance to start or st
116 </loctext>
117 </description>
118 <prop_pattern name='entities' type='fmri'
119 required='true'>
120 <description>
121 <loctext xml:lang='C'>
122 The services, service instances, or files used to calculate this dependency.
123 </loctext>
124 </description>
```

```

125         <cardinality min='1' />
126     </prop_pattern>
127     <prop_pattern name='external' type='boolean'>
128         <description>
129             <loctext xml:lang='C'>
130 This dependency was declared by the service defined in entities. It will be rem
131             </loctext>
132         </description>
133         <visibility value='readonly' />
134         <cardinality min='1' max='1' />
135     </prop_pattern>
136     <prop_pattern name='grouping' type='astring'
137         required='true'>
138         <description>
139             <loctext xml:lang='C'>
140 How to decide whether this dependency is satisfied.
141             </loctext>
142         </description>
143         <cardinality min='1' max='1' />
144         <constraints>
145             <value name='require_all'>
146                 <description>
147                     <loctext xml:lang='C'>
148 Satisfied when all cited services are running (online or degraded), or when all
149                     </loctext>
150                 </description>
151             </value>
152             <value name='require_any'>
153                 <description>
154                     <loctext xml:lang='C'>
155 Satisfied when one of the cited services is running (online or degraded), or whe
156                     </loctext>
157                 </description>
158             </value>
159             <value name='optional_all'>
160                 <description>
161                     <loctext xml:lang='C'>
162 Satisfied if the cited services are running (online or degraded) or will not run
163                     </loctext>
164                 </description>
165             </value>
166             <value name='exclude_all'>
167                 <description>
168                     <loctext xml:lang='C'>
169 Satisfied when all of the cited services are disabled, in the maintenance state,
170                     </loctext>
171                 </description>
172             </value>
173         </constraints>
174         <choices>
175             <include_values type='constraints' />
176         </choices>
177     </prop_pattern>
178     <prop_pattern name='restart_on' type='astring'
179         required='true'>
180         <description>
181             <loctext xml:lang='C'>
182 Determines whether to restart the service due to a dependency refresh, restart,
183             </loctext>
184         </description>
185         <cardinality min='1' max='1' />
186         <constraints>
187             <value name='none'>
188                 <description>
189                     <loctext xml:lang='C'>
190

```

```

191 Never restart due to dependency refresh, restart, or failure.
192             </loctext>
193         </description>
194     </value>
195     <value name='error'>
196         <description>
197             <loctext xml:lang='C'>
198 Restart only if the dependency encounters an error, such as an uncorrectable har
199             </loctext>
200         </description>
201     </value>
202     <value name='restart'>
203         <description>
204             <loctext xml:lang='C'>
205 Restart if the dependency encounters an error or is explicitly restarted.
206             </loctext>
207         </description>
208     </value>
209     <value name='refresh'>
210         <description>
211             <loctext xml:lang='C'>
212 Restart if the dependency encounters an error, is explicitly restarted, or expli
213             </loctext>
214         </description>
215     </value>
216 </constraints>
217 <choices>
218     <include_values type='constraints' />
219 </choices>
220 </prop_pattern>
221 <prop_pattern name='type' type='astring'
222     required='true'>
223     <description>
224         <loctext xml:lang='C'>
225 The type of the dependency: service or file.
226         </loctext>
227     </description>
228     <cardinality min='1' max='1' />
229     <constraints>
230         <value name='service'>
231             <description>
232                 <loctext xml:lang='C'>
233 Depend on services or instances.
234                 </loctext>
235             </description>
236         </value>
237         <value name='path'>
238             <description>
239                 <loctext xml:lang='C'>
240 Depend on the existence of a file path.
241                 </loctext>
242             </description>
243         </value>
244     </constraints>
245     <choices>
246         <include_values type='constraints' />
247     </choices>
248 </prop_pattern>
249 </pg_pattern>
250
251 <pg_pattern type='template_pg_pattern' target='all'
252     required='false'>
253     <description>
254         <loctext xml:lang='C'>
255 Template data about property groups. This information is provided in the manife
256         </loctext>

```

```

257         </description>
259         <prop_pattern name='name' type='astring'
260           required='false'>
261           <description>
262             <loctext xml:lang='C'>
263 Optional name of a property group which is described by this template. No name
264             </loctext>
265           </description>
266           <visibility value='hidden'/>
267           <cardinality min='1' max='1'/>
268         </prop_pattern>
269         <prop_pattern name='type' type='astring'
270           required='false'>
271           <description>
272             <loctext xml:lang='C'>
273 Optional type of property groups which are described by this template. No type
274             </loctext>
275           </description>
276           <visibility value='hidden'/>
277           <cardinality min='1' max='1'/>
278         </prop_pattern>
279         <prop_pattern name='required' type='boolean'
280           required='false'>
281           <description>
282             <loctext xml:lang='C'>
283 If true, entities without a property group which matches this pattern are consid
284             </loctext>
285           </description>
286           <visibility value='hidden'/>
287           <cardinality min='1' max='1'/>
288         </prop_pattern>
289         <prop_pattern name='target' type='astring'
290           required='false'>
291           <description>
292             <loctext xml:lang='C'>
293 The services or service instances to which this template should be applied.
294             </loctext>
295           </description>
296           <visibility value='hidden'/>
297           <cardinality min='1' max='1'/>
298           <constraints>
299             <value name='this'>
300               <description>
301                 <loctext xml:lang='C'>
302 The service or instance on which the property group resides.
303                 </loctext>
304               </description>
305             </value>
306             <value name='instance'>
307               <description>
308                 <loctext xml:lang='C'>
309 This instance, or any instance of this service.
310                 </loctext>
311               </description>
312             </value>
313             <value name='delegate'>
314               <description>
315                 <loctext xml:lang='C'>
316 All instances which currently define this service as their restarter.
317                 </loctext>
318               </description>
319             </value>
320             <value name='all'>
321               <description>
322                 <loctext xml:lang='C'>

```

```

323 All services and instances on the system. "all" may only be set on the global s
324         </loctext>
325       </description>
326     </value>
327   </constraints>
328 </prop_pattern>
329 </pg_pattern>
331 <pg_pattern type='template_prop_pattern' target='all'
332   required='false'>
333   <description>
334     <loctext xml:lang='C'>
335 Template data about properties. This information is provided in the manifest by
336     </loctext>
337   </description>
338   <prop_pattern name='name' type='astring'
339     required='true'>
340     <description>
341       <loctext xml:lang='C'>
342 Name of property this template applies to.
343       </loctext>
344     </description>
345     <visibility value='hidden'/>
346     <cardinality min='1' max='1'/>
347   </prop_pattern>
348   <prop_pattern name='pg_pattern' type='astring'
349     required='true'>
350     <description>
351       <loctext xml:lang='C'>
352 Name of property group that describes the enclosing property group pattern.
353       </loctext>
354     </description>
355     <visibility value='hidden'/>
356     <cardinality min='1' max='1'/>
357   </prop_pattern>
358   <prop_pattern name='required' type='boolean'
359     required='false'>
360     <description>
361       <loctext xml:lang='C'>
362 Defines whether a property matched by this template is required.
363       </loctext>
364     </description>
365     <visibility value='hidden'/>
366     <cardinality min='1' max='1'/>
367   </prop_pattern>
368   <prop_pattern name='type' type='astring'
369     required='false'>
370     <description>
371       <loctext xml:lang='C'>
372 The type that a property which this template refers to should be.
373       </loctext>
374     </description>
375     <visibility value='hidden'/>
376     <cardinality min='1' max='1'/>
377   </prop_pattern>
378   <prop_pattern name='visibility' type='astring'
379     required='false'>
380     <description>
381       <loctext xml:lang='C'>
382 The visibility of this property, which is readwrite by default. Visibility is o
383       </loctext>
384     </description>
385     <visibility value='hidden'/>
386     <cardinality min='1' max='1'/>
387   <constraints>
388     <value name='hidden'>

```

```

389         <description>
390             <loctext xml:lang='C'>
391 Hidden in default user interface views.
392             </loctext>
393         </description>
394     </value>
395     <value name='readonly'>
396         <description>
397             <loctext xml:lang='C'>
398 Expected to be read only in most user interfaces.
399             </loctext>
400         </description>
401     </value>
402     <value name='readwrite'>
403         <description>
404             <loctext xml:lang='C'>
405 Expected to be manipulated in many user interfaces.
406             </loctext>
407         </description>
408     </value>
409 </constraints>
410 </prop_pattern>
411 <prop_pattern name='cardinality_min' type='count'
412     required='false'>
413     <description>
414         <loctext xml:lang='C'>
415 Minimum number of required values.
416         </loctext>
417     </description>
418     <cardinality min='1' max='1' />
419 </prop_pattern>
420 <prop_pattern name='cardinality_max' type='count'
421     required='false'>
422     <description>
423         <loctext xml:lang='C'>
424 Maximum number of required values.
425         </loctext>
426     </description>
427     <visibility value='hidden' />
428     <cardinality min='1' max='1' />
429 </prop_pattern>
430 <prop_pattern name='internal_separators' type='astring'
431     required='false'>
432     <description>
433         <loctext xml:lang='C'>
434 List of separator characters for values.
435         </loctext>
436     </description>
437     <visibility value='hidden' />
438     <cardinality min='1' />
439 </prop_pattern>
440 <prop_pattern name='constraint_name' type='astring'
441     required='false'>
442     <description>
443         <loctext xml:lang='C'>
444 Values the property is expected to be constrained to.
445         </loctext>
446     </description>
447     <visibility value='hidden' />
448     <cardinality min='1' />
449 </prop_pattern>
450 <prop_pattern name='constraint_range' type='astring'
451     required='false'>
452     <description>
453         <loctext xml:lang='C'>
454 Ranges the property is expected to be constrained to.

```

```

455         </loctext>
456     </description>
457     <visibility value='hidden' />
458     <cardinality min='1' />
459     <internal_separators>,</internal_separators>
460 </prop_pattern>
461 <prop_pattern name='choices_range' type='astring'
462     required='false'>
463     <description>
464         <loctext xml:lang='C'>
465 Ranges a user should be offered as a choice for this property.
466         </loctext>
467     </description>
468     <visibility value='hidden' />
469     <cardinality min='1' />
470     <internal_separators>,</internal_separators>
471 </prop_pattern>
472 <prop_pattern name='choices_name' type='astring'
473     required='false'>
474     <description>
475         <loctext xml:lang='C'>
476 Values a users should be offered as a choice for this property.
477         </loctext>
478     </description>
479     <visibility value='hidden' />
480     <cardinality min='1' />
481 </prop_pattern>
482 <prop_pattern name='choices_include_values'
483     type='astring' required='false'>
484     <description>
485         <loctext xml:lang='C'>
486 Whether the choices should include the defined constraints or values.
487         </loctext>
488     </description>
489     <visibility value='hidden' />
490     <cardinality min='1' max='1' />
491     <constraints>
492     <value name='constraints'>
493         <description>
494             <loctext xml:lang='C'>
495 Include all defined constraints as choices.
496             </loctext>
497         </description>
498     </value>
499     <value name='values'>
500         <description>
501             <loctext xml:lang='C'>
502 Include all defined values as choices.
503             </loctext>
504         </description>
505     </value>
506 </constraints>
507 </prop_pattern>
508 </pg_pattern>
509
510 <pg_pattern name='method_context' type='framework'
511     target='all' required='false'>
512     <description>
513         <loctext xml:lang='C'>
514 Specifies the default execution context for all service methods. It is defined
515         </loctext>
516     </description>
517
518     <!-- method_context direct properties -->
519     <prop_pattern name='working_directory' type='astring'
520     required='false'>

```

```

521         <description>
522             <loctext xml:lang='C'>
523 The working directory to launch the method from. ":default" can be used as a to
524             </loctext>
525         </description>
526         <cardinality min='1' max='1' />
527     </prop_pattern>
528     <prop_pattern name='project' type='astring'
529         required='false'>
530         <description>
531             <loctext xml:lang='C'>
532 The project ID in numeric or text form. ":default" can be used as a token to in
533             </loctext>
534         </description>
535         <cardinality min='1' max='1' />
536     </prop_pattern>
537     <prop_pattern name='resource_pool' type='astring'
538         required='false'>
539         <description>
540             <loctext xml:lang='C'>
541 The resource pool name in which to launch the method. ":default" can be used
542 as a token to indicate the pool specified in the project(4) entry given in
543 the project attribute.
544             </loctext>
545         </description>
546         <cardinality min='1' max='1' />
547     </prop_pattern>
548     <prop_pattern name='security_flags' type='astring'
549         required='false'>
550         <description>
551             <loctext xml:lang='C'>
552 An optional string specifying the security flags as defined in security-flags(5)
553             </loctext>
554         </description>
555         <cardinality min='1' max='1' />
556     </prop_pattern>
557 #endif /* ! codereview */

559     <!-- method_credential properties -->
560     <prop_pattern name='user' type='astring'
561         required='false'>
562         <description>
563             <loctext xml:lang='C'>
564 The user ID in numeric or text form.
565             </loctext>
566         </description>
567         <cardinality min='1' max='1' />
568     </prop_pattern>
569     <prop_pattern name='group' type='astring'
570         required='false'>
571         <description>
572             <loctext xml:lang='C'>
573 The group ID in numeric or text form.
574             </loctext>
575         </description>
576         <cardinality min='1' max='1' />
577     </prop_pattern>
578     <prop_pattern name='supp_groups' type='astring'
579         required='false'>
580         <description>
581             <loctext xml:lang='C'>
582 An optional string that specifies the supplemental group memberships by ID,
583 in numeric or text form.
584             </loctext>
585         </description>
586         <cardinality min='1' max='1' />

```

```

587     </prop_pattern>
588     <prop_pattern name='privileges' type='astring'
589         required='false'>
590         <description>
591             <loctext xml:lang='C'>
592 An optional string specifying the privilege set as defined in privileges(5).
593             </loctext>
594         </description>
595         <cardinality min='1' max='1' />
596     </prop_pattern>
597     <prop_pattern name='limit_privileges' type='astring'
598         required='false'>
599         <description>
600             <loctext xml:lang='C'>
601 An optional string specifying the limit privilege set as defined in
602 privileges(5).
603             </loctext>
604         </description>
605         <cardinality min='1' max='1' />
606     </prop_pattern>

608     <!-- method_profile properties -->
609     <prop_pattern name='use_profile' type='boolean'
610         required='false'>
611         <description>
612             <loctext xml:lang='C'>
613 A boolean that specifies whether the profile should be used instead of the
614 user, group, privileges, and limit_privileges properties.
615             </loctext>
616         </description>
617         <cardinality min='1' max='1' />
618     </prop_pattern>
619     <prop_pattern name='profile' type='astring'
620         required='false'>
621         <description>
622             <loctext xml:lang='C'>
623 The name of an RBAC (role-based access control) profile which, along with the
624 method executable, identifies an entry in exec_attr(4).
625             </loctext>
626         </description>
627         <cardinality min='1' max='1' />
628     </prop_pattern>
629 </pg_pattern>

631 <pg_pattern name='firewall_context'
632     type='com.sun.fw_definition' target='all' required='false'>
633     <common_name>
634         <loctext xml:lang='C'>
635 Static definition
636         </loctext>
637     </common_name>
638     <description>
639         <loctext xml:lang='C'>
640 Service static network and firewall definition.
641         </loctext>
642     </description>
643     <prop_pattern name='name' type='astring'
644         required='false'>
645         <common_name>
646             <loctext xml:lang='C'>
647 Service name
648             </loctext>
649         </common_name>
650     </description>
651     <loctext xml:lang='C'>
652 IANA name or RPC name for non-inetd service, equivalent to inetd/name property.

```

```

653         </loctext>
654     </description>
655 </prop_pattern>
656 <prop_pattern name='isrpc' type='boolean'
657     required='false'
658     <common_name
659         <loctext xml:lang='C'>
660 RPC service
661     </loctext>
662     </common_name>
663     <description>
664         <loctext xml:lang='C'>
665 A boolean property where a "true" value indicates an RPC service, equivalent to
666     </loctext>
667     </description>
668 </prop_pattern>
669 <prop_pattern name='ipf_method' type='astring'
670     required='false'
671     <common_name
672         <loctext xml:lang='C'>
673 Custom firewall script
674     </loctext>
675     </common_name>
676     <description>
677         <loctext xml:lang='C'>
678 A script that generates ipf rules for a service. Services that require custom IP
679     </loctext>
680     </description>
681 </prop_pattern> </pg_pattern>

683 <pg_pattern name='firewall_config'
684     type='com.sun.fw_configuration' target='all'
685     required='false'
686     <common_name
687         <loctext xml:lang='C'>
688 Firewall configuration
689     </loctext>
690     </common_name>
691     <description>
692         <loctext xml:lang='C'>
693 Service firewall configuration.
694     </loctext>
695     </description>
696 <prop_pattern name='policy' type='astring'
697     required='true'
698     <common_name
699         <loctext xml:lang='C'>
700 Firewall policy
701     </loctext>
702     </common_name>
703     <description>
704         <loctext xml:lang='C'>
705 Service firewall policy.
706     </loctext>
707     </description>
708     <visibility value='readwrite' />
709     <cardinality min='1' max='1' />
710     <values>
711         <value name='use_global'
712             <description>
713                 <loctext xml:lang='C'>
714 Apply Global Default policy, specified in network/ipfilter for the service. This
715             </loctext>
716         </description>
717     </value>
718     <value name='none'

```

```

719     <description>
720         <loctext xml:lang='C'>
721 No firewall (allow all).
722     </loctext>
723     </description>
724     </value>
725     <value name='deny'
726         <description>
727             <loctext xml:lang='C'>
728 Deny access to entities specified in 'apply_to' property.
729         </loctext>
730         </description>
731     </value>
732     <value name='allow'
733         <description>
734             <loctext xml:lang='C'>
735 Allow access to entities specified in 'apply_to' property.
736         </loctext>
737         </description>
738     </value>
739 </values>
740 <choices>
741     <include_values type='values' />
742 </choices>
743 </prop_pattern>
744 <prop_pattern name='block_policy' type='astring'
745     required='false'
746     <common_name
747         <loctext xml:lang='C'>
748 Firewall block policy
749     </loctext>
750     </common_name>
751     <description>
752         <loctext xml:lang='C'>
753 Service firewall block policy.
754     </loctext>
755     </description>
756     <visibility value='readwrite' />
757     <cardinality min='1' max='1' />
758     <values>
759         <value name='use_global'
760             <description>
761                 <loctext xml:lang='C'>
762 Apply Global Default block policy, specified in network/ipfilter for the service
763             </loctext>
764         </description>
765     </value>
766     <value name='none'
767         <description>
768             <loctext xml:lang='C'>
769 Block by dropping packets.
770         </loctext>
771         </description>
772     </value>
773     <value name='return'
774         <description>
775             <loctext xml:lang='C'>
776 Block by returning RST or ICMP messages.
777         </loctext>
778         </description>
779     </value>
780 </values>
781 <choices>
782     <include_values type='values' />
783 </choices>
784 </prop_pattern>

```



```

785     <prop_pattern name="apply_to" type="astring"
786         required="false">
787         <common_name>
788             <loctext xml:lang='C'>
789 Apply policy to
790             </loctext>
791         </common_name>
792         <description>
793             <loctext xml:lang="C">
794 The source host and network IPv4 addresses, incoming network interfaces, and ipp
795             </loctext>
796         </description>
797     </prop_pattern>
798     <prop_pattern name="apply_to_6" type="astring"
799         required="false">
800         <common_name>
801             <loctext xml:lang='C'>
802 Apply policy to
803             </loctext>
804         </common_name>
805         <description>
806             <loctext xml:lang="C">
807 The source host and network IPv6 addresses, incoming network interfaces, and ipp
808             </loctext>
809         </description>
810     </prop_pattern>
811     <prop_pattern name="exceptions" type="astring"
812         required="false">
813         <common_name>
814             <loctext xml:lang='C'>
815 Make exceptions to
816             </loctext>
817         </common_name>
818         <description>
819             <loctext xml:lang="C">
820 The source host and network IPv4 addresses, incoming network interfaces, and ipp
821             </loctext>
822         </description>
823     </prop_pattern>
824     <prop_pattern name="exceptions_6" type="astring"
825         required="false">
826         <common_name>
827             <loctext xml:lang='C'>
828 Make exceptions to
829             </loctext>
830         </common_name>
831         <description>
832             <loctext xml:lang="C">
833 The source host and network IPv6 addresses, incoming network interfaces, and ip
834             </loctext>
835         </description>
836     </prop_pattern>
837     <prop_pattern name="target" type="astring"
838         required="false">
839         <common_name>
840             <loctext xml:lang='C'>
841 Apply policy to
842             </loctext>
843         </common_name>
844         <description>
845             <loctext xml:lang="C">
846 The destination host and network IPv4 addresses, and ippools to deny if the poli
847             </loctext>
848         </description>
849     </prop_pattern>
850     <prop_pattern name="target6" type="astring"

```

```

851         required="false">
852         <common_name>
853             <loctext xml:lang='C'>
854 Apply policy to
855             </loctext>
856         </common_name>
857         <description>
858             <loctext xml:lang="C">
859 The destination host and network IPv6 addresses, and ippools to deny if the poli
860             </loctext>
861         </description>
862     </prop_pattern>
863 </pg_pattern>
864     <pg_pattern type='notify_params' target='all' required='false'>
865         <common_name>
866             <loctext xml:lang='C'>
867 FMA and SMF notification parameters
868             </loctext>
869         </common_name>
870         <description>
871             <loctext xml:lang='C'>
872 Parameters for notification of FMA events and SMF state transitions.
873             </loctext>
874         </description>
875         <prop_pattern name='smtp,active' type='boolean'
876             required='false'>
877             <common_name>
878                 <loctext xml:lang='C'>
879 smtp notification active
880                 </loctext>
881             </common_name>
882             <description>
883                 <loctext xml:lang='C'>
884 Notification mechanism smtp active status.
885                 </loctext>
886             </description>
887         </prop_pattern>
888         <prop_pattern name='smtp,to' type='astring'
889             required='false'>
890             <common_name>
891                 <loctext xml:lang='C'>
892 smtp notification recipient
893                 </loctext>
894             </common_name>
895             <description>
896                 <loctext xml:lang='C'>
897 Recipient for smtp notification mechanism.
898                 </loctext>
899             </description>
900         </prop_pattern>
901         <prop_pattern name='smtp,reply-to' type='astring'
902             required='false'>
903             <common_name>
904                 <loctext xml:lang='C'>
905 smtp notification reply-to
906                 </loctext>
907             </common_name>
908             <description>
909                 <loctext xml:lang='C'>
910 Header reply-to for smtp notification mechanism.
911                 </loctext>
912             </description>
913         </prop_pattern>
914         <prop_pattern name='snmp,active' type='boolean'
915             required='false'>
916             <common_name>

```

```
917         <loctext xml:lang='C'>
918 snmp notification active
919         </loctext>
920         </common_name>
921         <description>
922         <loctext xml:lang='C'>
923 Notification mechanism snmp active status.
924         </loctext>
925         </description>
926     </prop_pattern>
927     <prop_pattern name='syslog,active' type='boolean'
928         required='false'>
929         <common_name>
930         <loctext xml:lang='C'>
931 syslog notification active
932         </loctext>
933         </common_name>
934         <description>
935         <loctext xml:lang='C'>
936 Notification mechanism syslog active status.
937         </loctext>
938         </description>
939     </prop_pattern>
940 </pg_pattern>
941 </template>
942 </service>
944 </service_bundle>
```

```

*****
2941 Thu Jun 30 22:01:39 2016
new/usr/src/cmd/svc/milestone/process-security.xml
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
1 <?xml version='1.0'?>
2 <!DOCTYPE service_bundle SYSTEM '/usr/share/lib/xml/dtd/service_bundle.dtd.1'>

4 <!--
5 Copyright 2015, Richard Lowe.

7 CDDL HEADER START

9 This file and its contents are supplied under the terms of the
10 Common Development and Distribution License ("CDDL"), version 1.0.
11 You may only use this file in accordance with the terms of version
12 1.0 of the CDDL.

14 A full copy of the text of the CDDL should have accompanied this
15 source. A copy of the CDDL is also available via the Internet at
16 http://www.illumos.org/license/CDDL.

18 CDDL HEADER END

20 NOTE: This service manifest is not editable; its contents will
21 be overwritten by package or patch operations, including
22 operating system upgrade. Make customizations in a different
23 file.
24 -->

26 <service_bundle type="manifest" name="process-security">
27   <service name="system/process-security" type="service" version="1">
28     <!-- Initial state of the service is disabled -->
29     <create_default_instance enabled="false" />

31     <single_instance />

33     <!-- We don't actually have any methods, but we create a
34     default instance so that we show up in svcs -a -->

36     <exec_method type="method" name="start" exec=":true" timeout_sec
37     <exec_method type="method" name="stop" exec=":true" timeout_sec

39     <property_group name='startd' type='framework'>
40       <propval name='duration' type='astring' value='transient' />
41     </property_group>

43     <property_group name='default' type='application'>
44       <propval name='aslr' type='boolean' value='false' />
45       <propval name='forbidnullmap' type='boolean' value='false' />
46       <propval name='noexecstack' type='boolean' value='false' />

48       <propval name='value_authorization' type='astring'
49       value='solaris.smf.value.process-security' />
50     </property_group>

52     <property_group name='lower' type='application'>
53       <propval name='aslr' type='boolean' value='false' />
54       <propval name='forbidnullmap' type='boolean' value='false' />
55       <propval name='noexecstack' type='boolean' value='false' />

57     <propval name='value_authorization' type='astring'
58     value='solaris.smf.value.process-security' />

```

```

59     </property_group>

61     <property_group name='upper' type='application'>
62       <propval name='aslr' type='boolean' value='true' />
63       <propval name='forbidnullmap' type='boolean' value='true' />
64       <propval name='noexecstack' type='boolean' value='true' />

66       <propval name='value_authorization' type='astring'
67       value='solaris.smf.value.process-security' />
68     </property_group>

72     <stability value="Unstable" />

74     <template>
75       <common_name>
76         <loctext xml:lang='C'>Security Flag Configuratio
77       </common_name>
78       <documentation>
79         <manpage title='security-flags' section='5'
80         manpath='/usr/share/man' />
81         <manpage title='psecflags' section='1'
82         manpath='/usr/share/man' />
83       </documentation>
84     </template>
85   </service>
86 </service_bundle>
87 #endif /* ! codereview */

```

```

*****
39483 Thu Jun 30 22:01:39 2016
new/usr/src/cmd/svc/milestone/restarter.xml
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
1 <?xml version="1.0"?>
2 <!DOCTYPE service_bundle SYSTEM "/usr/share/lib/xml/dtd/service_bundle.dtd.1">
3 <!--
4 Copyright (c) 2004, 2010, Oracle and/or its affiliates. All rights reserved.
5 Copyright 2015 Nexenta Systems, Inc. All rights reserved.

7 CDDL HEADER START

9 The contents of this file are subject to the terms of the
10 Common Development and Distribution License (the "License").
11 You may not use this file except in compliance with the License.

13 You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
14 or http://www.opensolaris.org/os/licensing.
15 See the License for the specific language governing permissions
16 and limitations under the License.

18 When distributing Covered Code, include this CDDL HEADER in each
19 file and include the License file at usr/src/OPENSOLARIS.LICENSE.
20 If applicable, add the following below this CDDL HEADER, with the
21 fields enclosed by brackets "[]" replaced with your own identifying
22 information: Portions Copyright [yyyy] [name of copyright owner]

24 CDDL HEADER END

26 NOTE: This service manifest is not editable; its contents will
27 be overwritten by package or patch operations, including
28 operating system upgrade. Make customizations in a different
29 file.
30 -->

32 <service_bundle type='manifest' name='SUNWcsr:restarter'>

34 <service
35 name='system/svc/restarter'
36 type='restarter'
37 version='1'>

39 <!--
40 svc.startd manages itself. However, this manifest allows
41 us to set non-persistent properties before filesystems
42 have been mounted r/w.
43 -->

45 <create_default_instance enabled='true' />

47 <single_instance/>

49 <!--
50 The restarter is actually started by init, so these methods are
51 ignored. However, they are required by definition and are
52 included here to avoid spurious validation errors.
53 -->
54 <exec_method
55 type='method'
56 name='start'
57 exec=':true'
58 timeout_seconds='0' />

```

```

60 <exec_method
61 type='method'
62 name='stop'
63 exec=':true'
64 timeout_seconds='0' />

66 <stability value='Unstable' />

68 <template>
69 <common_name>
70 <loctext xml:lang='C'>
71 master restarter
72 </loctext>
73 </common_name>
74 <documentation>
75 <manpage title='svc.startd' section='1M'
76 manpath='/usr/share/man' />
77 <manpage title='smf_method' section='5'
78 manpath='/usr/share/man' />
79 <manpage title='smf' section='5'
80 manpath='/usr/share/man' />
81 </documentation>

83 <!--
84 Much of the restarter pg is populated by librestart, but
85 because svc.startd augments the property group, we choose
86 to define the entire property group as restarter-specific
87 rather than define it globally and miss some of the
88 properties. Templates does not currently provide a way
89 to have multiple entities 'own' and describe a property
90 group.
91 -->
92 <pg_pattern name='restarter' type='framework'
93 target='delegate' required='false'>
94 <description>
95 <loctext xml:lang='C'>
96 Communicate restarter-set status of the service.
97 </loctext>
98 </description>
99 <prop_pattern name='alt_logfile' type='astring'
100 required='false'>
101 <description>
102 <loctext xml:lang='C'>
103 The logfile for restarter actions on this service and any direct output from its
104 </loctext>
105 </description>
106 <visibility value='readonly' />
107 <cardinality min='1' max='1' />
108 </prop_pattern>
109 <prop_pattern name='logfile' type='astring'
110 required='false'>
111 <description>
112 <loctext xml:lang='C'>
113 The logfile for restarter actions on this service and any direct output from its
114 </loctext>
115 </description>
116 <visibility value='readonly' />
117 <cardinality min='1' max='1' />
118 </prop_pattern>
119 <prop_pattern name='contract' type='count'
120 required='false'>
121 <description>
122 <loctext xml:lang='C'>
123 Primary process contract for a 'contract' or 'child' service.
124 </loctext>

```

```

125         </description>
126         <visibility value='readonly' />
127         <cardinality min='1' max='1' />
128     </prop_pattern>
129     <prop_pattern name='start_pid' type='count'
130         required='false'>
131         <description>
132             <loctext xml:lang='C'>
133 PID last launched as the start method for this service.
134             </loctext>
135         </description>
136         <visibility value='readonly' />
137         <cardinality min='1' max='1' />
138     </prop_pattern>
139     <prop_pattern name='start_method_timestamp' type='time'
140         required='false'>
141         <description>
142             <loctext xml:lang='C'>
143 Time the start method was last run.
144             </loctext>
145         </description>
146         <visibility value='readonly' />
147         <cardinality min='1' max='1' />
148     </prop_pattern>
149     <prop_pattern name='auxiliary_state' type='astring'
150         required='false'>
151         <description>
152             <loctext xml:lang='C'>
153 Restarter-set auxiliary information about the current state.
154             </loctext>
155         </description>
156         <visibility value='readonly' />
157         <cardinality min='1' max='1' />
158     </prop_pattern>
159     <prop_pattern name='auxiliary_fmri' type='astring'
160         required='false'>
161         <description>
162             <loctext xml:lang='C'>
163 Auxiliary fmri information for service state diagnosis.
164             </loctext>
165         </description>
166         <visibility value='hidden' />
167     </prop_pattern>
168     <prop_pattern name='state_timestamp' type='time'
169         required='false'>
170         <description>
171             <loctext xml:lang='C'>
172 Time the current state was reached.
173             </loctext>
174         </description>
175         <visibility value='readonly' />
176         <cardinality min='1' max='1' />
177     </prop_pattern>
178     <prop_pattern name='state' type='astring'
179         required='false'>
180         <description>
181             <loctext xml:lang='C'>
182 The current state of this service instance.
183             </loctext>
184         </description>
185         <visibility value='readonly' />
186         <cardinality min='1' max='1' />
187         <constraints>
188             <value name="online">
189                 <description>
190                     <loctext xml:lang='C'>

```

```

191 The instance is online and running.
192             </loctext>
193         </description>
194     </value>
195     <value name="offline">
196         <description>
197             <loctext xml:lang='C'>
198 The instance is enabled, but not yet running or available to run. The most comm
199             </loctext>
200         </description>
201     </value>
202     <value name="uninitialized">
203         <description>
204             <loctext xml:lang='C'>
205 The initial state for all instances before svc.startd has had a chance to evalua
206             </loctext>
207         </description>
208     </value>
209     <value name="degraded">
210         <description>
211             <loctext xml:lang='C'>
212 The instance is enabled and running or available to run. The instance, however,
213             </loctext>
214         </description>
215     </value>
216     <value name="disabled">
217         <description>
218             <loctext xml:lang='C'>
219 The instance is disabled.
220             </loctext>
221         </description>
222     </value>
223     <value name="maintenance">
224         <description>
225             <loctext xml:lang='C'>
226 The instance is enabled, but not able to run. Administrative action is required
227             </loctext>
228         </description>
229     </value>
230 </constraints>
231 </prop_pattern>
232     <prop_pattern name='next_state' type='astring'
233         required='false'>
234         <description>
235             <loctext xml:lang='C'>
236 The next expected state of this instance.
237             </loctext>
238         </description>
239         <visibility value='readonly' />
240         <cardinality min='1' max='1' />
241         <constraints>
242             <value name="online">
243                 <description>
244                     <loctext xml:lang='C'>
245 The instance is being started, and will soon be online and running. This transi
246                     </loctext>
247                 </description>
248             </value>
249             <value name="offline">
250                 <description>
251                     <loctext xml:lang='C'>
252 The instance has been temporarily stopped. Most instances will leave this state
253                     </loctext>
254                 </description>
255             </value>
256             <value name="degraded">

```

```

257         <description>
258             <loctext xml:lang='C'>
259 The instance will be enabled and available to run, although in a limited capacity
260             </loctext>
261         </description>
262     </value>
263     <value name="disabled">
264         <description>
265             <loctext xml:lang='C'>
266 The instance will be disabled.
267             </loctext>
268         </description>
269     </value>
270     <value name="maintenance">
271         <description>
272             <loctext xml:lang='C'>
273 The instance will be in maintenance, and administrative action will be required
274             </loctext>
275         </description>
276     </value>
277     <value name="none">
278         <description>
279             <loctext xml:lang='C'>
280 The instance is not currently transitioning between states.
281             </loctext>
282         </description>
283     </value>
284 </constraints>
285 </prop_pattern>
286 </pg_pattern>
287
288 <pg_pattern name='options' type='application'
289     target='this' required='false'>
290     <description>
291         <loctext xml:lang='C'>
292 Specify options for the svc.startd restarter.
293         </loctext>
294     </description>
295
296     <prop_pattern name='boot_messages' type='astring'
297         required='false'>
298         <description>
299             <loctext xml:lang='C'>
300 Define verbosity of messages to print to the console during boot.
301             </loctext>
302         </description>
303         <cardinality min='1' max='1' />
304         <constraints>
305             <value name='quiet'>
306                 <description>
307                     <loctext xml:lang='C'>
308 Issue console messages only on service failures.
309                     </loctext>
310                 </description>
311             </value>
312             <value name='verbose'>
313                 <description>
314                     <loctext xml:lang='C'>
315 Print a message per service started to indicate success or failure.
316                     </loctext>
317                 </description>
318             </value>
319         </constraints>
320         <choices>
321             <include_values type='constraints' />
322         </choices>

```

```

323     </prop_pattern>
324
325     <prop_pattern name='logging' type='astring'
326         required='false'>
327         <description>
328             <loctext xml:lang='C'>
329 Control the level of global service logging for svc.startd.
330             </loctext>
331         </description>
332         <cardinality min='1' max='1' />
333         <constraints>
334             <value name='quiet'>
335                 <description>
336                     <loctext xml:lang='C'>
337 Send error messages requiring administrative intervention to console, syslog, an
338                     </loctext>
339                 </description>
340             </value>
341             <value name='verbose'>
342                 <description>
343                     <loctext xml:lang='C'>
344 Sends a message per service started to the console, error messages requiring adm
345                     </loctext>
346                 </description>
347             </value>
348             <value name='debug'>
349                 <description>
350                     <loctext xml:lang='C'>
351 Send debug messages to svc.startd's global logfile, error messages requiring adm
352                     </loctext>
353                 </description>
354             </value>
355         </constraints>
356         <choices>
357             <include_values type='constraints' />
358         </choices>
359     </prop_pattern>
360
361     <prop_pattern name='milestone' type='astring'
362         required='false'>
363         <description>
364             <loctext xml:lang='C'>
365 An FRMI which defines the milestone used as the default boot level.
366             </loctext>
367         </description>
368         <cardinality min='1' max='1' />
369         <constraints>
370             <value
371                 name='svc:/milestone/single-user:default'>
372             </value>
373             <value
374                 name='svc:/milestone/multi-user:default'>
375             </value>
376             <value name=
377                 'svc:/milestone/multi-user-server:default'>
378             </value>
379             <value name='all'>
380                 <description>
381                     <loctext xml:lang='C'>
382 Start all enabled services.
383                     </loctext>
384                 </description>
385             </value>
386             <value name='none'>
387                 <description>
388                     <loctext xml:lang='C'>

```

```

389 Start no services.
390                                     </loctext>
391                                     </description>
392                                     </value>
393                                     </constraints>
394                                     </prop_pattern>
395 <prop_pattern name='info_events_all' type='boolean'
396   required='false'>
397   <description>
398     <loctext xml:lang='C'>
399 Override notification parameters and raise Information Events on all state trans
400 </loctext>
401   </description>
402   <visibility value='hidden' />
403   </prop_pattern>
404 </pg_pattern>

406 <pg_pattern name='system' type='framework'
407   target='this' required='false'>
408   <prop_pattern name='reconfigure' type='boolean'
409     required='false'>
410     <description>
411       <loctext xml:lang='C'>
412 Indicates that a reconfiguration reboot has been requested.
413       </loctext>
414     </description>
415     <visibility value='readonly' />
416     <cardinality min='1' max='1' />
417   </prop_pattern>
418 </pg_pattern>

420 <pg_pattern name='startd' type='framework'
421   target='delegate' required='false'>
422   <description>
423     <loctext xml:lang='C'>
424 Information about how a service instance is managed by svc.startd, which is supp
425     </loctext>
426   </description>

428   <prop_pattern name='duration' type='astring'
429     required='false'>
430     <description>
431       <loctext xml:lang='C'>
432 Defines the service's model.
433       </loctext>
434     </description>
435     <cardinality min='1' max='1' />
436     <constraints>
437       <value name='contract'>
438         <description>
439           <loctext xml:lang='C'>
440 A standard system daemon, which runs forever to provide a service. It is not co
441           </loctext>
442         </description>
443       </value>
444       <value name='transient'>
445         <description>
446           <loctext xml:lang='C'>
447 The service is online as soon as its start method returns -- child processes are
448           </loctext>
449         </description>
450       </value>
451       <value name='child'>
452         <description>
453           <loctext xml:lang='C'>
454 A service which runs for the lifetime of the child process, and is restarted whe

```

```

455                                     </loctext>
456                                     </description>
457                                     </value>
458                                     </constraints>
459                                     <choices>
460                                       <include_values type='constraints' />
461                                     </choices>
462 </prop_pattern>
463 <prop_pattern name='ignore_error' type='astring'
464   required='false'>
465   <description>
466     <loctext xml:lang='C'>
467 A list of events which should not be considered service errors by svc.startd.
468     </loctext>
469   </description>
470   <cardinality min='1' max='1' />
471   <!--
472     We won't recommend this as a choice since
473     it's only here to work around the fact
474     that startd defines this as a single
475     value rather than a value list *and*
476     templates doesn't currently take care
477     of assembling separately defined
478     values with the defined internal separator.
479   -->
480   <values>
481     <value name='signal,core'>
482       <description>
483         <loctext xml:lang='C'>
484 svc.startd should ignore core dumps and signals sent from outside the service.
485         </loctext>
486       </description>
487     </value>
488   </values>
489   <choices>
490     <value name='core'>
491       <description>
492         <loctext xml:lang='C'>
493 svc.startd should ignore core dumps from subprocesses.
494         </loctext>
495       </description>
496     </value>
497     <value name='signal'>
498       <description>
499         <loctext xml:lang='C'>
500 svc.startd should ignore signals sent from outside the service.
501         </loctext>
502       </description>
503     </value>
504     <value name='core,signal'>
505       <description>
506         <loctext xml:lang='C'>
507 svc.startd should ignore core dumps and signals sent from outside the service.
508         </loctext>
509       </description>
510     </value>
511   </choices>
512 </prop_pattern>
513 <prop_pattern name='need_session' type='boolean'
514   required='false'>
515   <description>
516     <loctext xml:lang='C'>
517 The instance should be launched in its own session per setpgrp(2).
518     </loctext>
519   </description>
520   <cardinality min='1' max='1' />

```

```

521         </prop_pattern>
522         <prop_pattern name='utmpx_prefix' type='astring'
523             required='false'>
524             <description>
525                 <loctext xml:lang='C'>
526 The instance requires that svc.startd create a valid utmpx entry prior to start
527                 </loctext>
528             </description>
529             <cardinality min='1' max='1' />
530         </prop_pattern>
531     </pg_pattern>

533     <pg_pattern name='start' type='method' target='delegate'
534         required='true'>
535         <description>
536             <loctext xml:lang='C'>
537 The start method defines how svc.startd should start the instance.
538             </loctext>
539         </description>
540         <prop_pattern name='exec' type='astring'
541             required='true'>
542             <common_name>
543                 <loctext xml:lang='C'>
544 method executable
545                 </loctext>
546             </common_name>
547             <description>
548                 <loctext xml:lang='C'>
549 The method executable may be a script, program, or keyword.
550                 </loctext>
551             </description>
552             <cardinality min='1' max='1' />
553             <values>
554                 <value name=':true'>
555                     <description>
556                         <loctext xml:lang='C'>
557 Always returns SMF_EXIT_OK. This token should be used when the start method is u
558                         </loctext>
559                     </description>
560                 </value>
561                 <value name=':kill [-signal]'>
562                     <description>
563                         <loctext xml:lang='C'>
564 Sends the specified signal, which is SIGTERM by default, to all processes in the
565                         </loctext>
566                 </value>
567             </values>
568             <choices>
569                 <include_values type='values' />
570             </choices>
571         </prop_pattern>

575     <prop_pattern name='type' type='astring'
576         required='true'>
577         <description>
578             <loctext xml:lang='C'>
579 A method may only be of type method.
580             </loctext>
581         </description>
582         <cardinality min='1' max='1' />
583         <constraints>
584             <value name="method" />
585         </constraints>
586     </prop_pattern>

```

```

588         <prop_pattern name='timeout_seconds' type='count'
589             required='true'>
590             <description>
591                 <loctext xml:lang='C'>
592 Number of seconds before the method is considered unresponsive. After the metho
593                 </loctext>
594             </description>
595             <cardinality min='1' max='1' />
596             <values>
597                 <value name="0">
598                     <common_name>
599                         <loctext xml:lang='C'>
600 infinite
601                         </loctext>
602                     </common_name>
603                     <description>
604                         <loctext xml:lang='C'>
605 This method will never time out.
606                         </loctext>
607                     </description>
608                 </value>
609                 <value name="-1">
610                     <common_name>
611                         <loctext xml:lang='C'>
612 infinite (legacy)
613                         </loctext>
614                     </common_name>
615                     <description>
616                         <loctext xml:lang='C'>
617 This method will never time out. 0 is the preferred value.
618                         </loctext>
619                     </description>
620                 </value>
621             </values>
622         </prop_pattern>

624     <!-- method_context direct properties -->
625     <prop_pattern name='working_directory' type='astring'
626         required='false'>
627         <description>
628             <loctext xml:lang='C'>
629 The working directory to launch the method from. ":default" can be used as a to
630             </loctext>
631         </description>
632         <cardinality min='1' max='1' />
633     </prop_pattern>
634     <prop_pattern name='project' type='astring'
635         required='false'>
636         <description>
637             <loctext xml:lang='C'>
638 The project ID in numeric or text form. :default can be used as a token to indi
639             </loctext>
640         </description>
641         <cardinality min='1' max='1' />
642     </prop_pattern>
643     <prop_pattern name='resource_pool' type='astring'
644         required='false'>
645         <common_name>
646             <loctext xml:lang='C'>
647 method context resource pool
648             </loctext>
649         </common_name>
650         <description>
651             <loctext xml:lang='C'>
652 The resource pool name on which to launch the method. :default can be used

```



```

653 as a token to indicate the pool specified in the project(4) entry given in
654 the project attribute.
655         </loctext>
656     </description>
657     <cardinality min='1' max='1' />
658 </prop_pattern>

660     <prop_pattern name='security_flags' type='astring'
661         required='false'>
662         <common_name>
663             <loctext xml:lang='C'>
664 method credential security flags
665             </loctext>
666         </common_name>
667         <description>
668             <loctext xml:lang='C'>
669 An optional string specifying the security flags as defined in security-flags(5)
670             </loctext>
671         </description>
672         <cardinality min='1' max='1' />
673         <internal_separators>,</internal_separators>
674     </prop_pattern>

676 #endif /* ! codereview */
677 <!-- method_credential properties -->
678 <prop_pattern name='user' type='astring'
679     required='false'>
680     <common_name>
681         <loctext xml:lang='C'>
682 method credential user
683         </loctext>
684     </common_name>
685     <description>
686         <loctext xml:lang='C'>
687 The user ID in numeric or text form.
688         </loctext>
689     </description>
690     <cardinality min='1' max='1' />
691 </prop_pattern>
692 <prop_pattern name='group' type='astring'
693     required='false'>
694     <common_name>
695         <loctext xml:lang='C'>
696 method credential group
697         </loctext>
698     </common_name>
699     <description>
700         <loctext xml:lang='C'>
701 The group ID in numeric or text form.
702         </loctext>
703     </description>
704     <cardinality min='1' max='1' />
705 </prop_pattern>
706 <prop_pattern name='supp_groups' type='astring'
707     required='false'>
708     <common_name>
709         <loctext xml:lang='C'>
710 method credential supplemental groups
711         </loctext>
712     </common_name>
713     <description>
714         <loctext xml:lang='C'>
715 An optional string that specifies the supplemental group memberships by ID,
716 in numeric or text form.
717         </loctext>
718     </description>

```

```

719         <cardinality min='1' max='1' />
720     <internal_separators>,</internal_separators>
721 </prop_pattern>
722 <prop_pattern name='privileges' type='astring'
723     required='false'>
724     <common_name>
725         <loctext xml:lang='C'>
726 method credential privileges
727         </loctext>
728     </common_name>
729     <description>
730         <loctext xml:lang='C'>
731 An optional string specifying the privilege set as defined in privileges(5).
732         </loctext>
733     </description>
734     <cardinality min='1' max='1' />
735     <internal_separators>,</internal_separators>
736 </prop_pattern>
737 <prop_pattern name='limit_privileges' type='astring'
738     required='false'>
739     <common_name>
740         <loctext xml:lang='C'>
741 method credential limit privilege set
742         </loctext>
743     </common_name>
744     <description>
745         <loctext xml:lang='C'>
746 An optional string specifying the limit privilege set as defined in
747 privileges(5).
748         </loctext>
749     </description>
750     <cardinality min='1' max='1' />
751     <internal_separators>,</internal_separators>
752 </prop_pattern>

754 <!-- method_profile properties -->
755 <prop_pattern name='use_profile' type='boolean'
756     required='false'>
757     <description>
758         <loctext xml:lang='C'>
759 A boolean that specifies whether the profile should be used instead of the
760 user, group, privileges, and limit_privileges properties.
761         </loctext>
762     </description>
763     <cardinality min='1' max='1' />
764 </prop_pattern>
765 <prop_pattern name='profile' type='astring'
766     required='false'>
767     <common_name>
768         <loctext xml:lang='C'>
769 method profile RBAC profile specification
770         </loctext>
771     </common_name>
772     <description>
773         <loctext xml:lang='C'>
774 The name of an RBAC (role-based access control) profile which, along with the
775 method executable, identifies an entry in exec_attr(4).
776         </loctext>
777     </description>
778     <cardinality min='1' max='1' />
779 </prop_pattern>

781 <!-- method_environment properties -->
782 <prop_pattern name='environment' type='astring'
783     required='false'>
784     <common_name>

```

```

785         <loctext xml:lang='C'>
786 method environment variables
787         </loctext>
788     </common_name>
789     <description>
790         <loctext xml:lang='C'>
791 Environment variables to insert into the environment of the method, in the
792 form of a number of NAME=value strings.
793         </loctext>
794     </description>
795 </prop_pattern>
796 </pg_pattern>

798     <pg_pattern name='stop' type='method' target='delegate'
799         required='true'>
800         <description>
801             <loctext xml:lang='C'>
802 The stop method defines how svc.startd should stop the instance.
803             </loctext>
804         </description>
805         <prop_pattern name='exec' type='astring'
806             required='true'>
807             <common_name>
808                 <loctext xml:lang='C'>
809 method executable
810                 </loctext>
811             </common_name>
812             <description>
813                 <loctext xml:lang='C'>
814 The method executable may be a script, program, or keyword.
815                 </loctext>
816             </description>
817             <cardinality min='1' max='1' />
818             <values>
819                 <value name=':true'>
820                     <description>
821                         <loctext xml:lang='C'>
822 Always returns SMF_EXIT_OK. This token should be used when the stop method is us
823                         </loctext>
824                     </description>
825                 </value>
826                 <value name=':kill [-signal]'>
827                     <description>
828                         <loctext xml:lang='C'>
829 Sends the specified signal, which is SIGTERM by default, to all processes in the
830                         </loctext>
831                     </description>
832                 </value>
833             </values>
834             <choices>
835                 <include_values type='values' />
836             </choices>
837         </prop_pattern>

839     <prop_pattern name='type' type='astring'
840         required='true'>
841         <description>
842             <loctext xml:lang='C'>
843 A method may only be of type method.
844             </loctext>
845         </description>
846         <cardinality min='1' max='1' />
847         <constraints>
848             <value name="method" />
849         </constraints>
850     </prop_pattern>

```

```

852         <prop_pattern name='timeout_seconds' type='count'
853             required='true'>
854             <description>
855                 <loctext xml:lang='C'>
856 Number of seconds before the method is considered unresponsive. After the metho
857                 </loctext>
858             </description>
859             <cardinality min='1' max='1' />
860             <values>
861                 <value name="0">
862                     <common_name>
863                         <loctext xml:lang='C'>
864 infinite
865                         </loctext>
866                     </common_name>
867                     <description>
868                         <loctext xml:lang='C'>
869 This method will never time out.
870                         </loctext>
871                     </description>
872                 </value>
873                 <value name="-1">
874                     <common_name>
875                         <loctext xml:lang='C'>
876 infinite (legacy)
877                         </loctext>
878                     </common_name>
879                     <description>
880                         <loctext xml:lang='C'>
881 This method will never time out. 0 is the preferred value.
882                         </loctext>
883                     </description>
884                 </value>
885             </values>
886         </prop_pattern>

888     <!-- method_context direct properties -->
889     <prop_pattern name='working_directory' type='astring'
890         required='false'>
891         <description>
892             <loctext xml:lang='C'>
893 The working directory to launch the method from. ":default" can be used as a to
894             </loctext>
895         </description>
896         <cardinality min='1' max='1' />
897     </prop_pattern>
898     <prop_pattern name='project' type='astring'
899         required='false'>
900         <description>
901             <loctext xml:lang='C'>
902 The project ID in numeric or text form. :default can be used as a token to indi
903             </loctext>
904         </description>
905         <cardinality min='1' max='1' />
906     </prop_pattern>
907     <prop_pattern name='resource_pool' type='astring'
908         required='false'>
909         <common_name>
910             <loctext xml:lang='C'>
911 method context resource pool
912             </loctext>
913         </common_name>
914         <description>
915             <loctext xml:lang='C'>
916 The resource pool name on which to launch the method. :default can be used

```

```

917 as a token to indicate the pool specified in the project(4) entry given in
918 the project attribute.
919         </loctext>
920     </description>
921     <cardinality min='1' max='1' />
922 </prop_pattern>
923
924     <prop_pattern name='security_flags' type='astring'
925         required='false'>
926         <common_name>
927             <loctext xml:lang='C'>
928 method credential security flags
929             </loctext>
930         </common_name>
931         <description>
932             <loctext xml:lang='C'>
933 An optional string specifying the security flags as defined in security-flags(5)
934             </loctext>
935         </description>
936         <cardinality min='1' max='1' />
937         <internal_separators>,</internal_separators>
938     </prop_pattern>
939
940 #endif /* ! codereview */
941 <!-- method_credential properties -->
942 <prop_pattern name='user' type='astring'
943     required='false'>
944     <common_name>
945         <loctext xml:lang='C'>
946 method credential user
947         </loctext>
948     </common_name>
949     <description>
950         <loctext xml:lang='C'>
951 The user ID in numeric or text form.
952         </loctext>
953     </description>
954     <cardinality min='1' max='1' />
955 </prop_pattern>
956 <prop_pattern name='group' type='astring'
957     required='false'>
958     <common_name>
959         <loctext xml:lang='C'>
960 method credential group
961         </loctext>
962     </common_name>
963     <description>
964         <loctext xml:lang='C'>
965 The group ID in numeric or text form.
966         </loctext>
967     </description>
968     <cardinality min='1' max='1' />
969 </prop_pattern>
970 <prop_pattern name='supp_groups' type='astring'
971     required='false'>
972     <common_name>
973         <loctext xml:lang='C'>
974 method credential supplemental groups
975         </loctext>
976     </common_name>
977     <description>
978         <loctext xml:lang='C'>
979 An optional string that specifies the supplemental group memberships by ID,
980 in numeric or text form.
981         </loctext>
982     </description>

```

```

983         <cardinality min='1' max='1' />
984     <internal_separators>,</internal_separators>
985 </prop_pattern>
986 <prop_pattern name='privileges' type='astring'
987     required='false'>
988     <common_name>
989         <loctext xml:lang='C'>
990 method credential privileges
991         </loctext>
992     </common_name>
993     <description>
994         <loctext xml:lang='C'>
995 An optional string specifying the privilege set as defined in privileges(5).
996         </loctext>
997     </description>
998     <cardinality min='1' max='1' />
999     <internal_separators>,</internal_separators>
1000 </prop_pattern>
1001 <prop_pattern name='limit_privileges' type='astring'
1002     required='false'>
1003     <common_name>
1004         <loctext xml:lang='C'>
1005 method credential limit privilege set
1006         </loctext>
1007     </common_name>
1008     <description>
1009         <loctext xml:lang='C'>
1010 An optional string specifying the limit privilege set as defined in
1011 privileges(5).
1012         </loctext>
1013     </description>
1014     <cardinality min='1' max='1' />
1015     <internal_separators>,</internal_separators>
1016 </prop_pattern>
1017
1018 <!-- method_profile properties -->
1019 <prop_pattern name='use_profile' type='boolean'
1020     required='false'>
1021     <description>
1022         <loctext xml:lang='C'>
1023 A boolean that specifies whether the profile should be used instead of the
1024 user, group, privileges, and limit_privileges properties.
1025         </loctext>
1026     </description>
1027     <cardinality min='1' max='1' />
1028 </prop_pattern>
1029 <prop_pattern name='profile' type='astring'
1030     required='false'>
1031     <common_name>
1032         <loctext xml:lang='C'>
1033 method profile RBAC profile specification
1034         </loctext>
1035     </common_name>
1036     <description>
1037         <loctext xml:lang='C'>
1038 The name of an RBAC (role-based access control) profile which, along with the
1039 method executable, identifies an entry in exec_attr(4).
1040         </loctext>
1041     </description>
1042     <cardinality min='1' max='1' />
1043 </prop_pattern>
1044
1045 <!-- method_environment properties -->
1046 <prop_pattern name='environment' type='astring'
1047     required='false'>
1048     <common_name>

```

```

1049         <loctext xml:lang='C'>
1050 method environment variables
1051         </loctext>
1052         </common_name>
1053         <description>
1054         <loctext xml:lang='C'>
1055 Environment variables to insert into the environment of the method, in the
1056 form of a number of NAME=value strings.
1057         </loctext>
1058         </description>
1059     </prop_pattern>
1060 </pg_pattern>

1062     <pg_pattern name='refresh' type='method' target='delegate'
1063     required='false'>
1064         <description>
1065         <loctext xml:lang='C'>
1066 The refresh method defines how svc.startd should upload new configuration to the
1067         </loctext>
1068         </description>
1069         <prop_pattern name='exec' type='astring'
1070         required='true'>
1071             <common_name>
1072             <loctext xml:lang='C'>
1073 method executable
1074             </loctext>
1075             </common_name>
1076             <description>
1077             <loctext xml:lang='C'>
1078 The method executable may be a script, program, or keyword.
1079             </loctext>
1080             </description>
1081             <cardinality min='1' max='1' />
1082             <values>
1083                 <value name=':true'>
1084                     <description>
1085                     <loctext xml:lang='C'>
1086 Always returns SMF_EXIT_OK.
1087                     </loctext>
1088                     </description>
1089                 </value>
1090                 <value name=':kill [-signal]'>
1091                     <description>
1092                     <loctext xml:lang='C'>
1093 Sends the specified signal, which is SIGTERM by default, to all processes in the
1094                     </loctext>
1095                     </description>
1096                 </value>
1097             </values>
1098             <choices>
1099                 <include_values type='values' />
1100             </choices>
1101         </prop_pattern>

1103     <prop_pattern name='type' type='astring'
1104     required='true'>
1105         <description>
1106         <loctext xml:lang='C'>
1107 A method may only be of type method.
1108         </loctext>
1109         </description>
1110         <cardinality min='1' max='1' />
1111         <constraints>
1112             <value name="method" />
1113         </constraints>
1114     </prop_pattern>

```

```

1116         <prop_pattern name='timeout_seconds' type='count'
1117         required='true'>
1118             <description>
1119             <loctext xml:lang='C'>
1120 Number of seconds before the method is considered unresponsive. After the metho
1121             </loctext>
1122             </description>
1123             <cardinality min='1' max='1' />
1124             <values>
1125                 <value name="0">
1126                     <common_name>
1127                     <loctext xml:lang='C'>
1128 infinite
1129                     </loctext>
1130                     </common_name>
1131                     <description>
1132                     <loctext xml:lang='C'>
1133 This method will never time out.
1134                     </loctext>
1135                     </description>
1136                 </value>
1137                 <value name="-1">
1138                     <common_name>
1139                     <loctext xml:lang='C'>
1140 infinite (legacy)
1141                     </loctext>
1142                     </common_name>
1143                     <description>
1144                     <loctext xml:lang='C'>
1145 This method will never time out. 0 is the preferred value.
1146                     </loctext>
1147                     </description>
1148                 </value>
1149             </values>
1150         </prop_pattern>

1152     <!-- method_context direct properties -->
1153     <prop_pattern name='working_directory' type='astring'
1154     required='false'>
1155         <description>
1156         <loctext xml:lang='C'>
1157 The working directory to launch the method from. ":default" can be used as a to
1158         </loctext>
1159         </description>
1160         <cardinality min='1' max='1' />
1161     </prop_pattern>
1162     <prop_pattern name='project' type='astring'
1163     required='false'>
1164         <description>
1165         <loctext xml:lang='C'>
1166 The project ID in numeric or text form. :default can be used as a token to indi
1167         </loctext>
1168         </description>
1169         <cardinality min='1' max='1' />
1170     </prop_pattern>
1171     <prop_pattern name='resource_pool' type='astring'
1172     required='false'>
1173         <common_name>
1174         <loctext xml:lang='C'>
1175 method context resource pool
1176         </loctext>
1177         </common_name>
1178         <description>
1179         <loctext xml:lang='C'>
1180 The resource pool name on which to launch the method. :default can be used

```

```

1181 as a token to indicate the pool specified in the project(4) entry given in
1182 the project attribute.
1183         </loctext>
1184     </description>
1185     <cardinality min='1' max='1'/>
1186 </prop_pattern>
1187 <prop_pattern name='security_flags' type='astring'
1188     required='false'>
1189     <common_name>
1190     <loctext xml:lang='C'>
1191 method security flags
1192     </loctext>
1193     </common_name>
1194     <description>
1195     <loctext xml:lang='C'>
1196 An optional string specifying the security flags as defined in security-flags(5)
1197     </loctext>
1198     </description>
1199     <cardinality min='1' max='1'/>
1200     <internal_separators>,</internal_separators>
1201 </prop_pattern>
1202 #endif /* ! codereview */

1204     <!-- method_credential properties -->
1205     <prop_pattern name='user' type='astring'
1206     required='false'>
1207     <common_name>
1208     <loctext xml:lang='C'>
1209 method credential user
1210     </loctext>
1211     </common_name>
1212     <description>
1213     <loctext xml:lang='C'>
1214 The user ID in numeric or text form.
1215     </loctext>
1216     </description>
1217     <cardinality min='1' max='1'/>
1218 </prop_pattern>
1219 <prop_pattern name='group' type='astring'
1220     required='false'>
1221     <common_name>
1222     <loctext xml:lang='C'>
1223 method credential group
1224     </loctext>
1225     </common_name>
1226     <description>
1227     <loctext xml:lang='C'>
1228 The group ID in numeric or text form.
1229     </loctext>
1230     </description>
1231     <cardinality min='1' max='1'/>
1232 </prop_pattern>
1233 <prop_pattern name='supp_groups' type='astring'
1234     required='false'>
1235     <common_name>
1236     <loctext xml:lang='C'>
1237 method credential supplemental groups
1238     </loctext>
1239     </common_name>
1240     <description>
1241     <loctext xml:lang='C'>
1242 An optional string that specifies the supplemental group memberships by ID,
1243 in numeric or text form.
1244     </loctext>
1245     </description>
1246     <cardinality min='1' max='1'/>

```

```

1247     <internal_separators>,</internal_separators>
1248 </prop_pattern>
1249 <prop_pattern name='privileges' type='astring'
1250     required='false'>
1251     <common_name>
1252     <loctext xml:lang='C'>
1253 method credential privileges
1254     </loctext>
1255     </common_name>
1256     <description>
1257     <loctext xml:lang='C'>
1258 An optional string specifying the privilege set as defined in privileges(5).
1259     </loctext>
1260     </description>
1261     <cardinality min='1' max='1'/>
1262     <internal_separators>,</internal_separators>
1263 </prop_pattern>
1264 <prop_pattern name='limit_privileges' type='astring'
1265     required='false'>
1266     <common_name>
1267     <loctext xml:lang='C'>
1268 method credential limit privilege set
1269     </loctext>
1270     </common_name>
1271     <description>
1272     <loctext xml:lang='C'>
1273 An optional string specifying the limit privilege set as defined in
1274 privileges(5).
1275     </loctext>
1276     </description>
1277     <cardinality min='1' max='1'/>
1278     <internal_separators>,</internal_separators>
1279 </prop_pattern>

1281     <!-- method_profile properties -->
1282     <prop_pattern name='use_profile' type='boolean'
1283     required='false'>
1284     <description>
1285     <loctext xml:lang='C'>
1286 A boolean that specifies whether the profile should be used instead of the
1287 user, group, privileges, and limit_privileges properties.
1288     </loctext>
1289     </description>
1290     <cardinality min='1' max='1'/>
1291 </prop_pattern>
1292 <prop_pattern name='profile' type='astring'
1293     required='false'>
1294     <common_name>
1295     <loctext xml:lang='C'>
1296 method profile RBAC profile specification
1297     </loctext>
1298     </common_name>
1299     <description>
1300     <loctext xml:lang='C'>
1301 The name of an RBAC (role-based access control) profile which, along with the
1302 method executable, identifies an entry in exec_attr(4).
1303     </loctext>
1304     </description>
1305     <cardinality min='1' max='1'/>
1306 </prop_pattern>

1308     <!-- method_environment properties -->
1309     <prop_pattern name='environment' type='astring'
1310     required='false'>
1311     <common_name>
1312     <loctext xml:lang='C'>

```

```
1313 method environment variables
1314                                     </loctext>
1315                                     </common_name>
1316                                     <description>
1317                                     <loctext xml:lang='C'>
1318 Environment variables to insert into the environment of the method, in the
1319 form of a number of NAME=value strings.
1320                                     </loctext>
1321                                     </description>
1322                                     </prop_pattern>
1323                                     </pg_pattern>
1325                                     </template>
1326 </service>
1328 </service_bundle>
```

```

*****
424967 Thu Jun 30 22:01:40 2016
new/usr/src/cmd/svc/svccfg/svccfg_libscf.c
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
_____unchanged_portion_omitted_____

9446 /*
9447  * As above, but for a method property group.
9448  */
9449 static void
9450 export_method(scf_propertygroup_t *pg, struct entity_elts *eelts)
9451 {
9452     xmlNodePtr n, env;
9453     char *str;
9454     int err = 0, nonenv, ret;
9455     uint8_t use_profile;
9456     struct pg_elts elts;
9457     xmlNodePtr ctxt = NULL;

9459     n = xmlNewNode(NULL, (xmlChar *)"exec_method");

9461     /* Get the required attributes. */

9463     /* name */
9464     if (scf_pg_get_name(pg, exp_str, max_scf_name_len + 1) < 0)
9465         scfdie();
9466     safe_setprop(n, name_attr, exp_str);

9468     /* type */
9469     if (pg_get_prop(pg, SCF_PROPERTY_TYPE, exp_prop) != 0 ||
9470         set_attr_from_prop(exp_prop, n, type_attr) != 0)
9471         err = 1;

9473     /* exec */
9474     if (pg_get_prop(pg, SCF_PROPERTY_EXEC, exp_prop) != 0 ||
9475         set_attr_from_prop(exp_prop, n, "exec") != 0)
9476         err = 1;

9478     /* timeout */
9479     if (pg_get_prop(pg, SCF_PROPERTY_TIMEOUT, exp_prop) == 0 &&
9480         prop_check_type(exp_prop, SCF_TYPE_COUNT) == 0 &&
9481         prop_get_val(exp_prop, exp_val) == 0) {
9482         uint64_t c;

9484         if (scf_value_get_count(exp_val, &c) != SCF_SUCCESS)
9485             scfdie();

9487         str = uu_msprintf("%llu", c);
9488         if (str == NULL)
9489             uu_die(gettext("Could not create string"));

9491         safe_setprop(n, "timeout_seconds", str);
9492         free(str);
9493     } else
9494         err = 1;

9496     if (err) {
9497         xmlFreeNode(n);

9499         export_pg(pg, eelts, SCE_ALL_VALUES);

9501         return;

```

```

9502     }

9505     /*
9506     * If we're going to have a method_context child, we need to know
9507     * before we iterate through the properties. Since method_context's
9508     * are optional, we don't want to complain about any properties
9509     * missing if none of them are there. Thus we can't use the
9510     * convenience functions.
9511     */
9512     nonenv =
9513         scf_pg_get_property(pg, SCF_PROPERTY_WORKING_DIRECTORY, NULL) ==
9514         SCF_SUCCESS ||
9515         scf_pg_get_property(pg, SCF_PROPERTY_PROJECT, NULL) ==
9516         SCF_SUCCESS ||
9517         scf_pg_get_property(pg, SCF_PROPERTY_RESOURCE_POOL, NULL) ==
9518         SCF_SUCCESS ||
9519         scf_pg_get_property(pg, SCF_PROPERTY_SECFLAGS, NULL) ==
9520         SCF_SUCCESS ||
9521     #endif /* ! codereview */
9522         scf_pg_get_property(pg, SCF_PROPERTY_USE_PROFILE, NULL) ==
9523         SCF_SUCCESS;

9525     if (nonenv) {
9526         ctxt = xmlNewNode(NULL, (xmlChar *)"method_context");
9527         if (ctxt == NULL)
9528             uu_die(emsmsg_create_xml);

9530         if (pg_get_prop(pg, SCF_PROPERTY_WORKING_DIRECTORY, exp_prop) ==
9531             0 &&
9532             set_attr_from_prop_default(exp_prop, ctxt,
9533                 "working_directory", ":default") != 0)
9534             err = 1;

9536         if (pg_get_prop(pg, SCF_PROPERTY_PROJECT, exp_prop) == 0 &&
9537             set_attr_from_prop_default(exp_prop, ctxt, "project",
9538                 ":default") != 0)
9539             err = 1;

9541         if (pg_get_prop(pg, SCF_PROPERTY_RESOURCE_POOL, exp_prop) ==
9542             0 &&
9543             set_attr_from_prop_default(exp_prop, ctxt,
9544                 "resource_pool", ":default") != 0)
9545             err = 1;

9547         if (pg_get_prop(pg, SCF_PROPERTY_SECFLAGS, exp_prop) == 0 &&
9548             set_attr_from_prop_default(exp_prop, ctxt,
9549                 "security_flags", ":default") != 0)
9550             err = 1;

9552     #endif /* ! codereview */
9553     /*
9554     * We only want to complain about profile or credential
9555     * properties if we will use them. To determine that we must
9556     * examine USE_PROFILE.
9557     */
9558     if (pg_get_prop(pg, SCF_PROPERTY_USE_PROFILE, exp_prop) == 0 &&
9559         prop_check_type(exp_prop, SCF_TYPE_BOOLEAN) == 0 &&
9560         prop_get_val(exp_prop, exp_val) == 0) {
9561         if (scf_value_get_boolean(exp_val, &use_profile) !=
9562             SCF_SUCCESS) {
9563             scfdie();
9564         }

9566         if (use_profile) {
9567             xmlNodePtr prof;

```

```

9569         prof = xmlNewChild(ctxt, NULL,
9570             (xmlChar *)"method_profile", NULL);
9571         if (prof == NULL)
9572             uu_die(msg_create_xml);
9573
9574         if (pg_get_prop(pg, SCF_PROPERTY_PROFILE,
9575             exp_prop) != 0 ||
9576             set_attr_from_prop(exp_prop, prof,
9577                 name_attr) != 0)
9578             err = 1;
9579     } else {
9580         xmlNodePtr cred;
9581
9582         cred = xmlNewChild(ctxt, NULL,
9583             (xmlChar *)"method_credential", NULL);
9584         if (cred == NULL)
9585             uu_die(msg_create_xml);
9586
9587         if (pg_get_prop(pg, SCF_PROPERTY_USER,
9588             exp_prop) != 0 ||
9589             set_attr_from_prop(exp_prop, cred,
9590                 "user") != 0) {
9591             err = 1;
9592         }
9593
9594         if (pg_get_prop(pg, SCF_PROPERTY_GROUP,
9595             exp_prop) == 0 &&
9596             set_attr_from_prop_default(exp_prop, cred,
9597                 "group", ":default") != 0)
9598             err = 1;
9599
9600         if (pg_get_prop(pg, SCF_PROPERTY_SUPP_GROUPS,
9601             exp_prop) == 0 &&
9602             set_attr_from_prop_default(exp_prop, cred,
9603                 "supp_groups", ":default") != 0)
9604             err = 1;
9605
9606         if (pg_get_prop(pg, SCF_PROPERTY_PRIVILEGES,
9607             exp_prop) == 0 &&
9608             set_attr_from_prop_default(exp_prop, cred,
9609                 "privileges", ":default") != 0)
9610             err = 1;
9611
9612         if (pg_get_prop(pg,
9613             SCF_PROPERTY_LIMIT_PRIVILEGES,
9614             exp_prop) == 0 &&
9615             set_attr_from_prop_default(exp_prop, cred,
9616                 "limit_privileges", ":default") != 0)
9617             err = 1;
9618     }
9619 }
9620
9621 if ((env = export_method_environment(pg)) != NULL) {
9622     if (ctxt == NULL) {
9623         ctxt = xmlNewNode(NULL, (xmlChar *)"method_context");
9624         if (ctxt == NULL)
9625             uu_die(msg_create_xml);
9626     }
9627     (void) xmlAddChild(ctxt, env);
9628 }
9629
9630 if (env != NULL || (nonenv && err == 0))
9631     (void) xmlAddChild(n, ctxt);
9632 else
9633

```

```

9634         xmlFreeNode(ctxt);
9635
9636         nonenv = (err == 0);
9637
9638         if (scf_iter_pg_properties(exp_prop_iter, pg) != SCF_SUCCESS)
9639             scfdie();
9640
9641         (void) memset(&elts, 0, sizeof (elts));
9642
9643         while ((ret = scf_iter_next_property(exp_prop_iter, exp_prop)) == 1) {
9644             if (scf_property_get_name(exp_prop, exp_str, exp_str_sz) < 0)
9645                 scfdie();
9646
9647             if (strcmp(exp_str, SCF_PROPERTY_TYPE) == 0 ||
9648                 strcmp(exp_str, SCF_PROPERTY_EXEC) == 0 ||
9649                 strcmp(exp_str, SCF_PROPERTY_TIMEOUT) == 0) {
9650                 continue;
9651             } else if (strcmp(exp_str, SCF_PROPERTY_STABILITY) == 0) {
9652                 xmlNodePtr m;
9653
9654                 m = xmlNewNode(NULL, (xmlChar *)"stability");
9655                 if (m == NULL)
9656                     uu_die(msg_create_xml);
9657
9658                 if (set_attr_from_prop(exp_prop, m, value_attr) == 0) {
9659                     elts.stability = m;
9660                     continue;
9661                 }
9662
9663                 xmlFreeNode(m);
9664             } else if (strcmp(exp_str, SCF_PROPERTY_WORKING_DIRECTORY) ==
9665                 0 ||
9666                 strcmp(exp_str, SCF_PROPERTY_PROJECT) == 0 ||
9667                 strcmp(exp_str, SCF_PROPERTY_RESOURCE_POOL) == 0 ||
9668                 strcmp(exp_str, SCF_PROPERTY_USE_PROFILE) == 0) {
9669                 if (nonenv)
9670                     continue;
9671             } else if (strcmp(exp_str, SCF_PROPERTY_USER) == 0 ||
9672                 strcmp(exp_str, SCF_PROPERTY_GROUP) == 0 ||
9673                 strcmp(exp_str, SCF_PROPERTY_SUPP_GROUPS) == 0 ||
9674                 strcmp(exp_str, SCF_PROPERTY_PRIVILEGES) == 0 ||
9675                 strcmp(exp_str, SCF_PROPERTY_LIMIT_PRIVILEGES) == 0 ||
9676                 strcmp(exp_str, SCF_PROPERTY_SECFLAGS) == 0) {
9677                 if (nonenv && !use_profile)
9678                     continue;
9679             } else if (strcmp(exp_str, SCF_PROPERTY_PROFILE) == 0) {
9680                 if (nonenv && use_profile)
9681                     continue;
9682             } else if (strcmp(exp_str, SCF_PROPERTY_ENVIRONMENT) == 0) {
9683                 if (env != NULL)
9684                     continue;
9685             }
9686
9687             export_property(exp_prop, exp_str, &elts, SCE_ALL_VALUES);
9688         }
9689         if (ret == -1)
9690             scfdie();
9691
9692         (void) xmlAddChild(n, elts.stability);
9693         (void) xmlAddChildList(n, elts.propvals);
9694         (void) xmlAddChildList(n, elts.properties);
9695
9696         if (elts->exec_methods == NULL)
9697             elts->exec_methods = n;
9698         else

```



```

9699         (void) xmlAddSibling(eelts->exec_methods, n);
9700     }
_____unchanged_portion_omitted_____
9813 static void
9814 export_method_context(scf_propertygroup_t *pg, struct entity_elts *elts)
9815 {
9816     xmlNodePtr n, prof, cred, env;
9817     uint8_t use_profile;
9818     int ret, err = 0;
9820     n = xmlNewNode(NULL, (xmlChar *)"method_context");
9822     env = export_method_environment(pg);
9824     /* Need to know whether we'll use a profile or not. */
9825     if (pg_get_prop(pg, SCF_PROPERTY_USE_PROFILE, exp_prop) == 0 &&
9826         prop_check_type(exp_prop, SCF_TYPE_BOOLEAN) == 0 &&
9827         prop_get_val(exp_prop, exp_val) == 0) {
9828         if (scf_value_get_boolean(exp_val, &use_profile) != SCF_SUCCESS)
9829             scfdie();
9831         if (use_profile)
9832             prof =
9833                 xmlNewChild(n, NULL, (xmlChar *)"method_profile",
9834                             NULL);
9835         else
9836             cred =
9837                 xmlNewChild(n, NULL, (xmlChar *)"method_credential",
9838                             NULL);
9839     }
9841     if (env != NULL)
9842         (void) xmlAddChild(n, env);
9844     if (scf_iter_pg_properties(exp_prop_iter, pg) != SCF_SUCCESS)
9845         scfdie();
9847     while ((ret = scf_iter_next_property(exp_prop_iter, exp_prop)) == 1) {
9848         if (scf_property_get_name(exp_prop, exp_str, exp_str_sz) < 0)
9849             scfdie();
9851         if (strcmp(exp_str, SCF_PROPERTY_WORKING_DIRECTORY) == 0) {
9852             if (set_attr_from_prop(exp_prop, n,
9853                 "working_directory") != 0)
9854                 err = 1;
9855         } else if (strcmp(exp_str, SCF_PROPERTY_PROJECT) == 0) {
9856             if (set_attr_from_prop(exp_prop, n, "project") != 0)
9857                 err = 1;
9858         } else if (strcmp(exp_str, SCF_PROPERTY_RESOURCE_POOL) == 0) {
9859             if (set_attr_from_prop(exp_prop, n,
9860                 "resource_pool") != 0)
9861                 err = 1;
9862         } else if (strcmp(exp_str, SCF_PROPERTY_SECFLAGS) == 0) {
9863             if (set_attr_from_prop(exp_prop, n,
9864                 "security_flags") != 0)
9865                 err = 1;
9866 #endif /* ! codereview */
9867         } else if (strcmp(exp_str, SCF_PROPERTY_USE_PROFILE) == 0) {
9868             /* EMPTY */
9869         } else if (strcmp(exp_str, SCF_PROPERTY_USER) == 0) {
9870             if (use_profile ||
9871                 set_attr_from_prop(exp_prop, cred, "user") != 0)
9872                 err = 1;
9873         } else if (strcmp(exp_str, SCF_PROPERTY_GROUP) == 0) {
9874             if (use_profile ||

```

```

9875         set_attr_from_prop(exp_prop, cred, "group") != 0)
9876             err = 1;
9877     } else if (strcmp(exp_str, SCF_PROPERTY_SUPP_GROUPS) == 0) {
9878         if (use_profile || set_attr_from_prop(exp_prop, cred,
9879             "supp_groups") != 0)
9880             err = 1;
9881     } else if (strcmp(exp_str, SCF_PROPERTY_PRIVILEGES) == 0) {
9882         if (use_profile || set_attr_from_prop(exp_prop, cred,
9883             "privileges") != 0)
9884             err = 1;
9885     } else if (strcmp(exp_str, SCF_PROPERTY_LIMIT_PRIVILEGES) ==
9886         0) {
9887         if (use_profile || set_attr_from_prop(exp_prop, cred,
9888             "limit_privileges") != 0)
9889             err = 1;
9890     } else if (strcmp(exp_str, SCF_PROPERTY_PROFILE) == 0) {
9891         if (!use_profile || set_attr_from_prop(exp_prop,
9892             prof, name_attr) != 0)
9893             err = 1;
9894     } else {
9895         /* Can't have generic properties in method_context's */
9896         err = 1;
9897     }
9898     }
9899     if (ret == -1)
9900         scfdie();
9902     if (err && env == NULL) {
9903         xmlFreeNode(n);
9904         export_pg(pg, elts, SCE_ALL_VALUES);
9905         return;
9906     }
9908     elts->method_context = n;
9909 }
9911 /*
9912  * Given a dependency property group in the tfmri entity (target fmri), return
9913  * a dependent element which represents it.
9914  */
9915 static xmlNodePtr
9916 export_dependent(scf_propertygroup_t *pg, const char *name, const char *tfmri)
9917 {
9918     uint8_t b;
9919     xmlNodePtr n, sf;
9920     int err = 0, ret;
9921     struct pg_elts pgelts;
9923     /*
9924      * If external isn't set to true then exporting the service will
9925      * export this as a normal dependency, so we should stop to avoid
9926      * duplication.
9927      */
9928     if (scf_pg_get_property(pg, scf_property_external, exp_prop) != 0 ||
9929         scf_property_get_value(exp_prop, exp_val) != 0 ||
9930         scf_value_get_boolean(exp_val, &b) != 0 || !b) {
9931         if (g_verbose) {
9932             warn(gettext("Dependent \"%s\" cannot be exported \"
9933                 \"properly because the \"%s\" property of the \"
9934                 \"%s\" dependency of %s is not set to true.\n\"),
9935                 name, scf_property_external, name, tfmri);
9936         }
9938         return (NULL);
9939     }

```

```

9941     n = xmlNewNode(NULL, (xmlChar *)"dependent");
9942     if (n == NULL)
9943         uu_die(msg_create_xml);
9944
9945     safe_setprop(n, name_attr, name);
9946
9947     /* Get the required attributes */
9948     if (pg_get_prop(pg, SCF_PROPERTY_RESTART_ON, exp_prop) != 0 ||
9949         set_attr_from_prop(exp_prop, n, "restart_on") != 0)
9950         err = 1;
9951
9952     if (pg_get_prop(pg, SCF_PROPERTY_GROUPING, exp_prop) != 0 ||
9953         set_attr_from_prop(exp_prop, n, "grouping") != 0)
9954         err = 1;
9955
9956     if (pg_get_prop(pg, SCF_PROPERTY_ENTITIES, exp_prop) == 0 &&
9957         prop_check_type(exp_prop, SCF_TYPE_FMRI) == 0 &&
9958         prop_get_val(exp_prop, exp_val) == 0) {
9959         /* EMPTY */
9960     } else
9961         err = 1;
9962
9963     if (err) {
9964         xmlFreeNode(n);
9965         return (NULL);
9966     }
9967
9968     sf = xmlNewChild(n, NULL, (xmlChar *)"service_fmri", NULL);
9969     if (sf == NULL)
9970         uu_die(msg_create_xml);
9971
9972     safe_setprop(sf, value_attr, tfmri);
9973
9974     /*
9975     * Now add elements for the other properties.
9976     */
9977     if (scf_iter_pg_properties(exp_prop_iter, pg) != SCF_SUCCESS)
9978         scfdie();
9979
9980     (void) memset(&pgelts, 0, sizeof (pgelts));
9981
9982     while ((ret = scf_iter_next_property(exp_prop_iter, exp_prop)) == 1) {
9983         if (scf_property_get_name(exp_prop, exp_str_sz) < 0)
9984             scfdie();
9985
9986         if (strcmp(exp_str, scf_property_external) == 0 ||
9987             strcmp(exp_str, SCF_PROPERTY_RESTART_ON) == 0 ||
9988             strcmp(exp_str, SCF_PROPERTY_GROUPING) == 0 ||
9989             strcmp(exp_str, SCF_PROPERTY_ENTITIES) == 0) {
9990             continue;
9991         } else if (strcmp(exp_str, SCF_PROPERTY_TYPE) == 0) {
9992             if (prop_check_type(exp_prop, SCF_TYPE_ASTRING) == 0 &&
9993                 prop_get_val(exp_prop, exp_val) == 0) {
9994                 char type[sizeof ("service") + 1];
9995
9996                 if (scf_value_get_astring(exp_val, type,
9997                     sizeof (type)) < 0)
9998                     scfdie();
9999
10000                 if (strcmp(type, "service") == 0)
10001                     continue;
10002             }
10003         } else if (strcmp(exp_str, SCF_PROPERTY_STABILITY) == 0) {
10004             xmlNodePtr s;
10005
10006             s = xmlNewNode(NULL, (xmlChar *)"stability");

```

```

10007         if (s == NULL)
10008             uu_die(msg_create_xml);
10009
10010         if (set_attr_from_prop(exp_prop, s, value_attr) == 0) {
10011             pgelts.stability = s;
10012             continue;
10013         }
10014
10015         xmlFreeNode(s);
10016     }
10017
10018     export_property(exp_prop, exp_str, &pgelts, SCE_ALL_VALUES);
10019 }
10020 if (ret == -1)
10021     scfdie();
10022
10023 (void) xmlAddChild(n, pgelts.stability);
10024 (void) xmlAddChildList(n, pgelts.propvals);
10025 (void) xmlAddChildList(n, pgelts.properties);
10026
10027 return (n);
10028 }
10029
10030 static void
10031 export_dependents(scf_propertygroup_t *pg, struct entity_elts *eelts)
10032 {
10033     scf_propertygroup_t *opg;
10034     scf_iter_t *iter;
10035     char *type, *fmri;
10036     int ret;
10037     struct pg_elts pgelts;
10038     xmlNodePtr n;
10039     scf_error_t serr;
10040
10041     if ((opg = scf_pg_create(g_hndl)) == NULL ||
10042         (iter = scf_iter_create(g_hndl)) == NULL)
10043         scfdie();
10044
10045     /* Can't use exp_prop_iter due to export_dependent(). */
10046     if (scf_iter_pg_properties(iter, pg) != SCF_SUCCESS)
10047         scfdie();
10048
10049     type = safe_malloc(max_scf_pg_type_len + 1);
10050
10051     /* Get an extra byte so we can tell if values are too long. */
10052     fmri = safe_malloc(max_scf_fmri_len + 2);
10053
10054     (void) memset(&pgelts, 0, sizeof (pgelts));
10055
10056     while ((ret = scf_iter_next_property(iter, exp_prop)) == 1) {
10057         void *entity;
10058         int isservice;
10059         scf_type_t ty;
10060
10061         if (scf_property_type(exp_prop, &ty) != SCF_SUCCESS)
10062             scfdie();
10063
10064         if ((ty != SCF_TYPE_ASTRING &&
10065             prop_check_type(exp_prop, SCF_TYPE_FMRI) != 0) ||
10066             prop_get_val(exp_prop, exp_val) != 0) {
10067             export_property(exp_prop, NULL, &pgelts,
10068                 SCE_ALL_VALUES);
10069             continue;
10070         }
10071
10072         if (scf_property_get_name(exp_prop, exp_str, exp_str_sz) < 0)

```

```

10073         scfdie();
10075     if (scf_value_get_astring(exp_val, fmri,
10076         max_scf_fmri_len + 2) < 0)
10077         scfdie();
10079     /* Look for a dependency group in the target fmri. */
10080     serr = fmri_to_entity(g_hdl, fmri, &entity, &isservice);
10081     switch (serr) {
10082     case SCF_ERROR_NONE:
10083         break;
10085     case SCF_ERROR_NO_MEMORY:
10086         uu_die(gettext("Out of memory.\n"));
10087         /* NOTREACHED */
10089     case SCF_ERROR_INVALID_ARGUMENT:
10090         if (g_verbose) {
10091             if (scf_property_to_fmri(exp_prop, fmri,
10092             max_scf_fmri_len + 2) < 0)
10093                 scfdie();
10095             warn(gettext("The value of %s is not a valid "
10096             "FMRI.\n"), fmri);
10097         }
10099         export_property(exp_prop, exp_str, &pgelts,
10100             SCE_ALL_VALUES);
10101         continue;
10103     case SCF_ERROR_CONSTRAINT_VIOLATED:
10104         if (g_verbose) {
10105             if (scf_property_to_fmri(exp_prop, fmri,
10106             max_scf_fmri_len + 2) < 0)
10107                 scfdie();
10109             warn(gettext("The value of %s does not specify "
10110             "a service or an instance.\n"), fmri);
10111         }
10113         export_property(exp_prop, exp_str, &pgelts,
10114             SCE_ALL_VALUES);
10115         continue;
10117     case SCF_ERROR_NOT_FOUND:
10118         if (g_verbose) {
10119             if (scf_property_to_fmri(exp_prop, fmri,
10120             max_scf_fmri_len + 2) < 0)
10121                 scfdie();
10123             warn(gettext("The entity specified by %s does "
10124             "not exist.\n"), fmri);
10125         }
10127         export_property(exp_prop, exp_str, &pgelts,
10128             SCE_ALL_VALUES);
10129         continue;
10131     default:
10132     #ifndef NDEBUG
10133         (void) fprintf(stderr, "%s:%d: %s() failed with "
10134         "unexpected error %d.\n", __FILE__, __LINE__,
10135         "fmri_to_entity", serr);
10136     #endif
10137         abort();
10138     }

```

```

10140         if (entity_get_pg(entity, isservice, exp_str, opg) != 0) {
10141             if (scf_error() != SCF_ERROR_NOT_FOUND)
10142                 scfdie();
10144             warn(gettext("Entity %s is missing dependency property "
10145             "group %s.\n"), fmri, exp_str);
10147             export_property(exp_prop, NULL, &pgelts,
10148             SCE_ALL_VALUES);
10149             continue;
10150         }
10152         if (scf_pg_get_type(opg, type, max_scf_pg_type_len + 1) < 0)
10153             scfdie();
10155         if (strcmp(type, SCF_GROUP_DEPENDENCY) != 0) {
10156             if (scf_pg_to_fmri(opg, fmri, max_scf_fmri_len + 2) < 0)
10157                 scfdie();
10159             warn(gettext("Property group %s is not of "
10160             "expected type %s.\n"), fmri, SCF_GROUP_DEPENDENCY);
10162             export_property(exp_prop, NULL, &pgelts,
10163             SCE_ALL_VALUES);
10164             continue;
10165         }
10167         n = export_dependent(opg, exp_str, fmri);
10168         if (n == NULL) {
10169             export_property(exp_prop, exp_str, &pgelts,
10170             SCE_ALL_VALUES);
10171         } else {
10172             if (eelts->dependents == NULL)
10173                 eelts->dependents = n;
10174             else
10175                 (void) xmlAddSibling(eelts->dependents,
10176                 n);
10177         }
10178     }
10179     if (ret == -1)
10180         scfdie();
10182     free(fmri);
10183     free(type);
10185     scf_iter_destroy(iter);
10186     scf_pg_destroy(opg);
10188     if (pgelts.propvals != NULL || pgelts.properties != NULL)
10189         export_pg_elts(&pgelts, SCF_PG_DEPENDENTS, scf_group_framework,
10190         eelts);
10191 }
10193 static void
10194 make_node(xmlNodePtr *nodep, const char *name)
10195 {
10196     if (*nodep == NULL) {
10197         *nodep = xmlNewNode(NULL, (xmlChar *)name);
10198         if (*nodep == NULL)
10199             uu_die(msg_create_xml);
10200     }
10201 }
10203 static xmlNodePtr
10204 export_tm_loctext(scf_propertygroup_t *pg, const char *parname)

```

```

10205 {
10206     int ret;
10207     xmlNodePtr parent = NULL;
10208     xmlNodePtr loctext = NULL;
10210
10210     if (scf_iter_pg_properties(exp_prop_iter, pg) != SCF_SUCCESS)
10211         scfdie();
10213
10213     while ((ret = scf_iter_next_property(exp_prop_iter, exp_prop)) == 1) {
10214         if (prop_check_type(exp_prop, SCF_TYPE_USTRING) != 0 ||
10215             prop_get_val(exp_prop, exp_val) != 0)
10216             continue;
10218
10218         if (scf_value_get_ustring(exp_val, exp_str, exp_str_sz) < 0)
10219             scfdie();
10221
10221         make_node(&parent, parname);
10222         loctext = xmlNewTextChild(parent, NULL, (xmlChar *)"loctext",
10223             (xmlChar *)exp_str);
10224         if (loctext == NULL)
10225             uu_die(msg_create_xml);
10227
10227         if (scf_property_get_name(exp_prop, exp_str, exp_str_sz) < 0)
10228             scfdie();
10230
10230         safe_setprop(loctext, "xml:lang", exp_str);
10231     }
10233
10233     if (ret == -1)
10234         scfdie();
10236
10236     return (parent);
10237 }
10239
10239 static xmlNodePtr
10240 export_tm_manpage(scf_propertygroup_t *pg)
10241 {
10242     xmlNodePtr manpage = xmlNewNode(NULL, (xmlChar *)"manpage");
10243     if (manpage == NULL)
10244         uu_die(msg_create_xml);
10246
10246     if (pg_get_prop(pg, SCF_PROPERTY_TM_TITLE, exp_prop) != 0 ||
10247         set_attr_from_prop(exp_prop, manpage, "title") != 0 ||
10248         pg_get_prop(pg, SCF_PROPERTY_TM_SECTION, exp_prop) != 0 ||
10249         set_attr_from_prop(exp_prop, manpage, "section") != 0) {
10250         xmlFreeNode(manpage);
10251         return (NULL);
10252     }
10254
10254     if (pg_get_prop(pg, SCF_PROPERTY_TM_MANPATH, exp_prop) == 0)
10255         (void) set_attr_from_prop_default(exp_prop,
10256             manpage, "manpath", ".default");
10258
10258     return (manpage);
10259 }
10261
10261 static xmlNodePtr
10262 export_tm_doc_link(scf_propertygroup_t *pg)
10263 {
10264     xmlNodePtr doc_link = xmlNewNode(NULL, (xmlChar *)"doc_link");
10265     if (doc_link == NULL)
10266         uu_die(msg_create_xml);
10268
10268     if (pg_get_prop(pg, SCF_PROPERTY_TM_NAME, exp_prop) != 0 ||
10269         set_attr_from_prop(exp_prop, doc_link, "name") != 0 ||
10270         pg_get_prop(pg, SCF_PROPERTY_TM_URI, exp_prop) != 0 ||

```

```

10271         set_attr_from_prop(exp_prop, doc_link, "uri") != 0) {
10272             xmlFreeNode(doc_link);
10273             return (NULL);
10274         }
10275         return (doc_link);
10276     }
10278
10278     /*
10279     * Process template information for a service or instances.
10280     */
10281     static void
10282     export_template(scf_propertygroup_t *pg, struct entity_elts *elts,
10283         struct template_elts *telts)
10284     {
10285         size_t mansz = strlen(SCF_PG_TM_MAN_PREFIX);
10286         size_t docsz = strlen(SCF_PG_TM_DOC_PREFIX);
10287         xmlNodePtr child = NULL;
10289
10289         if (scf_pg_get_name(pg, exp_str, exp_str_sz) < 0)
10290             scfdie();
10292
10292         if (strcmp(exp_str, SCF_PG_TM_COMMON_NAME) == 0) {
10293             telts->common_name = export_tm_loctext(pg, "common_name");
10294             if (telts->common_name == NULL)
10295                 export_pg(pg, elts, SCE_ALL_VALUES);
10296             return;
10297         } else if (strcmp(exp_str, SCF_PG_TM_DESCRIPTION) == 0) {
10298             telts->description = export_tm_loctext(pg, "description");
10299             if (telts->description == NULL)
10300                 export_pg(pg, elts, SCE_ALL_VALUES);
10301             return;
10302         }
10304
10304         if (strncmp(exp_str, SCF_PG_TM_MAN_PREFIX, mansz) == 0) {
10305             child = export_tm_manpage(pg);
10306         } else if (strncmp(exp_str, SCF_PG_TM_DOC_PREFIX, docsz) == 0) {
10307             child = export_tm_doc_link(pg);
10308         }
10310
10310         if (child != NULL) {
10311             make_node(&telts->documentation, "documentation");
10312             (void) xmlAddChild(telts->documentation, child);
10313         } else {
10314             export_pg(pg, elts, SCE_ALL_VALUES);
10315         }
10316     }
10318
10318     /*
10319     * Process parameter and paramval elements
10320     */
10321     static void
10322     export_parameter(scf_property_t *prop, const char *name,
10323         struct params_elts *elts)
10324     {
10325         xmlNodePtr param;
10326         scf_error_t err = 0;
10327         int ret;
10329
10329         if (scf_property_get_value(prop, exp_val) == SCF_SUCCESS) {
10330             if ((param = xmlNewNode(NULL, (xmlChar *)"paramval")) == NULL)
10331                 uu_die(msg_create_xml);
10333
10333             safe_setprop(param, name_attr, name);
10335
10335             if (scf_value_get_as_string(exp_val, exp_str, exp_str_sz) < 0)
10336                 scfdie();

```

```

10337     safe_setprop(param, value_attr, exp_str);
10339
10339     if (elts->paramval == NULL)
10340         elts->paramval = param;
10341     else
10342         (void) xmlAddSibling(elts->paramval, param);
10344
10344     return;
10345 }
10347
10347     err = scf_error();
10349
10349     if (err != SCF_ERROR_CONSTRAINT_VIOLATED &&
10350         err != SCF_ERROR_NOT_FOUND)
10351         scfdie();
10353
10353     if ((param = xmlNewNode(NULL, (xmlChar *)"parameter")) == NULL)
10354         uu_die(msg_create_xml);
10356
10356     safe_setprop(param, name_attr, name);
10358
10358     if (err == SCF_ERROR_CONSTRAINT_VIOLATED) {
10359         if (scf_iter_property_values(exp_val_iter, prop) != SCF_SUCCESS)
10360             scfdie();
10362
10362         while ((ret = scf_iter_next_value(exp_val_iter, exp_val)) ==
10363             1) {
10364             xmlNodePtr vn;
10366
10366             if ((vn = xmlNewChild(param, NULL,
10367                 (xmlChar *)"value_node", NULL)) == NULL)
10368                 uu_die(msg_create_xml);
10370
10370             if (scf_value_get_as_string(exp_val, exp_str,
10371                 exp_str_sz) < 0)
10372                 scfdie();
10374
10374                 safe_setprop(vn, value_attr, exp_str);
10375             }
10376             if (ret != 0)
10377                 scfdie();
10378         }
10380
10380     if (elts->parameter == NULL)
10381         elts->parameter = param;
10382     else
10383         (void) xmlAddSibling(elts->parameter, param);
10384 }
10386 /*
10387  * Process notification parameters for a service or instance
10388  */
10389 static void
10390 export_notify_params(scf_propertygroup_t *pg, struct entity_elts *elts)
10391 {
10392     xmlNodePtr n, event, *type;
10393     struct params_elts *eelts;
10394     int ret, err, i;
10396
10396     n = xmlNewNode(NULL, (xmlChar *)"notification_parameters");
10397     event = xmlNewNode(NULL, (xmlChar *)"event");
10398     if (n == NULL || event == NULL)
10399         uu_die(msg_create_xml);
10401
10401     /* event value */
10402     if (scf_pg_get_name(pg, exp_str, max_scf_name_len + 1) < 0)

```

```

10403         scfdie();
10404     safe_setprop(event, value_attr, exp_str);
10406
10406     (void) xmlAddChild(n, event);
10408
10408     if ((type = calloc(URI_SCHEME_NUM, sizeof(xmlNodePtr))) == NULL ||
10409         (eelts = calloc(URI_SCHEME_NUM,
10410             sizeof(struct params_elts))) == NULL)
10411         uu_die(gettext("Out of memory.\n"));
10413
10413     err = 0;
10415
10415     if (scf_iter_pg_properties(exp_prop_iter, pg) != SCF_SUCCESS)
10416         scfdie();
10418
10418     while ((ret = scf_iter_next_property(exp_prop_iter, exp_prop) == 1) {
10419         char *t, *p;
10421
10421         if (scf_property_get_name(exp_prop, exp_str, exp_str_sz) < 0)
10422             scfdie();
10424
10424         if ((t = strtok_r(exp_str, ",", &p)) == NULL || p == NULL) {
10425             /*
10426              * this is not a well formed notification parameters
10427              * element, we should export as regular pg
10428              */
10429             err = 1;
10430             break;
10431         }
10433
10433         if ((i = check_uri_protocol(t)) < 0) {
10434             err = 1;
10435             break;
10436         }
10438
10438         if (type[i] == NULL) {
10439             if ((type[i] = xmlNewNode(NULL, (xmlChar *)"type")) ==
10440                 NULL)
10441                 uu_die(msg_create_xml);
10443
10443                 safe_setprop(type[i], name_attr, t);
10444         }
10445         if (strcmp(p, active_attr) == 0) {
10446             if (set_attr_from_prop(exp_prop, type[i],
10447                 active_attr) != 0) {
10448                 err = 1;
10449                 break;
10450             }
10451             continue;
10452         }
10453         /*
10454          * We export the parameter
10455          */
10456         export_parameter(exp_prop, p, &eelts[i]);
10457     }
10459
10459     if (ret == -1)
10460         scfdie();
10462
10462     if (err == 1) {
10463         for (i = 0; i < URI_SCHEME_NUM; ++i)
10464             xmlFree(type[i]);
10465         free(type);
10467
10467         export_pg(pg, elts, SCE_ALL_VALUES);

```

```

10469         return;
10470     } else {
10471         for (i = 0; i < URI_SCHEME_NUM; ++i)
10472             if (type[i] != NULL) {
10473                 (void) xmlAddChildList(type[i],
10474                     elts[i].paramval);
10475                 (void) xmlAddChildList(type[i],
10476                     elts[i].parameter);
10477                 (void) xmlAddSibling(event, type[i]);
10478             }
10479     }
10480     free(type);

10482     if (elts->notify_params == NULL)
10483         elts->notify_params = n;
10484     else
10485         (void) xmlAddSibling(elts->notify_params, n);
10486 }

10488 /*
10489  * Process the general property group for an instance.
10490  */
10491 static void
10492 export_inst_general(scf_propertygroup_t *pg, xmlNodePtr inode,
10493     struct entity_elts *elts)
10494 {
10495     uint8_t enabled;
10496     struct pg_elts pgelts;
10497     int ret;

10499     /* enabled */
10500     if (pg_get_prop(pg, scf_property_enabled, exp_prop) == 0 &&
10501         prop_check_type(exp_prop, SCF_TYPE_BOOLEAN) == 0 &&
10502         prop_get_val(exp_prop, exp_val) == 0) {
10503         if (scf_value_get_boolean(exp_val, &enabled) != SCF_SUCCESS)
10504             scfdie();
10505     } else {
10506         enabled = 0;
10507     }

10509     safe_setprop(inode, enabled_attr, enabled ? true : false);

10511     if (scf_iter_pg_properties(exp_prop_iter, pg) != SCF_SUCCESS)
10512         scfdie();

10514     (void) memset(&pgelts, 0, sizeof(pgelts));

10516     while ((ret = scf_iter_next_property(exp_prop_iter, exp_prop)) == 1) {
10517         if (scf_property_get_name(exp_prop, exp_str, exp_str_sz) < 0)
10518             scfdie();

10520         if (strcmp(exp_str, scf_property_enabled) == 0) {
10521             continue;
10522         } else if (strcmp(exp_str, SCF_PROPERTY_RESTARTER) == 0) {
10523             xmlNodePtr rnode, sfnode;

10525             rnode = xmlNewNode(NULL, (xmlChar *)"restarter");
10526             if (rnode == NULL)
10527                 uu_die(msg_create_xml);

10529             sfnode = xmlNewChild(rnode, NULL,
10530                 (xmlChar *)"service_fmri", NULL);
10531             if (sfnode == NULL)
10532                 uu_die(msg_create_xml);

10534             if (set_attr_from_prop(exp_prop, sfnode,

```

```

10535         value_attr == 0) {
10536             elts->restarter = rnode;
10537             continue;
10538         }

10540         xmlFreeNode(rnode);
10541     }

10543     export_property(exp_prop, exp_str, &pgelts, SCE_ALL_VALUES);
10544 }
10545 if (ret == -1)
10546     scfdie();

10548     if (pgelts.propvals != NULL || pgelts.properties != NULL)
10549         export_pg_elts(&pgelts, scf_pg_general, scf_group_framework,
10550             elts);
10551 }

10553 /*
10554  * Put an instance element for the given instance into selts.
10555  */
10556 static void
10557 export_instance(scf_instance_t *inst, struct entity_elts *selts, int flags)
10558 {
10559     xmlNodePtr n;
10560     boolean_t isdefault;
10561     struct entity_elts elts;
10562     struct template_elts template_elts;
10563     int ret;

10565     n = xmlNewNode(NULL, (xmlChar *)"instance");
10566     if (n == NULL)
10567         uu_die(msg_create_xml);

10569     /* name */
10570     if (scf_instance_get_name(inst, exp_str, exp_str_sz) < 0)
10571         scfdie();
10572     safe_setprop(n, name_attr, exp_str);
10573     isdefault = strcmp(exp_str, "default") == 0;

10575     /* check existence of general pg (since general/enabled is required) */
10576     if (scf_instance_get_pg(inst, scf_pg_general, exp_pg) != SCF_SUCCESS) {
10577         if (scf_error() != SCF_ERROR_NOT_FOUND)
10578             scfdie();

10580         if (g_verbose) {
10581             if (scf_instance_to_fmri(inst, exp_str, exp_str_sz) < 0)
10582                 scfdie();

10584             warn(gettext("Instance %s has no general property "
10585                 "group; it will be marked disabled.\n"), exp_str);
10586         }

10588         safe_setprop(n, enabled_attr, false);
10589     } else if (scf_pg_get_type(exp_pg, exp_str, exp_str_sz) < 0 ||
10590         strcmp(exp_str, scf_group_framework) != 0) {
10591         if (g_verbose) {
10592             if (scf_pg_to_fmri(exp_pg, exp_str, exp_str_sz) < 0)
10593                 scfdie();

10595             warn(gettext("Property group %s is not of type "
10596                 "framework; the instance will be marked "
10597                 "disabled.\n"), exp_str);
10598         }

10600         safe_setprop(n, enabled_attr, false);

```

```

10601     }
10603     /* property groups */
10604     if (scf_iter_instanc_pgs(exp_pg_iter, inst) < 0)
10605         scfdie();
10607     (void) memset(&elts, 0, sizeof (elts));
10608     (void) memset(&template_elts, 0, sizeof (template_elts));
10610     while ((ret = scf_iter_next_pg(exp_pg_iter, exp_pg)) == 1) {
10611         uint32_t pgflags;
10613         if (scf_pg_get_flags(exp_pg, &pgflags) != 0)
10614             scfdie();
10616         if (pgflags & SCF_PG_FLAG_NONPERSISTENT)
10617             continue;
10619         if (scf_pg_get_type(exp_pg, exp_str, exp_str_sz) < 0)
10620             scfdie();
10622         if (strcmp(exp_str, SCF_GROUP_DEPENDENCY) == 0) {
10623             export_dependency(exp_pg, &elts);
10624             continue;
10625         } else if (strcmp(exp_str, SCF_GROUP_METHOD) == 0) {
10626             export_method(exp_pg, &elts);
10627             continue;
10628         } else if (strcmp(exp_str, scf_group_framework) == 0) {
10629             if (scf_pg_get_name(exp_pg, exp_str,
10630                 max_scf_name_len + 1) < 0)
10631                 scfdie();
10633             if (strcmp(exp_str, scf_pg_general) == 0) {
10634                 export_inst_general(exp_pg, n, &elts);
10635                 continue;
10636             } else if (strcmp(exp_str, SCF_PG_METHOD_CONTEXT) ==
10637                 0) {
10638                 export_method_context(exp_pg, &elts);
10639                 continue;
10640             } else if (strcmp(exp_str, SCF_PG_DEPENDENTS) == 0) {
10641                 export_dependents(exp_pg, &elts);
10642                 continue;
10643             }
10644         } else if (strcmp(exp_str, SCF_GROUP_TEMPLATE) == 0) {
10645             export_template(exp_pg, &elts, &template_elts);
10646             continue;
10647         } else if (strcmp(exp_str, SCF_NOTIFY_PARAMS_PG_TYPE) == 0) {
10648             export_notify_params(exp_pg, &elts);
10649             continue;
10650         }
10652         /* Ordinary pg. */
10653         export_pg(exp_pg, &elts, flags);
10654     }
10655     if (ret == -1)
10656         scfdie();
10658     if (template_elts.common_name != NULL) {
10659         elts.template = xmlNewNode(NULL, (xmlChar *)"template");
10660         (void) xmlAddChild(elts.template, template_elts.common_name);
10661         (void) xmlAddChild(elts.template, template_elts.description);
10662         (void) xmlAddChild(elts.template, template_elts.documentation);
10663     } else {
10664         xmlFreeNode(template_elts.description);
10665         xmlFreeNode(template_elts.documentation);
10666     }

```

```

10668         if (isdefault && elts.restarter == NULL &&
10669             elts.dependencies == NULL && elts.method_context == NULL &&
10670             elts.exec_methods == NULL && elts.notify_params == NULL &&
10671             elts.property_groups == NULL && elts.template == NULL) {
10672             xmlChar *eval;
10674             /* This is a default instance */
10675             eval = xmlGetProp(n, (xmlChar *)"enabled_attr");
10677             xmlFreeNode(n);
10679             n = xmlNewNode(NULL, (xmlChar *)"create_default_instance");
10680             if (n == NULL)
10681                 uu_die(msg_create_xml);
10683             safe_setprop(n, enabled_attr, (char *)eval);
10684             xmlFree(eval);
10686             selts->create_default_instance = n;
10687         } else {
10688             /* Assemble the children in order. */
10689             (void) xmlAddChild(n, elts.restarter);
10690             (void) xmlAddChildList(n, elts.dependencies);
10691             (void) xmlAddChildList(n, elts.dependents);
10692             (void) xmlAddChild(n, elts.method_context);
10693             (void) xmlAddChildList(n, elts.exec_methods);
10694             (void) xmlAddChildList(n, elts.notify_params);
10695             (void) xmlAddChildList(n, elts.property_groups);
10696             (void) xmlAddChild(n, elts.template);
10698             if (selts->instances == NULL)
10699                 selts->instances = n;
10700             else
10701                 (void) xmlAddSibling(selts->instances, n);
10702         }
10703     }
10705     /*
10706     * Return a service element for the given service.
10707     */
10708     static xmlNodePtr
10709     export_service(scf_service_t *svc, int flags)
10710     {
10711         xmlNodePtr snode;
10712         struct entity_elts elts;
10713         struct template_elts template_elts;
10714         int ret;
10716         snode = xmlNewNode(NULL, (xmlChar *)"service");
10717         if (snode == NULL)
10718             uu_die(msg_create_xml);
10720         /* Get & set name attribute */
10721         if (scf_service_get_name(svc, exp_str, max_scf_name_len + 1) < 0)
10722             scfdie();
10723         safe_setprop(snode, name_attr, exp_str);
10725         safe_setprop(snode, type_attr, "service");
10726         safe_setprop(snode, "version", "0");
10728         /* Acquire child elements. */
10729         if (scf_iter_service_pgs(exp_pg_iter, svc) != SCF_SUCCESS)
10730             scfdie();
10732         (void) memset(&elts, 0, sizeof (elts));

```

```

10733     (void) memset(&template_elts, 0, sizeof (template_elts));
10735
10735     while ((ret = scf_iter_next_pg(exp_pg_iter, exp_pg)) == 1) {
10736         uint32_t pgflags;
10738
10738         if (scf_pg_get_flags(exp_pg, &pgflags) != 0)
10739             scfdie();
10741
10741         if (pgflags & SCF_PG_FLAG_NONPERSISTENT)
10742             continue;
10744
10744         if (scf_pg_get_type(exp_pg, exp_str, exp_str_sz) < 0)
10745             scfdie();
10747
10747         if (strcmp(exp_str, SCF_GROUP_DEPENDENCY) == 0) {
10748             export_dependency(exp_pg, &elts);
10749             continue;
10750         } else if (strcmp(exp_str, SCF_GROUP_METHOD) == 0) {
10751             export_method(exp_pg, &elts);
10752             continue;
10753         } else if (strcmp(exp_str, scf_group_framework) == 0) {
10754             if (scf_pg_get_name(exp_pg, exp_str,
10755                 max_scf_name_len + 1) < 0)
10756                 scfdie();
10758
10758             if (strcmp(exp_str, scf_pg_general) == 0) {
10759                 export_svc_general(exp_pg, &elts);
10760                 continue;
10761             } else if (strcmp(exp_str, SCF_PG_METHOD_CONTEXT) ==
10762                 0) {
10763                 export_method_context(exp_pg, &elts);
10764                 continue;
10765             } else if (strcmp(exp_str, SCF_PG_DEPENDENTS) == 0) {
10766                 export_dependents(exp_pg, &elts);
10767                 continue;
10768             } else if (strcmp(exp_str, SCF_PG_MANIFESTFILES) == 0) {
10769                 continue;
10770             }
10771         } else if (strcmp(exp_str, SCF_GROUP_TEMPLATE) == 0) {
10772             export_template(exp_pg, &elts, &template_elts);
10773             continue;
10774         } else if (strcmp(exp_str, SCF_NOTIFY_PARAMS_PG_TYPE) == 0) {
10775             export_notify_params(exp_pg, &elts);
10776             continue;
10777         }
10779
10779         export_pg(exp_pg, &elts, flags);
10780     }
10781     if (ret == -1)
10782         scfdie();
10784
10784     if (template_elts.common_name != NULL) {
10785         elts.template = xmlNewNode(NULL, (xmlChar *)"template");
10786         (void) xmlAddChild(elts.template, template_elts.common_name);
10787         (void) xmlAddChild(elts.template, template_elts.description);
10788         (void) xmlAddChild(elts.template, template_elts.documentation);
10789     } else {
10790         xmlFreeNode(template_elts.description);
10791         xmlFreeNode(template_elts.documentation);
10792     }
10794
10794     /* Iterate instances */
10795     if (scf_iter_service_instances(exp_inst_iter, svc) != SCF_SUCCESS)
10796         scfdie();
10798
10798     while ((ret = scf_iter_next_instance(exp_inst_iter, exp_inst)) == 1)

```

```

10799         export_instance(exp_inst, &elts, flags);
10800     if (ret == -1)
10801         scfdie();
10803
10803     /* Now add all of the accumulated elements in order. */
10804     (void) xmlAddChild(snode, elts.create_default_instance);
10805     (void) xmlAddChild(snode, elts.single_instance);
10806     (void) xmlAddChild(snode, elts.restarter);
10807     (void) xmlAddChildList(snode, elts.dependencies);
10808     (void) xmlAddChildList(snode, elts.dependents);
10809     (void) xmlAddChild(snode, elts.method_context);
10810     (void) xmlAddChildList(snode, elts.exec_methods);
10811     (void) xmlAddChildList(snode, elts.notify_params);
10812     (void) xmlAddChildList(snode, elts.property_groups);
10813     (void) xmlAddChildList(snode, elts.instances);
10814     (void) xmlAddChild(snode, elts.stability);
10815     (void) xmlAddChild(snode, elts.template);
10817
10817     return (snode);
10818 }
10820
10820 static int
10821 export_callback(void *data, scf_walkinfo_t *wip)
10822 {
10823     FILE *f;
10824     xmlDocPtr doc;
10825     xmlNodePtr sb;
10826     int result;
10827     struct export_args *argsp = (struct export_args *)data;
10829
10829     if ((exp_inst = scf_instance_create(g_hndl)) == NULL ||
10830         (exp_pg = scf_pg_create(g_hndl)) == NULL ||
10831         (exp_prop = scf_property_create(g_hndl)) == NULL ||
10832         (exp_val = scf_value_create(g_hndl)) == NULL ||
10833         (exp_inst_iter = scf_iter_create(g_hndl)) == NULL ||
10834         (exp_pg_iter = scf_iter_create(g_hndl)) == NULL ||
10835         (exp_prop_iter = scf_iter_create(g_hndl)) == NULL ||
10836         (exp_val_iter = scf_iter_create(g_hndl)) == NULL)
10837         scfdie();
10839
10839     exp_str_sz = max_scf_name_len + 1;
10840     exp_str = safe_malloc(exp_str_sz);
10842
10842     if (argsp->filename != NULL) {
10843         errno = 0;
10844         f = fopen(argsp->filename, "wb");
10845         if (f == NULL) {
10846             if (errno == 0)
10847                 uu_die(gettext("Could not open \"%s\": no free "
10848                     "stdio streams.\n"), argsp->filename);
10849             else
10850                 uu_die(gettext("Could not open \"%s\"",
10851                     argsp->filename));
10852         }
10853     } else
10854         f = stdout;
10856
10856     doc = xmlNewDoc((xmlChar *)"1.0");
10857     if (doc == NULL)
10858         uu_die(gettext("Could not create XML document.\n"));
10860
10860     if (xmlCreateIntSubset(doc, (xmlChar *)"service_bundle", NULL,
10861         (xmlChar *)"MANIFEST_DTD_PATH") == NULL)
10862         uu_die(emacs_create_xml);
10864
10864     sb = xmlNewNode(NULL, (xmlChar *)"service_bundle");

```



```

10865     if (sb == NULL)
10866         uu_die(msg_create_xml);
10867     safe_setprop(sb, type_attr, "manifest");
10868     safe_setprop(sb, name_attr, "export");
10869     (void) xmlAddSibling(doc->children, sb);
10871     (void) xmlAddChild(sb, export_service(wip->svc, argsp->flags));
10873     result = write_service_bundle(doc, f);
10875     free(exp_str);
10876     scf_iter_destroy(exp_val_iter);
10877     scf_iter_destroy(exp_prop_iter);
10878     scf_iter_destroy(exp_pg_iter);
10879     scf_iter_destroy(exp_inst_iter);
10880     scf_value_destroy(exp_val);
10881     scf_property_destroy(exp_prop);
10882     scf_pg_destroy(exp_pg);
10883     scf_instance_destroy(exp_inst);
10885     xmlFreeDoc(doc);
10887     if (f != stdout)
10888         (void) fclose(f);
10890     return (result);
10891 }
10893 /*
10894  * Get the service named by fmri, build an XML tree which represents it, and
10895  * dump it into filename (or stdout if filename is NULL).
10896  */
10897 int
10898 lscf_service_export(char *fmri, const char *filename, int flags)
10899 {
10900     struct export_args args;
10901     char *fmridup;
10902     const char *scope, *svc, *inst;
10903     size_t cblen = 3 * max_scf_name_len;
10904     char *canonbuf = alloca(cblen);
10905     int ret, err;
10907     lscf_prep_hdl();
10909     bzero(&args, sizeof (args));
10910     args.filename = filename;
10911     args.flags = flags;
10913     /*
10914      * If some poor user has passed an exact instance FMRI, of the sort
10915      * one might cut and paste from svcs(1) or an error message, warn
10916      * and chop off the instance instead of failing.
10917      */
10918     fmridup = alloca(strlen(fmri) + 1);
10919     (void) strcpy(fmridup, fmri);
10920     if (strncmp(fmridup, SCF_FMRI_SVC_PREFIX,
10921         sizeof (SCF_FMRI_SVC_PREFIX) - 1) == 0 &&
10922         scf_parse_svc_fmri(fmridup, &scope, &svc, &inst, NULL, NULL) == 0 &&
10923         inst != NULL) {
10924         (void) strncpy(canonbuf, "svc:/", cblen);
10925         if (strcmp(scope, SCF_FMRI_LOCAL_SCOPE) != 0) {
10926             (void) strlcat(canonbuf, "/", cblen);
10927             (void) strlcat(canonbuf, scope, cblen);
10928         }
10929         (void) strlcat(canonbuf, svc, cblen);
10930         fmri = canonbuf;

```

```

10932         warn(gettext("Only services may be exported; ignoring "
10933             "instance portion of argument.\n"));
10934     }
10936     err = 0;
10937     if ((ret = scf_walk_fmri(g_hdl, 1, (char **)&fmri,
10938         SCF_WALK_SERVICE | SCF_WALK_NOINSTANCE, export_callback,
10939         &args, &err, semerr) != 0) {
10940         if (ret != -1)
10941             semerr(gettext("Failed to walk instances: %s\n"),
10942                 scf_strerror(ret));
10943         return (-1);
10944     }
10946     /*
10947      * Error message has already been printed.
10948      */
10949     if (err != 0)
10950         return (-1);
10952     return (0);
10953 }
10956 /*
10957  * Archive
10958  */
10960 static xmlNodePtr
10961 make_archive(int flags)
10962 {
10963     xmlNodePtr sb;
10964     scf_scope_t *scope;
10965     scf_service_t *svc;
10966     scf_iter_t *iter;
10967     int r;
10969     if ((scope = scf_scope_create(g_hdl)) == NULL ||
10970         (svc = scf_service_create(g_hdl)) == NULL ||
10971         (iter = scf_iter_create(g_hdl)) == NULL ||
10972         (exp_inst = scf_instance_create(g_hdl)) == NULL ||
10973         (exp_pg = scf_pg_create(g_hdl)) == NULL ||
10974         (exp_prop = scf_property_create(g_hdl)) == NULL ||
10975         (exp_val = scf_value_create(g_hdl)) == NULL ||
10976         (exp_inst_iter = scf_iter_create(g_hdl)) == NULL ||
10977         (exp_pg_iter = scf_iter_create(g_hdl)) == NULL ||
10978         (exp_prop_iter = scf_iter_create(g_hdl)) == NULL ||
10979         (exp_val_iter = scf_iter_create(g_hdl)) == NULL)
10980         scfdie();
10982     exp_str_sz = max_scf_len + 1;
10983     exp_str = safe_malloc(exp_str_sz);
10985     sb = xmlNewNode(NULL, (xmlChar *) "service_bundle");
10986     if (sb == NULL)
10987         uu_die(msg_create_xml);
10988     safe_setprop(sb, type_attr, "archive");
10989     safe_setprop(sb, name_attr, "none");
10991     if (scf_handle_get_scope(g_hdl, SCF_SCOPE_LOCAL, scope) != 0)
10992         scfdie();
10993     if (scf_iter_scope_services(iter, scope) != 0)
10994         scfdie();
10996     for (;;) {

```

```

10997         r = scf_iter_next_service(iter, svc);
10998         if (r == 0)
10999             break;
11000         if (r != 1)
11001             scfdie();
11003
11004         if (scf_service_get_name(svc, exp_str,
11005             max_scf_name_len + 1) < 0)
11006             scfdie();
11007
11008         if (strcmp(exp_str, SCF_LEGACY_SERVICE) == 0)
11009             continue;
11010
11011         (void) xmlAddChild(sb, export_service(svc, flags));
11012     }
11013
11014     free(exp_str);
11015
11016     scf_iter_destroy(exp_val_iter);
11017     scf_iter_destroy(exp_prop_iter);
11018     scf_iter_destroy(exp_pg_iter);
11019     scf_iter_destroy(exp_inst_iter);
11020     scf_value_destroy(exp_val);
11021     scf_property_destroy(exp_prop);
11022     scf_pg_destroy(exp_pg);
11023     scf_instance_destroy(exp_inst);
11024     scf_iter_destroy(iter);
11025     scf_service_destroy(svc);
11026     scf_scope_destroy(scope);
11027
11028     return (sb);
11029 }
11030
11031 int
11032 lscf_archive(const char *filename, int flags)
11033 {
11034     FILE *f;
11035     xmlDocPtr doc;
11036     int result;
11037
11038     lscf_prep_hdl();
11039
11040     if (filename != NULL) {
11041         errno = 0;
11042         f = fopen(filename, "wb");
11043         if (f == NULL) {
11044             if (errno == 0)
11045                 uu_die(gettext("Could not open \"%s\": no free "
11046                     "stdio streams.\n"), filename);
11047             else
11048                 uu_die(gettext("Could not open \"%s\"",
11049                     filename));
11050         }
11051     } else
11052         f = stdout;
11053
11054     doc = xmlNewDoc((xmlChar *)"1.0");
11055     if (doc == NULL)
11056         uu_die(gettext("Could not create XML document.\n"));
11057
11058     if (xmlCreateIntSubset(doc, (xmlChar *)"service_bundle", NULL,
11059         (xmlChar *)MANIFEST_DTD_PATH) == NULL)
11060         uu_die(msg_create_xml);
11061
11062     (void) xmlAddSibling(doc->children, make_archive(flags));

```

```

11063         result = write_service_bundle(doc, f);
11064
11065         xmlFreeDoc(doc);
11066
11067         if (f != stdout)
11068             (void) fclose(f);
11069
11070         return (result);
11071     }
11072
11073     /*
11074     * "Extract" a profile.
11075     */
11076     int
11077     lscf_profile_extract(const char *filename)
11078     {
11079         FILE *f;
11080         xmlDocPtr doc;
11081         xmlNodePtr sb, snode, inode;
11082         scf_scope_t *scope;
11083         scf_service_t *svc;
11084         scf_instance_t *inst;
11085         scf_propertygroup_t *pg;
11086         scf_property_t *prop;
11087         scf_value_t *val;
11088         scf_iter_t *siter, *iiter;
11089         int r, s;
11090         char *namebuf;
11091         uint8_t b;
11092         int result;
11093
11094         lscf_prep_hdl();
11095
11096         if (filename != NULL) {
11097             errno = 0;
11098             f = fopen(filename, "wb");
11099             if (f == NULL) {
11100                 if (errno == 0)
11101                     uu_die(gettext("Could not open \"%s\": no "
11102                         "free stdio streams.\n"), filename);
11103                 else
11104                     uu_die(gettext("Could not open \"%s\"",
11105                         filename));
11106             }
11107         } else
11108             f = stdout;
11109
11110         doc = xmlNewDoc((xmlChar *)"1.0");
11111         if (doc == NULL)
11112             uu_die(gettext("Could not create XML document.\n"));
11113
11114         if (xmlCreateIntSubset(doc, (xmlChar *)"service_bundle", NULL,
11115             (xmlChar *)MANIFEST_DTD_PATH) == NULL)
11116             uu_die(msg_create_xml);
11117
11118         sb = xmlNewNode(NULL, (xmlChar *)"service_bundle");
11119         if (sb == NULL)
11120             uu_die(msg_create_xml);
11121         safe_setprop(sb, type_attr, "profile");
11122         safe_setprop(sb, name_attr, "extract");
11123         (void) xmlAddSibling(doc->children, sb);
11124
11125         if ((scope = scf_scope_create(g_hdl)) == NULL ||
11126             (svc = scf_service_create(g_hdl)) == NULL ||
11127             (inst = scf_instance_create(g_hdl)) == NULL ||

```

```

11129         (pg = scf_pg_create(g_hdl)) == NULL ||
11130         (prop = scf_property_create(g_hdl)) == NULL ||
11131         (val = scf_value_create(g_hdl)) == NULL ||
11132         (siter = scf_iter_create(g_hdl)) == NULL ||
11133         (iiter = scf_iter_create(g_hdl)) == NULL)
11134         scfdie();

11136     if (scf_handle_get_local_scope(g_hdl, scope) != SCF_SUCCESS)
11137         scfdie();

11139     if (scf_iter_scope_services(siter, scope) != SCF_SUCCESS)
11140         scfdie();

11142     namebuf = safe_malloc(max_scf_name_len + 1);

11144     while ((r = scf_iter_next_service(siter, svc)) == 1) {
11145         if (scf_iter_service_instances(iiter, svc) != SCF_SUCCESS)
11146             scfdie();

11148         snode = xmlNewNode(NULL, (xmlChar *)"service");
11149         if (snode == NULL)
11150             uu_die(msg_create_xml);

11152         if (scf_service_get_name(svc, namebuf, max_scf_name_len + 1) <
11153             0)
11154             scfdie();

11156         safe_setprop(snode, name_attr, namebuf);

11158         safe_setprop(snode, type_attr, "service");
11159         safe_setprop(snode, "version", "0");

11161         while ((s = scf_iter_next_instance(iiter, inst)) == 1) {
11162             if (scf_instance_get_pg(inst, scf_pg_general, pg) !=
11163                 SCF_SUCCESS) {
11164                 if (scf_error() != SCF_ERROR_NOT_FOUND)
11165                     scfdie();

11167                 if (g_verbose) {
11168                     ssize_t len;
11169                     char *fmri;

11171                     len =
11172                         scf_instance_to_fmri(inst, NULL, 0);
11173                     if (len < 0)
11174                         scfdie();

11176                     fmri = safe_malloc(len + 1);

11178                     if (scf_instance_to_fmri(inst, fmri,
11179                         len + 1) < 0)
11180                         scfdie();

11182                     warn("Instance %s has no \"%s\" "
11183                         "property group.\n", fmri,
11184                         scf_pg_general);

11186                     free(fmri);
11187                 }

11189                 continue;
11190             }

11192             if (pg_get_prop(pg, scf_property_enabled, prop) != 0 ||
11193                 prop_check_type(prop, SCF_TYPE_BOOLEAN) != 0 ||
11194                 prop_get_val(prop, val) != 0)

```

```

11195                 continue;

11197                 inode = xmlNewChild(snode, NULL, (xmlChar *)"instance",
11198                     NULL);
11199                 if (inode == NULL)
11200                     uu_die(msg_create_xml);

11202                 if (scf_instance_get_name(inst, namebuf,
11203                     max_scf_name_len + 1) < 0)
11204                     scfdie();

11206                 safe_setprop(inode, name_attr, namebuf);

11208                 if (scf_value_get_boolean(val, &b) != SCF_SUCCESS)
11209                     scfdie();

11211                 safe_setprop(inode, enabled_attr, b ? true : false);
11212             }
11213             if (s < 0)
11214                 scfdie();

11216             if (snode->children != NULL)
11217                 (void) xmlAddChild(sb, snode);
11218             else
11219                 xmlFreeNode(snode);
11220         }
11221         if (r < 0)
11222             scfdie();

11224         free(namebuf);

11226         result = write_service_bundle(doc, f);

11228         xmlFreeDoc(doc);

11230         if (f != stdout)
11231             (void) fclose(f);

11233         return (result);
11234     }

11237 /*
11238  * Entity manipulation commands
11239  */

11241 /*
11242  * Entity selection. If no entity is selected, then the current scope is in
11243  * cur_scope, and cur_svc and cur_inst are NULL. When a service is selected,
11244  * only cur_inst is NULL, and when an instance is selected, none are NULL.
11245  * When the snaplevel of a snapshot is selected, cur_level, cur_snap, and
11246  * cur_inst will be non-NULL.
11247  */

11249 /* Returns 1 if maybe absolute fmri, 0 on success (dies on failure) */
11250 static int
11251 select_inst(const char *name)
11252 {
11253     scf_instance_t *inst;
11254     scf_error_t err;

11256     assert(cur_svc != NULL);

11258     inst = scf_instance_create(g_hdl);
11259     if (inst == NULL)
11260         scfdie();

```

```

11262     if (scf_service_get_instance(cur_svc, name, inst) == SCF_SUCCESS) {
11263         cur_inst = inst;
11264         return (0);
11265     }
11267     err = scf_error();
11268     if (err != SCF_ERROR_NOT_FOUND && err != SCF_ERROR_INVALID_ARGUMENT)
11269         scfdie();
11271     scf_instance_destroy(inst);
11272     return (1);
11273 }
11275 /* Returns as above. */
11276 static int
11277 select_svc(const char *name)
11278 {
11279     scf_service_t *svc;
11280     scf_error_t err;
11282     assert(cur_scope != NULL);
11284     svc = scf_service_create(g_hndl);
11285     if (svc == NULL)
11286         scfdie();
11288     if (scf_scope_get_service(cur_scope, name, svc) == SCF_SUCCESS) {
11289         cur_svc = svc;
11290         return (0);
11291     }
11293     err = scf_error();
11294     if (err != SCF_ERROR_NOT_FOUND && err != SCF_ERROR_INVALID_ARGUMENT)
11295         scfdie();
11297     scf_service_destroy(svc);
11298     return (1);
11299 }
11301 /* ARGSUSED */
11302 static int
11303 select_callback(void *unused, scf_walkinfo_t *wip)
11304 {
11305     scf_instance_t *inst;
11306     scf_service_t *svc;
11307     scf_scope_t *scope;
11309     if (wip->inst != NULL) {
11310         if ((scope = scf_scope_create(g_hndl)) == NULL ||
11311             (svc = scf_service_create(g_hndl)) == NULL ||
11312             (inst = scf_instance_create(g_hndl)) == NULL)
11313             scfdie();
11315         if (scf_handle_decode_fmri(g_hndl, wip->fmri, scope, svc,
11316             inst, NULL, NULL, SCF_DECODE_FMRI_EXACT) != SCF_SUCCESS)
11317             scfdie();
11318     } else {
11319         assert(wip->svc != NULL);
11321         if ((scope = scf_scope_create(g_hndl)) == NULL ||
11322             (svc = scf_service_create(g_hndl)) == NULL)
11323             scfdie();
11325         if (scf_handle_decode_fmri(g_hndl, wip->fmri, scope, svc,
11326             NULL, NULL, NULL, SCF_DECODE_FMRI_EXACT) != SCF_SUCCESS)

```

```

11327         scfdie();
11329         inst = NULL;
11330     }
11332     /* Clear out the current selection */
11333     assert(cur_scope != NULL);
11334     scf_scope_destroy(cur_scope);
11335     scf_service_destroy(cur_svc);
11336     scf_instance_destroy(cur_inst);
11338     cur_scope = scope;
11339     cur_svc = svc;
11340     cur_inst = inst;
11342     return (0);
11343 }
11345 static int
11346 validate_callback(void *fmri_p, scf_walkinfo_t *wip)
11347 {
11348     char **fmri = fmri_p;
11350     *fmri = strdup(wip->fmri);
11351     if (*fmri == NULL)
11352         uu_die(gettext("Out of memory.\n"));
11354     return (0);
11355 }
11357 /*
11358  * validate [fmri]
11359  * Perform the validation of an FMRI instance.
11360  */
11361 void
11362 lscf_validate_fmri(const char *fmri)
11363 {
11364     int ret = 0;
11365     size_t inst_sz;
11366     char *inst_fmri = NULL;
11367     scf_tmpl_errors_t *errs = NULL;
11368     char *snapbuf = NULL;
11370     lscf_prep_hndl();
11372     if (fmri == NULL) {
11373         inst_sz = max_scf_fmri_len + 1;
11374         inst_fmri = safe_malloc(inst_sz);
11376         if (cur_snap != NULL) {
11377             snapbuf = safe_malloc(max_scf_name_len + 1);
11378             if (scf_snapshot_get_name(cur_snap, snapbuf,
11379                 max_scf_name_len + 1) < 0)
11380                 scfdie();
11381         }
11382         if (cur_inst == NULL) {
11383             semerr(gettext("No instance selected\n"));
11384             goto cleanup;
11385         } else if (scf_instance_to_fmri(cur_inst, inst_fmri,
11386             inst_sz) >= inst_sz) {
11387             /* sanity check. Should never get here */
11388             uu_die(gettext("Unexpected error! file %s, line %d\n"),
11389                 __FILE__, __LINE__);
11390         }
11391     } else {
11392         scf_error_t scf_err;

```

```

11393         int err = 0;
11395
11395         if ((scf_err = scf_walk_fmri(g_hndl, 1, (char **)&fmri, 0,
11396             validate_callback, &inst_fmri, &err, semerr)) != 0) {
11397             uu_warn("Failed to walk instances: %s\n",
11398                 scf_strerror(scf_err));
11399             goto cleanup;
11400         }
11401         if (err != 0) {
11402             /* error message displayed by scf_walk_fmri */
11403             goto cleanup;
11404         }
11405     }
11407
11407     ret = scf_tmpl_validate_fmri(g_hndl, inst_fmri, snapbuf, &errs,
11408         SCF_TMPL_VALIDATE_FLAG_CURRENT);
11409     if (ret == -1) {
11410         if (scf_error() == SCF_ERROR_TEMPLATE_INVALID) {
11411             warn(gettext("Template data for %s is invalid. "
11412                 "Considering reverting to a previous snapshot or "
11413                 "restoring original configuration.\n"), inst_fmri);
11414         } else {
11415             uu_warn("%s: %s\n",
11416                 gettext("Error validating the instance"),
11417                 scf_strerror(scf_error()));
11418         }
11419     } else if (ret == 1 && errs != NULL) {
11420         scf_tmpl_error_t *err = NULL;
11421         char *msg;
11422         size_t len = 256; /* initial error buffer size */
11423         int flag = (est->sc_cmd_flags & SC_CMD_IACTIVE) ?
11424             SCF_TMPL_STRERROR_HUMAN : 0;
11426
11426         msg = safe_malloc(len);
11428
11428         while ((err = scf_tmpl_next_error(errs)) != NULL) {
11429             int ret;
11431
11431             if ((ret = scf_tmpl_strerror(err, msg, len,
11432                 flag)) >= len) {
11433                 len = ret + 1;
11434                 msg = realloc(msg, len);
11435                 if (msg == NULL)
11436                     uu_die(gettext(
11437                         "Out of memory.\n"));
11438                 (void) scf_tmpl_strerror(err, msg, len,
11439                     flag);
11440             }
11441             (void) fprintf(stderr, "%s\n", msg);
11442         }
11443         if (msg != NULL)
11444             free(msg);
11445     }
11446     if (errs != NULL)
11447         scf_tmpl_errors_destroy(errs);
11449 cleanup:
11450     free(inst_fmri);
11451     free(snapbuf);
11452 }
11454 static void
11455 lscf_validate_file(const char *filename)
11456 {
11457     tmpl_errors_t *errs;

```

```

11459     bundle_t *b = internal_bundle_new();
11460     if (lxml_get_bundle_file(b, filename, SVCCFG_OP_IMPORT) == 0) {
11461         if (tmpl_validate_bundle(b, &errs) != TVS_SUCCESS) {
11462             tmpl_errors_print(stderr, errs, "");
11463             semerr(gettext("Validation failed.\n"));
11464         }
11465         tmpl_errors_destroy(errs);
11466     }
11467     (void) internal_bundle_free(b);
11468 }
11470 /*
11471  * validate [fmri|file]
11472  */
11473 void
11474 lscf_validate(const char *arg)
11475 {
11476     const char *str;
11478
11478     if (strncmp(arg, SCF_FMRI_FILE_PREFIX,
11479         sizeof(SCF_FMRI_FILE_PREFIX) - 1) == 0) {
11480         str = arg + sizeof(SCF_FMRI_FILE_PREFIX) - 1;
11481         lscf_validate_file(str);
11482     } else if (strncmp(arg, SCF_FMRI_SVC_PREFIX,
11483         sizeof(SCF_FMRI_SVC_PREFIX) - 1) == 0) {
11484         str = arg + sizeof(SCF_FMRI_SVC_PREFIX) - 1;
11485         lscf_validate_fmri(str);
11486     } else if (access(arg, R_OK | F_OK) == 0) {
11487         lscf_validate_file(arg);
11488     } else {
11489         lscf_validate_fmri(arg);
11490     }
11491 }
11493 void
11494 lscf_select(const char *fmri)
11495 {
11496     int ret, err;
11498
11498     lscf_prep_hndl();
11500
11500     if (cur_snap != NULL) {
11501         struct snaplevel *elt;
11502         char *buf;
11504
11504         /* Error unless name is that of the next level. */
11505         elt = uu_list_next(cur_levels, cur_elt);
11506         if (elt == NULL) {
11507             semerr(gettext("No children.\n"));
11508             return;
11509         }
11511
11511         buf = safe_malloc(max_scf_name_len + 1);
11513
11513         if (scf_snaplevel_get_instance_name(elt->sl, buf,
11514             max_scf_name_len + 1) < 0)
11515             scfdie();
11517
11517         if (strcmp(buf, fmri) != 0) {
11518             semerr(gettext("No such child.\n"));
11519             free(buf);
11520             return;
11521         }
11523
11523         free(buf);

```

```

11525         cur_elt = elt;
11526         cur_level = elt->sl;
11527         return;
11528     }

11530     /*
11531     * Special case for 'svc:', which takes the user to the scope level.
11532     */
11533     if (strcmp(fmri, "svc:") == 0) {
11534         scf_instance_destroy(cur_inst);
11535         scf_service_destroy(cur_svc);
11536         cur_inst = NULL;
11537         cur_svc = NULL;
11538         return;
11539     }

11541     /*
11542     * Special case for ':properties'. This appears as part of 'list' but
11543     * can't be selected. Give a more helpful error message in this case.
11544     */
11545     if (strcmp(fmri, ":properties") == 0) {
11546         semerr(gettext(":properties is not an entity. Try 'listprop' "
11547             "to list properties.\n"));
11548         return;
11549     }

11551     /*
11552     * First try the argument as relative to the current selection.
11553     */
11554     if (cur_inst != NULL) {
11555         /* EMPTY */;
11556     } else if (cur_svc != NULL) {
11557         if (select_inst(fmri) != 1)
11558             return;
11559     } else {
11560         if (select_svc(fmri) != 1)
11561             return;
11562     }

11564     err = 0;
11565     if ((ret = scf_walk_fmri(g_hdl, 1, (char **)&fmri, SCF_WALK_SERVICE,
11566         select_callback, NULL, &err, semerr)) != 0) {
11567         semerr(gettext("Failed to walk instances: %s\n"),
11568             scf_strerror(ret));
11569     }
11570 }

11572 void
11573 lscf_unselect(void)
11574 {
11575     lscf_prep_hdl();

11577     if (cur_snap != NULL) {
11578         struct snaplevel *elt;

11580         elt = uu_list_prev(cur_levels, cur_elt);
11581         if (elt == NULL) {
11582             semerr(gettext("No parent levels.\n"));
11583         } else {
11584             cur_elt = elt;
11585             cur_level = elt->sl;
11586         }
11587     } else if (cur_inst != NULL) {
11588         scf_instance_destroy(cur_inst);
11589         cur_inst = NULL;
11590     } else if (cur_svc != NULL) {

```

```

11591         scf_service_destroy(cur_svc);
11592         cur_svc = NULL;
11593     } else {
11594         semerr(gettext("Cannot unselect at scope level.\n"));
11595     }
11596 }

11598 /*
11599 * Return the FMRI of the current selection, for the prompt.
11600 */
11601 void
11602 lscf_get_selection_str(char *buf, size_t bufsz)
11603 {
11604     char *cp;
11605     ssize_t fmrlen, szret;
11606     boolean_t deleted = B_FALSE;

11608     if (g_hdl == NULL) {
11609         (void) strcpy(buf, "svc:", bufsz);
11610         return;
11611     }

11613     if (cur_level != NULL) {
11614         assert(cur_snap != NULL);

11616         /* [ snapshot ] FMRI [ : instance ] */
11617         assert(bufsz >= 1 + max_scf_name_len + 1 + max_scf_fmri_len
11618             + 2 + max_scf_name_len + 1 + 1);

11620         buf[0] = '[';

11622         szret = scf_snapshot_get_name(cur_snap, buf + 1,
11623             max_scf_name_len + 1);
11624         if (szret < 0) {
11625             if (scf_error() != SCF_ERROR_DELETED)
11626                 scfdie();

11628             goto snap_deleted;
11629         }

11631         (void) strcat(buf, "]svc:/");

11633         cp = strchr(buf, '\0');

11635         szret = scf_snaplevel_get_service_name(cur_level, cp,
11636             max_scf_name_len + 1);
11637         if (szret < 0) {
11638             if (scf_error() != SCF_ERROR_DELETED)
11639                 scfdie();

11641             goto snap_deleted;
11642         }

11644         cp = strchr(cp, '\0');

11646         if (snaplevel_is_instance(cur_level)) {
11647             *cp++ = ':';

11649             if (scf_snaplevel_get_instance_name(cur_level, cp,
11650                 max_scf_name_len + 1) < 0) {
11651                 if (scf_error() != SCF_ERROR_DELETED)
11652                     scfdie();

11654                 goto snap_deleted;
11655             }
11656         } else {

```

```

11657         *cp++ = '[';
11658         *cp++ = ':';

11660         if (scf_instance_get_name(cur_inst, cp,
11661             max_scf_name_len + 1) < 0) {
11662             if (scf_error() != SCF_ERROR_DELETED)
11663                 scfdie();

11665             goto snap_deleted;
11666         }

11668         (void) strcat(buf, "];");
11669     }

11671     return;

11673 snap_deleted:
11674     deleted = B_TRUE;
11675     free(buf);
11676     unselect_cursnap();
11677 }

11679 assert(cur_snap == NULL);

11681 if (cur_inst != NULL) {
11682     assert(cur_svc != NULL);
11683     assert(cur_scope != NULL);

11685     fmrlen = scf_instance_to_fmri(cur_inst, buf, bufsz);
11686     if (fmrlen >= 0) {
11687         assert(fmrlen < bufsz);
11688         if (deleted)
11689             warn(msg_deleted);
11690         return;
11691     }

11693     if (scf_error() != SCF_ERROR_DELETED)
11694         scfdie();

11696     deleted = B_TRUE;
11698     scf_instance_destroy(cur_inst);
11699     cur_inst = NULL;
11700 }

11702 if (cur_svc != NULL) {
11703     assert(cur_scope != NULL);

11705     szret = scf_service_to_fmri(cur_svc, buf, bufsz);
11706     if (szret >= 0) {
11707         assert(szret < bufsz);
11708         if (deleted)
11709             warn(msg_deleted);
11710         return;
11711     }

11713     if (scf_error() != SCF_ERROR_DELETED)
11714         scfdie();

11716     deleted = B_TRUE;
11717     scf_service_destroy(cur_svc);
11718     cur_svc = NULL;
11719 }

11721 assert(cur_scope != NULL);
11722 fmrlen = scf_scope_to_fmri(cur_scope, buf, bufsz);

```

```

11724     if (fmrlen < 0)
11725         scfdie();

11727     assert(fmrlen < bufsz);
11728     if (deleted)
11729         warn(msg_deleted);
11730 }

11732 /*
11733  * Entity listing. Entities and colon namespaces (e.g., :properties and
11734  * :statistics) are listed for the current selection.
11735  */
11736 void
11737 lscf_list(const char *pattern)
11738 {
11739     scf_iter_t *iter;
11740     char *buf;
11741     int ret;

11743     lscf_prep_hdl();

11745     if (cur_level != NULL) {
11746         struct snaplevel *elt;

11748         (void) fputs(COLON_NAMESPACES, stdout);

11750         elt = uu_list_next(cur_levels, cur_elt);
11751         if (elt == NULL)
11752             return;

11754         /*
11755          * For now, we know that the next level is an instance. But
11756          * if we ever have multiple scopes, this could be complicated.
11757          */
11758         buf = safe_malloc(max_scf_name_len + 1);
11759         if (scf_snaplevel_get_instance_name(elt->sl, buf,
11760             max_scf_name_len + 1) >= 0) {
11761             (void) puts(buf);
11762         } else {
11763             if (scf_error() != SCF_ERROR_DELETED)
11764                 scfdie();
11765         }

11767         free(buf);

11769         return;
11770     }

11772     if (cur_inst != NULL) {
11773         (void) fputs(COLON_NAMESPACES, stdout);
11774         return;
11775     }

11777     iter = scf_iter_create(g_hdl);
11778     if (iter == NULL)
11779         scfdie();

11781     buf = safe_malloc(max_scf_name_len + 1);

11783     if (cur_svc != NULL) {
11784         /* List the instances in this service. */
11785         scf_instance_t *inst;

11787         inst = scf_instance_create(g_hdl);
11788         if (inst == NULL)

```

```

11789         scfdie();
11791     if (scf_iter_service_instances(iter, cur_svc) == 0) {
11792         safe_printf(COLON_NAMESPACES);
11794         for (;;) {
11795             ret = scf_iter_next_instance(iter, inst);
11796             if (ret == 0)
11797                 break;
11798             if (ret != 1) {
11799                 if (scf_error() != SCF_ERROR_DELETED)
11800                     scfdie();
11802                 break;
11803             }
11805             if (scf_instance_get_name(inst, buf,
11806                 max_scf_name_len + 1) >= 0) {
11807                 if (pattern == NULL ||
11808                     fnmatch(pattern, buf, 0) == 0)
11809                     (void) puts(buf);
11810             } else {
11811                 if (scf_error() != SCF_ERROR_DELETED)
11812                     scfdie();
11813             }
11814         } else {
11815             if (scf_error() != SCF_ERROR_DELETED)
11816                 scfdie();
11817         }
11818     }
11820     scf_instance_destroy(inst);
11821 } else {
11822     /* List the services in this scope. */
11823     scf_service_t *svc;
11825     assert(cur_scope != NULL);
11827     svc = scf_service_create(g_hndl);
11828     if (svc == NULL)
11829         scfdie();
11831     if (scf_iter_scope_services(iter, cur_scope) != SCF_SUCCESS)
11832         scfdie();
11834     for (;;) {
11835         ret = scf_iter_next_service(iter, svc);
11836         if (ret == 0)
11837             break;
11838         if (ret != 1)
11839             scfdie();
11841         if (scf_service_get_name(svc, buf,
11842             max_scf_name_len + 1) >= 0) {
11843             if (pattern == NULL ||
11844                 fnmatch(pattern, buf, 0) == 0)
11845                 safe_printf("%s\n", buf);
11846         } else {
11847             if (scf_error() != SCF_ERROR_DELETED)
11848                 scfdie();
11849         }
11850     }
11852     scf_service_destroy(svc);
11853 }

```

```

11855     free(buf);
11856     scf_iter_destroy(iter);
11857 }
11859 /*
11860  * Entity addition. Creates an empty entity in the current selection.
11861  */
11862 void
11863 lscf_add(const char *name)
11864 {
11865     lscf_prep_hndl();
11867     if (cur_snap != NULL) {
11868         semerr(msg_cant_modify_snapshots);
11869     } else if (cur_inst != NULL) {
11870         semerr(gettext("Cannot add entities to an instance.\n"));
11871     } else if (cur_svc != NULL) {
11873         if (scf_service_add_instance(cur_svc, name, NULL) !=
11874             SCF_SUCCESS) {
11875             switch (scf_error()) {
11876                 case SCF_ERROR_INVALID_ARGUMENT:
11877                     semerr(gettext("Invalid name.\n"));
11878                     break;
11880                 case SCF_ERROR_EXISTS:
11881                     semerr(gettext("Instance already exists.\n"));
11882                     break;
11884                 case SCF_ERROR_PERMISSION_DENIED:
11885                     semerr(msg_permission_denied);
11886                     break;
11888                 default:
11889                     scfdie();
11890             }
11891         } else {
11892             assert(cur_scope != NULL);
11893         }
11895         if (scf_scope_add_service(cur_scope, name, NULL) !=
11896             SCF_SUCCESS) {
11897             switch (scf_error()) {
11898                 case SCF_ERROR_INVALID_ARGUMENT:
11899                     semerr(gettext("Invalid name.\n"));
11900                     break;
11902                 case SCF_ERROR_EXISTS:
11903                     semerr(gettext("Service already exists.\n"));
11904                     break;
11906                 case SCF_ERROR_PERMISSION_DENIED:
11907                     semerr(msg_permission_denied);
11908                     break;
11910                 case SCF_ERROR_BACKEND_READONLY:
11911                     semerr(msg_read_only);
11912                     break;
11914                 default:
11915                     scfdie();
11916             }
11917         }
11918     }
11919 }

```



```

11921 /* return 1 if the entity has no persistent pgs, else return 0 */
11922 static int
11923 entity_has_no_pgs(void *ent, int isservice)
11924 {
11925     scf_iter_t *iter = NULL;
11926     scf_propertygroup_t *pg = NULL;
11927     uint32_t flags;
11928     int err;
11929     int ret = 1;
11931
11932     if ((iter = scf_iter_create(g_hndl)) == NULL ||
11933         (pg = scf_pg_create(g_hndl)) == NULL)
11934         scfdie();
11935
11936     if (isservice) {
11937         if (scf_iter_service_pgs(iter, (scf_service_t *)ent) < 0)
11938             scfdie();
11939     } else {
11940         if (scf_iter_instance_pgs(iter, (scf_instance_t *)ent) < 0)
11941             scfdie();
11942     }
11943
11944     while ((err = scf_iter_next_pg(iter, pg)) == 1) {
11945         if (scf_pg_get_flags(pg, &flags) != 0)
11946             scfdie();
11947
11948         /* skip nonpersistent pgs */
11949         if (flags & SCF_PG_FLAG_NONPERSISTENT)
11950             continue;
11951
11952         ret = 0;
11953         break;
11954     }
11955
11956     if (err == -1)
11957         scfdie();
11958
11959     scf_pg_destroy(pg);
11960     scf_iter_destroy(iter);
11961
11962     return (ret);
11963 }
11964
11965 /* return 1 if the service has no instances, else return 0 */
11966 static int
11967 svc_has_no_insts(scf_service_t *svc)
11968 {
11969     scf_instance_t *inst;
11970     scf_iter_t *iter;
11971     int r;
11972     int ret = 1;
11973
11974     if ((inst = scf_instance_create(g_hndl)) == NULL ||
11975         (iter = scf_iter_create(g_hndl)) == NULL)
11976         scfdie();
11977
11978     if (scf_iter_service_instances(iter, svc) != 0)
11979         scfdie();
11980
11981     r = scf_iter_next_instance(iter, inst);
11982     if (r == 1) {
11983         ret = 0;
11984     } else if (r == 0) {
11985         ret = 1;
11986     } else if (r == -1) {
11987         scfdie();
11988     }

```

```

11987     } else {
11988         bad_error("scf_iter_next_instance", r);
11989     }
11990
11991     scf_iter_destroy(iter);
11992     scf_instance_destroy(inst);
11993
11994     return (ret);
11995 }
11996
11997 /*
11998  * Entity deletion.
11999  */
12000
12001 /*
12002  * Delete the property group <fmri>:/properties/<name>. Returns
12003  * SCF_ERROR_NONE on success (or if the entity is not found),
12004  * SCF_ERROR_INVALID_ARGUMENT if the fmri is bad, SCF_ERROR_TYPE_MISMATCH if
12005  * the pg is the wrong type, or SCF_ERROR_PERMISSION_DENIED if permission was
12006  * denied.
12007  */
12008 static scf_error_t
12009 delete_dependency_pg(const char *fmri, const char *name)
12010 {
12011     void *entity = NULL;
12012     int isservice;
12013     scf_propertygroup_t *pg = NULL;
12014     scf_error_t result;
12015     char *pgty;
12016     scf_service_t *svc = NULL;
12017     scf_instance_t *inst = NULL;
12018     scf_iter_t *iter = NULL;
12019     char *name_buf = NULL;
12020
12021     result = fmri_to_entity(g_hndl, fmri, &entity, &isservice);
12022     switch (result) {
12023     case SCF_ERROR_NONE:
12024         break;
12025
12026     case SCF_ERROR_NO_MEMORY:
12027         uu_die(gettext("Out of memory.\n"));
12028         /* NOTREACHED */
12029
12030     case SCF_ERROR_INVALID_ARGUMENT:
12031     case SCF_ERROR_CONSTRAINT_VIOLATED:
12032         return (SCF_ERROR_INVALID_ARGUMENT);
12033
12034     case SCF_ERROR_NOT_FOUND:
12035         result = SCF_ERROR_NONE;
12036         goto out;
12037
12038     default:
12039         bad_error("fmri_to_entity", result);
12040     }
12041
12042     pg = scf_pg_create(g_hndl);
12043     if (pg == NULL)
12044         scfdie();
12045
12046     if (entity_get_pg(entity, isservice, name, pg) != 0) {
12047         if (scf_error() != SCF_ERROR_NOT_FOUND)
12048             scfdie();
12049
12050         result = SCF_ERROR_NONE;
12051         goto out;
12052     }

```

```

12054     pgty = safe_malloc(max_scf_pg_type_len + 1);
12056     if (scf_pg_get_type(pg, pgty, max_scf_pg_type_len + 1) < 0)
12057         scfdie();

12059     if (strcmp(pgty, SCF_GROUP_DEPENDENCY) != 0) {
12060         result = SCF_ERROR_TYPE_MISMATCH;
12061         free(pgty);
12062         goto out;
12063     }

12065     free(pgty);

12067     if (scf_pg_delete(pg) != 0) {
12068         result = scf_error();
12069         if (result != SCF_ERROR_PERMISSION_DENIED)
12070             scfdie();
12071         goto out;
12072     }

12074     /*
12075     * We have to handle the case where we've just deleted the last
12076     * property group of a "dummy" entity (instance or service).
12077     * A "dummy" entity is an entity only present to hold an
12078     * external dependency.
12079     * So, in the case we deleted the last property group then we
12080     * can also delete the entity. If the entity is an instance then
12081     * we must verify if this was the last instance for the service
12082     * and if it is, we can also delete the service if it doesn't
12083     * have any property group either.
12084     */

12086     result = SCF_ERROR_NONE;

12088     if (isservice) {
12089         svc = (scf_service_t *)entity;

12091         if ((inst = scf_instance_create(g_hndl)) == NULL ||
12092             (iter = scf_iter_create(g_hndl)) == NULL)
12093             scfdie();

12095         name_buf = safe_malloc(max_scf_name_len + 1);
12096     } else {
12097         inst = (scf_instance_t *)entity;
12098     }

12100     /*
12101     * If the entity is an instance and we've just deleted its last
12102     * property group then we should delete it.
12103     */
12104     if (!isservice && entity_has_no_pgs(entity, isservice)) {
12105         /* find the service before deleting the inst. - needed later */
12106         if ((svc = scf_service_create(g_hndl)) == NULL)
12107             scfdie();

12109         if (scf_instance_get_parent(inst, svc) != 0)
12110             scfdie();

12112         /* delete the instance */
12113         if (scf_instance_delete(inst) != 0) {
12114             if (scf_error() != SCF_ERROR_PERMISSION_DENIED)
12115                 scfdie();

12117             result = SCF_ERROR_PERMISSION_DENIED;
12118             goto out;

```

```

12119     }
12120     /* no need to refresh the instance */
12121     inst = NULL;
12122 }

12124     /*
12125     * If the service has no more instances and pgs or we just deleted the
12126     * last instance and the service doesn't have anymore property groups
12127     * then the service should be deleted.
12128     */
12129     if (svc != NULL &&
12130         svc_has_no_insts(svc) &&
12131         entity_has_no_pgs((void *)svc, 1)) {
12132         if (scf_service_delete(svc) == 0) {
12133             if (isservice) {
12134                 /* no need to refresh the service */
12135                 svc = NULL;
12136             }

12138             goto out;
12139         }

12141         if (scf_error() != SCF_ERROR_PERMISSION_DENIED)
12142             scfdie();

12144         result = SCF_ERROR_PERMISSION_DENIED;
12145     }

12147     /* if the entity has not been deleted, refresh it */
12148     if ((!(isservice && svc != NULL) || (!isservice && inst != NULL)) {
12149         (void) refresh_entity(isservice, entity, fmri, inst, iter,
12150             name_buf);
12151     }

12153 out:
12154     if (isservice && (inst != NULL && iter != NULL)) {
12155         free(name_buf);
12156         scf_iter_destroy(iter);
12157         scf_instance_destroy(inst);
12158     }

12160     if (!isservice && svc != NULL) {
12161         scf_service_destroy(svc);
12162     }

12164     scf_pg_destroy(pg);
12165     if (entity != NULL)
12166         entity_destroy(entity, isservice);

12168     return (result);
12169 }

12171 static int
12172 delete_dependents(scf_propertygroup_t *pg)
12173 {
12174     char *pgty, *name, *fmri;
12175     scf_property_t *prop;
12176     scf_value_t *val;
12177     scf_iter_t *iter;
12178     int r;
12179     scf_error_t err;

12181     /* Verify that the pg has the correct type. */
12182     pgty = safe_malloc(max_scf_pg_type_len + 1);
12183     if (scf_pg_get_type(pg, pgty, max_scf_pg_type_len + 1) < 0)
12184         scfdie();

```

```

12186     if (strcmp(pgty, scf_group_framework) != 0) {
12187         if (g_verbose) {
12188             fmri = safe_malloc(max_scf_fmri_len + 1);
12189             if (scf_pg_to_fmri(pg, fmri, max_scf_fmri_len + 1) < 0)
12190                 scfdie();
12191
12192             warn(gettext("Property group %s is not of expected "
12193                 "type %s.\n"), fmri, scf_group_framework);
12194
12195             free(fmri);
12196         }
12197
12198         free(pgty);
12199         return (-1);
12200     }
12201
12202     free(pgty);
12203
12204     /* map delete_dependency_pg onto the properties. */
12205     if ((prop = scf_property_create(g_hndl)) == NULL ||
12206         (val = scf_value_create(g_hndl)) == NULL ||
12207         (iter = scf_iter_create(g_hndl)) == NULL)
12208         scfdie();
12209
12210     if (scf_iter_pg_properties(iter, pg) != SCF_SUCCESS)
12211         scfdie();
12212
12213     name = safe_malloc(max_scf_name_len + 1);
12214     fmri = safe_malloc(max_scf_fmri_len + 2);
12215
12216     while ((r = scf_iter_next_property(iter, prop)) == 1) {
12217         scf_type_t ty;
12218
12219         if (scf_property_get_name(prop, name, max_scf_name_len + 1) < 0)
12220             scfdie();
12221
12222         if (scf_property_type(prop, &ty) != SCF_SUCCESS)
12223             scfdie();
12224
12225         if ((ty != SCF_TYPE_ASTRING &&
12226             prop_check_type(prop, SCF_TYPE_FMRI) != 0) ||
12227             prop_get_val(prop, val) != 0)
12228             continue;
12229
12230         if (scf_value_get_astring(val, fmri, max_scf_fmri_len + 2) < 0)
12231             scfdie();
12232
12233         err = delete_dependency_pg(fmri, name);
12234         if (err == SCF_ERROR_INVALID_ARGUMENT && g_verbose) {
12235             if (scf_property_to_fmri(prop, fmri,
12236                 max_scf_fmri_len + 2) < 0)
12237                 scfdie();
12238
12239             warn(gettext("Value of %s is not a valid FMRI.\n"),
12240                 fmri);
12241         } else if (err == SCF_ERROR_TYPE_MISMATCH && g_verbose) {
12242             warn(gettext("Property group \"%s\" of entity \"%s\" "
12243                 "does not have dependency type.\n"), name, fmri);
12244         } else if (err == SCF_ERROR_PERMISSION_DENIED && g_verbose) {
12245             warn(gettext("Could not delete property group \"%s\" "
12246                 "of entity \"%s\" (permission denied).\n"), name,
12247                 fmri);
12248         }
12249     }
12250     if (r == -1)

```

```

12251         scfdie();
12252
12253         scf_value_destroy(val);
12254         scf_property_destroy(prop);
12255
12256         return (0);
12257     }
12258
12259     /*
12260     * Returns 1 if the instance may be running, and 0 otherwise.
12261     */
12262     static int
12263     inst_is_running(scf_instance_t *inst)
12264     {
12265         scf_propertygroup_t *pg;
12266         scf_property_t *prop;
12267         scf_value_t *val;
12268         char buf[MAX_SCF_STATE_STRING_SZ];
12269         int ret = 0;
12270         ssize_t szret;
12271
12272         if ((pg = scf_pg_create(g_hndl)) == NULL ||
12273             (prop = scf_property_create(g_hndl)) == NULL ||
12274             (val = scf_value_create(g_hndl)) == NULL)
12275             scfdie();
12276
12277         if (scf_instance_get_pg(inst, SCF_PG_RESTARTER, pg) != SCF_SUCCESS) {
12278             if (scf_error() != SCF_ERROR_NOT_FOUND)
12279                 scfdie();
12280             goto out;
12281         }
12282
12283         if (pg_get_prop(pg, SCF_PROPERTY_STATE, prop) != 0 ||
12284             prop_check_type(prop, SCF_TYPE_ASTRING) != 0 ||
12285             prop_get_val(prop, val) != 0)
12286             goto out;
12287
12288         szret = scf_value_get_astring(val, buf, sizeof(buf));
12289         assert(szret >= 0);
12290
12291         ret = (strcmp(buf, SCF_STATE_STRING_ONLINE) == 0 ||
12292             strcmp(buf, SCF_STATE_STRING_DEGRADED) == 0) ? 1 : 0;
12293
12294     out:
12295         scf_value_destroy(val);
12296         scf_property_destroy(prop);
12297         scf_pg_destroy(pg);
12298         return (ret);
12299     }
12300
12301     static uint8_t
12302     pg_is_external_dependency(scf_propertygroup_t *pg)
12303     {
12304         char *type;
12305         scf_value_t *val;
12306         scf_property_t *prop;
12307         uint8_t b = B_FALSE;
12308
12309         type = safe_malloc(max_scf_pg_type_len + 1);
12310
12311         if (scf_pg_get_type(pg, type, max_scf_pg_type_len + 1) < 0)
12312             scfdie();
12313
12314         if ((prop = scf_property_create(g_hndl)) == NULL ||
12315             (val = scf_value_create(g_hndl)) == NULL)
12316             scfdie();

```

```

12318     if (strcmp(type, SCF_GROUP_DEPENDENCY) == 0) {
12319         if (pg_get_prop(pg, scf_property_external, prop) == 0) {
12320             if (scf_property_get_value(prop, val) != 0)
12321                 scfdie();
12322             if (scf_value_get_boolean(val, &b) != 0)
12323                 scfdie();
12324         }
12325     }

12327     free(type);
12328     (void) scf_value_destroy(val);
12329     (void) scf_property_destroy(prop);

12331     return (b);
12332 }

12334 #define DELETE_FAILURE          -1
12335 #define DELETE_SUCCESS_NOEXTDEPS 0
12336 #define DELETE_SUCCESS_EXTDEPS  1

12338 /*
12339  * lscf_instance_delete() deletes an instance. Before calling
12340  * scf_instance_delete(), though, we make sure the instance isn't
12341  * running and delete dependencies in other entities which the instance
12342  * declared as "dependents". If there are dependencies which were
12343  * created for other entities, then instead of deleting the instance we
12344  * make it "empty" by deleting all other property groups and all
12345  * snapshots.
12346  *
12347  * lscf_instance_delete() verifies that there is no external dependency pgs
12348  * before suppressing the instance. If there is, then we must not remove them
12349  * now in case the instance is re-created otherwise the dependencies would be
12350  * lost. The external dependency pgs will be removed if the dependencies are
12351  * removed.
12352  *
12353  * Returns:
12354  * DELETE_FAILURE          on failure
12355  * DELETE_SUCCESS_NOEXTDEPS on success - no external dependencies
12356  * DELETE_SUCCESS_EXTDEPS  on success - external dependencies
12357  */
12358 static int
12359 lscf_instance_delete(scf_instance_t *inst, int force)
12360 {
12361     scf_propertygroup_t *pg;
12362     scf_snapshot_t *snap;
12363     scf_iter_t *iter;
12364     int err;
12365     int external = 0;

12367     /* If we're not forcing and the instance is running, refuse. */
12368     if (!force && inst_is_running(inst)) {
12369         char *fmri;

12371         fmri = safe_malloc(max_scf_fmri_len + 1);

12373         if (scf_instance_to_fmri(inst, fmri, max_scf_fmri_len + 1) < 0)
12374             scfdie();

12376         semerr(gettext("Instance %s may be running. "
12377             "Use delete -f if it is not.\n"), fmri);

12379         free(fmri);
12380         return (DELETE_FAILURE);
12381     }

```

```

12383     pg = scf_pg_create(g_hdl);
12384     if (pg == NULL)
12385         scfdie();

12387     if (scf_instance_get_pg(inst, SCF_PG_DEPENDENTS, pg) == 0)
12388         (void) delete_dependents(pg);
12389     else if (scf_error() != SCF_ERROR_NOT_FOUND)
12390         scfdie();

12392     scf_pg_destroy(pg);

12394     /*
12395      * If the instance has some external dependencies then we must
12396      * keep them in case the instance is reimported otherwise the
12397      * dependencies would be lost on reimport.
12398      */
12399     if ((iter = scf_iter_create(g_hdl)) == NULL ||
12400         (pg = scf_pg_create(g_hdl)) == NULL)
12401         scfdie();

12403     if (scf_iter_instance_pgs(iter, inst) < 0)
12404         scfdie();

12406     while ((err = scf_iter_next_pg(iter, pg)) == 1) {
12407         if (pg_is_external_dependency(pg)) {
12408             external = 1;
12409             continue;
12410         }

12412         if (scf_pg_delete(pg) != 0) {
12413             if (scf_error() != SCF_ERROR_PERMISSION_DENIED)
12414                 scfdie();
12415             else {
12416                 semerr(emsmsg_permission_denied);

12418                 (void) scf_iter_destroy(iter);
12419                 (void) scf_pg_destroy(pg);
12420                 return (DELETE_FAILURE);
12421             }
12422         }
12423     }

12425     if (err == -1)
12426         scfdie();

12428     (void) scf_iter_destroy(iter);
12429     (void) scf_pg_destroy(pg);

12431     if (external) {
12432         /*
12433          * All the pgs have been deleted for the instance except
12434          * the ones holding the external dependencies.
12435          * For the job to be complete, we must also delete the
12436          * snapshots associated with the instance.
12437          */
12438         if ((snap = scf_snapshot_create((scf_handle_t *)g_hdl)) ==
12439             NULL)
12440             scfdie();
12441         if ((iter = scf_iter_create((scf_handle_t *)g_hdl)) == NULL)
12442             scfdie();

12444         if (scf_iter_instance_snapshots(iter, inst) == -1)
12445             scfdie();

12447         while ((err = scf_iter_next_snapshot(iter, snap)) == 1) {
12448             if (!scf_snapshot_delete(snap)) {

```

```

12449         if (scf_error() != SCF_ERROR_PERMISSION_DENIED)
12450             scfdie();
12452         semerr(msg_permission_denied);
12454         (void) scf_iter_destroy(iter);
12455         (void) scf_snapshot_destroy(snap);
12456         return (DELETE_FAILURE);
12457     }
12458 }
12460     if (err == -1)
12461         scfdie();
12463         (void) scf_iter_destroy(iter);
12464         (void) scf_snapshot_destroy(snap);
12465         return (DELETE_SUCCESS_EXTDEPS);
12466     }
12468     if (scf_instance_delete(inst) != 0) {
12469         if (scf_error() != SCF_ERROR_PERMISSION_DENIED)
12470             scfdie();
12472         semerr(msg_permission_denied);
12474         return (DELETE_FAILURE);
12475     }
12477     return (DELETE_SUCCESS_NOEXTDEPS);
12478 }
12480 /*
12481  * lscf_service_delete() deletes a service. Before calling
12482  * scf_service_delete(), though, we call lscf_instance_delete() for
12483  * each of the instances and delete dependencies in other entities
12484  * which were created as "dependents" of this service. If there are
12485  * dependencies which were created for other entities, then we delete
12486  * all other property groups in the service and leave it as "empty".
12487  *
12488  * lscf_service_delete() verifies that there is no external dependency
12489  * pgs at the instance & service level before suppressing the service.
12490  * If there is, then we must not remove them now in case the service
12491  * is re-imported otherwise the dependencies would be lost. The external
12492  * dependency pgs will be removed if the dependencies are removed.
12493  *
12494  * Returns:
12495  *   DELETE_FAILURE           on failure
12496  *   DELETE_SUCCESS_NOEXTDEPS on success - no external dependencies
12497  *   DELETE_SUCCESS_EXTDEPS  on success - external dependencies
12498  */
12499 static int
12500 lscf_service_delete(scf_service_t *svc, int force)
12501 {
12502     int r;
12503     scf_instance_t *inst;
12504     scf_propertygroup_t *pg;
12505     scf_iter_t *iter;
12506     int ret;
12507     int external = 0;
12509     if ((inst = scf_instance_create(g_hndl)) == NULL ||
12510         (pg = scf_pg_create(g_hndl)) == NULL ||
12511         (iter = scf_iter_create(g_hndl)) == NULL)
12512         scfdie();
12514     if (scf_iter_service_instances(iter, svc) != 0)

```

```

12515         scfdie();
12517         for (r = scf_iter_next_instance(iter, inst);
12518              r == 1;
12519              r = scf_iter_next_instance(iter, inst)) {
12521             ret = lscf_instance_delete(inst, force);
12522             if (ret == DELETE_FAILURE) {
12523                 scf_iter_destroy(iter);
12524                 scf_pg_destroy(pg);
12525                 scf_instance_destroy(inst);
12526                 return (DELETE_FAILURE);
12527             }
12529             /*
12530              * Record the fact that there is some external dependencies
12531              * at the instance level.
12532              */
12533             if (ret == DELETE_SUCCESS_EXTDEPS)
12534                 external |= 1;
12535         }
12537         if (r != 0)
12538             scfdie();
12540         /* Delete dependency property groups in dependent services. */
12541         if (scf_service_get_pg(svc, SCF_PG_DEPENDENTS, pg) == 0)
12542             (void) delete_dependents(pg);
12543         else if (scf_error() != SCF_ERROR_NOT_FOUND)
12544             scfdie();
12546         scf_iter_destroy(iter);
12547         scf_pg_destroy(pg);
12548         scf_instance_destroy(inst);
12550         /*
12551          * If the service has some external dependencies then we don't
12552          * want to remove them in case the service is re-imported.
12553          */
12554         if ((pg = scf_pg_create(g_hndl)) == NULL ||
12555             (iter = scf_iter_create(g_hndl)) == NULL)
12556             scfdie();
12558         if (scf_iter_service_pgs(iter, svc) < 0)
12559             scfdie();
12561         while ((r = scf_iter_next_pg(iter, pg)) == 1) {
12562             if (pg_is_external_dependency(pg)) {
12563                 external |= 2;
12564                 continue;
12565             }
12567             if (scf_pg_delete(pg) != 0) {
12568                 if (scf_error() != SCF_ERROR_PERMISSION_DENIED)
12569                     scfdie();
12570                 else {
12571                     semerr(msg_permission_denied);
12573                     (void) scf_iter_destroy(iter);
12574                     (void) scf_pg_destroy(pg);
12575                     return (DELETE_FAILURE);
12576                 }
12577             }
12578         }
12580         if (r == -1)

```

```

12581         scfdie();
12583         (void) scf_iter_destroy(iter);
12584         (void) scf_pg_destroy(pg);
12586         if (external != 0)
12587             return (DELETE_SUCCESS_EXTDEPS);
12589         if (scf_service_delete(svc) == 0)
12590             return (DELETE_SUCCESS_NOEXTDEPS);
12592         if (scf_error() != SCF_ERROR_PERMISSION_DENIED)
12593             scfdie();
12595         semerr(msg_permission_denied);
12596         return (DELETE_FAILURE);
12597     }
12599     static int
12600     delete_callback(void *data, scf_walkinfo_t *wip)
12601     {
12602         int force = (int)data;
12604         if (wip->inst != NULL)
12605             (void) lscf_instance_delete(wip->inst, force);
12606         else
12607             (void) lscf_service_delete(wip->svc, force);
12609         return (0);
12610     }
12612     void
12613     lscf_delete(const char *fmri, int force)
12614     {
12615         scf_service_t *svc;
12616         scf_instance_t *inst;
12617         int ret;
12619         lscf_prep_hdl();
12621         if (cur_snap != NULL) {
12622             if (!snaplevel_is_instance(cur_level)) {
12623                 char *buf;
12625                 buf = safe_malloc(max_scf_name_len + 1);
12626                 if (scf_instance_get_name(cur_inst, buf,
12627                     max_scf_name_len + 1) >= 0) {
12628                     if (strcmp(buf, fmri) == 0) {
12629                         semerr(msg_cant_modify_snapshots);
12630                         free(buf);
12631                         return;
12632                     }
12633                 } else if (scf_error() != SCF_ERROR_DELETED) {
12634                     scfdie();
12635                 }
12636                 free(buf);
12637             }
12638         } else if (cur_inst != NULL) {
12639             /* EMPTY */;
12640         } else if (cur_svc != NULL) {
12641             inst = scf_instance_create(g_hdl);
12642             if (inst == NULL)
12643                 scfdie();
12645             if (scf_service_get_instance(cur_svc, fmri, inst) ==
12646                 SCF_SUCCESS) {

```

```

12647             (void) lscf_instance_delete(inst, force);
12648             scf_instance_destroy(inst);
12649             return;
12650         }
12652         if (scf_error() != SCF_ERROR_NOT_FOUND &&
12653             scf_error() != SCF_ERROR_INVALID_ARGUMENT)
12654             scfdie();
12656         scf_instance_destroy(inst);
12657     } else {
12658         assert(cur_scope != NULL);
12660         svc = scf_service_create(g_hdl);
12661         if (svc == NULL)
12662             scfdie();
12664         if (scf_scope_get_service(cur_scope, fmri, svc) ==
12665             SCF_SUCCESS) {
12666             (void) lscf_service_delete(svc, force);
12667             scf_service_destroy(svc);
12668             return;
12669         }
12671         if (scf_error() != SCF_ERROR_NOT_FOUND &&
12672             scf_error() != SCF_ERROR_INVALID_ARGUMENT)
12673             scfdie();
12675         scf_service_destroy(svc);
12676     }
12678     /*
12679     * Match FMRI to entity.
12680     */
12681     if ((ret = scf_walk_fmri(g_hdl, 1, (char **)&fmri, SCF_WALK_SERVICE,
12682         delete_callback, (void *)force, NULL, semerr)) != 0) {
12683         semerr(gettext("Failed to walk instances: %s\n"),
12684             scf_strerror(ret));
12685     }
12686 }
12690 /*
12691 * :properties commands. These all end with "pg" or "prop" and generally
12692 * operate on the currently selected entity.
12693 */
12695 /*
12696 * Property listing. List the property groups, properties, their types and
12697 * their values for the currently selected entity.
12698 */
12699     static void
12700     list_pg_info(const scf_propertygroup_t *pg, const char *name, size_t namewidth)
12701     {
12702         char *buf;
12703         uint32_t flags;
12705         buf = safe_malloc(max_scf_pg_type_len + 1);
12707         if (scf_pg_get_type(pg, buf, max_scf_pg_type_len + 1) < 0)
12708             scfdie();
12710         if (scf_pg_get_flags(pg, &flags) != SCF_SUCCESS)
12711             scfdie();

```



```

12845         free(stability);
12846     }
12847     } else if (scf_error() != SCF_ERROR_NOT_FOUND)
12848         scfdie();
12849 }

12851 scf_property_destroy(stability_prop);
12852 scf_value_destroy(stability_val);

12854 if (pgt == NULL)
12855     return;

12857 if (pg == NULL || templates == 2) {
12858     /* print type info only if scf_tmpl_pg_name succeeds */
12859     if (scf_tmpl_pg_name(pgt, &buf) != -1) {
12860         if (pg != NULL)
12861             safe_printf("%s", TEMPL_INDENT);
12862         safe_printf("%s: ", gettext("name"));
12863         safe_printf("%s\n", buf);
12864         free(buf);
12865     }

12867     /* print type info only if scf_tmpl_pg_type succeeds */
12868     if (scf_tmpl_pg_type(pgt, &buf) != -1) {
12869         if (pg != NULL)
12870             safe_printf("%s", TEMPL_INDENT);
12871         safe_printf("%s: ", gettext("type"));
12872         safe_printf("%s\n", buf);
12873         free(buf);
12874     }
12875 }

12877 if (templates == 2 && scf_tmpl_pg_required(pgt, &required) == 0)
12878     safe_printf("%s%s: %s\n", TEMPL_INDENT, gettext("required"),
12879               required ? "true" : "false");

12881 if (templates == 2 && scf_tmpl_pg_target(pgt, &buf) > 0) {
12882     safe_printf("%s%s: %s\n", TEMPL_INDENT, gettext("target"),
12883               buf);
12884     free(buf);
12885 }

12887 if (templates == 2 && scf_tmpl_pg_common_name(pgt, NULL, &buf) > 0) {
12888     safe_printf("%s%s: %s\n", TEMPL_INDENT, gettext("common name"),
12889               buf);
12890     free(buf);
12891 }

12893 if (scf_tmpl_pg_description(pgt, NULL, &buf) > 0) {
12894     if (templates == 2)
12895         safe_printf("%s%s: %s\n", TEMPL_INDENT,
12896                   gettext("description"), buf);
12897     else
12898         safe_printf("%s%s\n", TEMPL_INDENT, buf);
12899     free(buf);
12900 }

12902 }

12904 /*
12905  * With as_value set to true, indent as appropriate for the value level.
12906  * If false, indent to appropriate level for inclusion in constraint
12907  * or choice printout.
12908  */
12909 static void
12910 print_template_value_details(scf_prop_tmpl_t *prt, const char *val_buf,

```

```

12911     int as_value)
12912 {
12913     char *buf;

12915     if (scf_tmpl_value_common_name(prt, NULL, val_buf, &buf) > 0) {
12916         if (as_value == 0)
12917             safe_printf("%s", TEMPL_CHOICE_INDENT);
12918         else
12919             safe_printf("%s", TEMPL_INDENT);
12920         safe_printf("%s: %s\n", gettext("value common name"), buf);
12921         free(buf);
12922     }

12924     if (scf_tmpl_value_description(prt, NULL, val_buf, &buf) > 0) {
12925         if (as_value == 0)
12926             safe_printf("%s", TEMPL_CHOICE_INDENT);
12927         else
12928             safe_printf("%s", TEMPL_INDENT);
12929         safe_printf("%s: %s\n", gettext("value description"), buf);
12930         free(buf);
12931     }
12932 }

12934 static void
12935 print_template_value(scf_prop_tmpl_t *prt, const char *val_buf)
12936 {
12937     safe_printf("%s%s: ", TEMPL_VALUE_INDENT, gettext("value"));
12938     /* This is to be human-readable, so don't use CHARS_TO_QUOTE */
12939     safe_printf("%s\n", val_buf);

12941     print_template_value_details(prt, val_buf, 1);
12942 }

12944 static void
12945 print_template_constraints(scf_prop_tmpl_t *prt, int verbose)
12946 {
12947     int i, printed = 0;
12948     scf_values_t values;
12949     scf_count_ranges_t c_ranges;
12950     scf_int_ranges_t i_ranges;

12952     printed = 0;
12953     i = 0;
12954     if (scf_tmpl_value_name_constraints(prt, &values) == 0) {
12955         safe_printf("%s%s:\n", TEMPL_VALUE_INDENT,
12956                   gettext("value constraints"));
12957         printed++;
12958         for (i = 0; i < values.value_count; ++i) {
12959             safe_printf("%s%s: %s\n", TEMPL_INDENT,
12960                       gettext("value name"), values.values_as_strings[i]);
12961             if (verbose == 1)
12962                 print_template_value_details(prt,
12963                                               values.values_as_strings[i], 0);
12964         }
12966     }

12966     scf_values_destroy(&values);
12967 }

12969 if (scf_tmpl_value_count_range_constraints(prt, &c_ranges) == 0) {
12970     if (printed++ == 0)
12971         safe_printf("%s%s:\n", TEMPL_VALUE_INDENT,
12972                   gettext("value constraints"));
12973     for (i = 0; i < c_ranges.scr_num_ranges; ++i) {
12974         safe_printf("%s%s: %llu to %llu\n", TEMPL_INDENT,
12975                   gettext("range"), c_ranges.scr_min[i],
12976                   c_ranges.scr_max[i]);

```



```

12977     }
12978     scf_count_ranges_destroy(&c_ranges);
12979 } else if (scf_error() == SCF_ERROR_CONSTRAINT_VIOLATED &&
12980 scf_tmpl_value_int_range_constraints(prt, &i_ranges) == 0) {
12981     if (printed++ == 0)
12982         safe_printf("%s%s:\n", TEMPL_VALUE_INDENT,
12983             gettext("value constraints"));
12984     for (i = 0; i < i_ranges.sir_num_ranges; ++i) {
12985         safe_printf("%s%s: %lld to %lld\n", TEMPL_INDENT,
12986             gettext("range"), i_ranges.sir_min[i],
12987             i_ranges.sir_max[i]);
12988     }
12989     scf_int_ranges_destroy(&i_ranges);
12990 }
12991 }
12993 static void
12994 print_template_choices(scf_prop_tmpl_t *prt, int verbose)
12995 {
12996     int i = 0, printed = 0;
12997     scf_values_t values;
12998     scf_count_ranges_t c_ranges;
12999     scf_int_ranges_t i_ranges;
13001     printed = 0;
13002     if (scf_tmpl_value_name_choices(prt, &values) == 0) {
13003         safe_printf("%s%s:\n", TEMPL_VALUE_INDENT,
13004             gettext("value constraints"));
13005         printed++;
13006         for (i = 0; i < values.value_count; i++) {
13007             safe_printf("%s%s: %s\n", TEMPL_INDENT,
13008                 gettext("value name"), values.values_as_strings[i]);
13009             if (verbose == 1)
13010                 print_template_value_details(prt,
13011                     values.values_as_strings[i], 0);
13012         }
13014     }
13015     scf_values_destroy(&values);
13017     if (scf_tmpl_value_count_range_choices(prt, &c_ranges) == 0) {
13018         for (i = 0; i < c_ranges.scr_num_ranges; ++i) {
13019             if (printed++ == 0)
13020                 safe_printf("%s%s:\n", TEMPL_VALUE_INDENT,
13021                     gettext("value choices"));
13022             safe_printf("%s%s: %llu to %llu\n", TEMPL_INDENT,
13023                 gettext("range"), c_ranges.scr_min[i],
13024                 c_ranges.scr_max[i]);
13025         }
13026         scf_count_ranges_destroy(&c_ranges);
13027     } else if (scf_error() == SCF_ERROR_CONSTRAINT_VIOLATED &&
13028 scf_tmpl_value_int_range_choices(prt, &i_ranges) == 0) {
13029         for (i = 0; i < i_ranges.sir_num_ranges; ++i) {
13030             if (printed++ == 0)
13031                 safe_printf("%s%s:\n", TEMPL_VALUE_INDENT,
13032                     gettext("value choices"));
13033             safe_printf("%s%s: %lld to %lld\n", TEMPL_INDENT,
13034                 gettext("range"), i_ranges.sir_min[i],
13035                 i_ranges.sir_max[i]);
13036         }
13037         scf_int_ranges_destroy(&i_ranges);
13038     }
13039 }
13041 static void
13042 list_values_by_template(scf_prop_tmpl_t *prt)

```

```

13043 {
13044     print_template_constraints(prt, 1);
13045     print_template_choices(prt, 1);
13046 }
13048 static void
13049 list_values_tmpl(scf_prop_tmpl_t *prt, scf_property_t *prop)
13050 {
13051     char *val_buf;
13052     scf_iter_t *iter;
13053     scf_value_t *val;
13054     int ret;
13056     if ((iter = scf_iter_create(g_hndl)) == NULL ||
13057         (val = scf_value_create(g_hndl)) == NULL)
13058         scfdie();
13060     if (scf_iter_property_values(iter, prop) != SCF_SUCCESS)
13061         scfdie();
13063     val_buf = safe_malloc(max_scf_value_len + 1);
13065     while ((ret = scf_iter_next_value(iter, val)) == 1) {
13066         if (scf_value_get_as_string(val, val_buf,
13067             max_scf_value_len + 1) < 0)
13068             scfdie();
13070         print_template_value(prt, val_buf);
13071     }
13072     if (ret != 0 && scf_error() != SCF_ERROR_PERMISSION_DENIED)
13073         scfdie();
13074     free(val_buf);
13076     print_template_constraints(prt, 0);
13077     print_template_choices(prt, 0);
13079 }
13081 /*
13082  * Outputs property info for the describe subcommand
13083  * Verbose output if templates == 2, -v option of svccfg describe
13084  * Displays template data if prop is not NULL, -t option of svccfg describe
13085  */
13086 static void
13087 list_prop_tmpl(scf_prop_tmpl_t *prt, scf_property_t *prop, int templates)
13088 {
13089     char *buf;
13090     uint8_t u_buf;
13091     int i;
13092     uint64_t min, max;
13093     scf_values_t values;
13095     if (prt == NULL || templates == 0)
13096         return;
13098     if (prop == NULL) {
13099         safe_printf("%s%s: ", TEMPL_VALUE_INDENT, gettext("name"));
13100         if (scf_tmpl_prop_name(prt, &buf) > 0) {
13101             safe_printf("%s\n", buf);
13102             free(buf);
13103         } else
13104             safe_printf("(%s)\n", gettext("any"));
13105     }
13107     if (prop == NULL || templates == 2) {
13108         if (prop != NULL)

```

```

13109         safe_printf("%s", TEMPL_INDENT);
13110     else
13111         safe_printf("%s", TEMPL_VALUE_INDENT);
13112     safe_printf("%s: ", gettext("type"));
13113     if ((buf = _scf_read_tmpl_prop_type_as_string(prt)) != NULL) {
13114         safe_printf("%s\n", buf);
13115         free(buf);
13116     } else
13117         safe_printf("(%s)\n", gettext("any"));
13118 }

13120 if (templates == 2 && scf_tmpl_prop_required(prt, &u_buf) == 0)
13121     safe_printf("%s: %s\n", TEMPL_INDENT, gettext("required"),
13122                u_buf ? "true" : "false");

13124 if (templates == 2 && scf_tmpl_prop_common_name(prt, NULL, &buf) > 0) {
13125     safe_printf("%s: %s\n", TEMPL_INDENT, gettext("common name"),
13126                buf);
13127     free(buf);
13128 }

13130 if (templates == 2 && scf_tmpl_prop_units(prt, NULL, &buf) > 0) {
13131     safe_printf("%s: %s\n", TEMPL_INDENT, gettext("units"),
13132                buf);
13133     free(buf);
13134 }

13136 if (scf_tmpl_prop_description(prt, NULL, &buf) > 0) {
13137     safe_printf("%s\n", TEMPL_INDENT, buf);
13138     free(buf);
13139 }

13141 if (templates == 2 && scf_tmpl_prop_visibility(prt, &u_buf) == 0)
13142     safe_printf("%s: %s\n", TEMPL_INDENT, gettext("visibility"),
13143                scf_tmpl_visibility_to_string(u_buf));

13145 if (templates == 2 && scf_tmpl_prop_cardinality(prt, &min, &max) == 0) {
13146     safe_printf("%s: %" PRIu64 "\n", TEMPL_INDENT,
13147                gettext("minimum number of values"), min);
13148     if (max == ULLONG_MAX) {
13149         safe_printf("%s: %s\n", TEMPL_INDENT,
13150                    gettext("maximum number of values"),
13151                    gettext("unlimited"));
13152     } else {
13153         safe_printf("%s: %" PRIu64 "\n", TEMPL_INDENT,
13154                    gettext("maximum number of values"), max);
13155     }
13156 }

13158 if (templates == 2 && scf_tmpl_prop_internal_seps(prt, &values) == 0) {
13159     for (i = 0; i < values.value_count; i++) {
13160         if (i == 0) {
13161             safe_printf("%s:", TEMPL_INDENT,
13162                        gettext("internal separators"));
13163         }
13164         safe_printf(" \"%s\"", values.values_as_strings[i]);
13165     }
13166     safe_printf("\n");
13167 }

13169 if (templates != 2)
13170     return;

13172 if (prop != NULL)
13173     list_values_tmpl(prt, prop);
13174 else

```

```

13175         list_values_by_template(prt);
13176     }

13178 static char *
13179 read_astring(scf_propertygroup_t *pg, const char *prop_name)
13180 {
13181     char *rv;

13183     rv = _scf_read_single_astring_from_pg(pg, prop_name);
13184     if (rv == NULL) {
13185         switch (scf_error()) {
13186             case SCF_ERROR_NOT_FOUND:
13187                 break;
13188             default:
13189                 scfdie();
13190         }
13191     }
13192     return (rv);
13193 }

13195 static void
13196 display_documentation(scf_iter_t *iter, scf_propertygroup_t *pg)
13197 {
13198     size_t doc_len;
13199     size_t man_len;
13200     char *pg_name;
13201     char *text = NULL;
13202     int rv;

13204     doc_len = strlen(SCF_PG_TM_DOC_PREFIX);
13205     man_len = strlen(SCF_PG_TM_MAN_PREFIX);
13206     pg_name = safe_malloc(max_scf_name_len + 1);
13207     while ((rv = scf_iter_next_pg(iter, pg)) == 1) {
13208         if (scf_pg_get_name(pg, pg_name, max_scf_name_len + 1) == -1) {
13209             scfdie();
13210         }
13211         if (strncmp(pg_name, SCF_PG_TM_DOC_PREFIX, doc_len) == 0) {
13212             /* Display doc link and and uri */
13213             safe_printf("%s:\n", TEMPL_INDENT,
13214                        gettext("doc_link"));
13215             text = read_astring(pg, SCF_PROPERTY_TM_NAME);
13216             if (text != NULL) {
13217                 safe_printf("%s: %s\n", TEMPL_INDENT,
13218                            TEMPL_INDENT, gettext("name"), text);
13219                 uu_free(text);
13220             }
13221             text = read_astring(pg, SCF_PROPERTY_TM_URI);
13222             if (text != NULL) {
13223                 safe_printf("%s: %s\n", TEMPL_INDENT_2X,
13224                            gettext("uri"), text);
13225                 uu_free(text);
13226             }
13227         } else if (strncmp(pg_name, SCF_PG_TM_MAN_PREFIX,
13228                            man_len) == 0) {
13229             /* Display manpage title, section and path */
13230             safe_printf("%s:\n", TEMPL_INDENT,
13231                        gettext("manpage"));
13232             text = read_astring(pg, SCF_PROPERTY_TM_TITLE);
13233             if (text != NULL) {
13234                 safe_printf("%s: %s\n", TEMPL_INDENT,
13235                            TEMPL_INDENT, gettext("title"), text);
13236                 uu_free(text);
13237             }
13238             text = read_astring(pg, SCF_PROPERTY_TM_SECTION);
13239             if (text != NULL) {
13240                 safe_printf("%s: %s\n", TEMPL_INDENT,

```

```

13241         TEMPL_INDENT, gettext("section"), text);
13242         uu_free(text);
13243     }
13244     text = read_astring(pg, SCF_PROPERTY_TM_MANPATH);
13245     if (text != NULL) {
13246         safe_printf("%s%s: %s\n", TEMPL_INDENT,
13247                   TEMPL_INDENT, gettext("manpath"), text);
13248         uu_free(text);
13249     }
13250 }
13251 }
13252 if (rv == -1)
13253     scfdie();

13255 done:
13256     free(pg_name);
13257 }

13259 static void
13260 list_entity_tmpl(int templates)
13261 {
13262     char *common_name = NULL;
13263     char *description = NULL;
13264     char *locale = NULL;
13265     scf_iter_t *iter;
13266     scf_propertygroup_t *pg;
13267     scf_property_t *prop;
13268     int r;
13269     scf_value_t *val;

13271     if ((pg = scf_pg_create(g_hndl)) == NULL ||
13272         (prop = scf_property_create(g_hndl)) == NULL ||
13273         (val = scf_value_create(g_hndl)) == NULL ||
13274         (iter = scf_iter_create(g_hndl)) == NULL)
13275         scfdie();

13277     locale = setlocale(LC_MESSAGES, NULL);

13279     if (get_pg(SCF_PG_TM_COMMON_NAME, pg) == 0) {
13280         common_name = safe_malloc(max_scf_value_len + 1);

13282         /* Try both the current locale and the "C" locale. */
13283         if (scf_pg_get_property(pg, locale, prop) == 0 ||
13284             (scf_error() == SCF_ERROR_NOT_FOUND &&
13285              scf_pg_get_property(pg, "C", prop) == 0)) {
13286             if (prop_get_val(prop, val) == 0 &&
13287                 scf_value_get_ustring(val, common_name,
13288                                       max_scf_value_len + 1) != -1) {
13289                 safe_printf("%s%s: %s\n", TEMPL_INDENT,
13290                             gettext("common name"), common_name);
13291             }
13292         }
13293     }

13295     /*
13296     * Do description, manpages, and doc links if templates == 2.
13297     */
13298     if (templates == 2) {
13299         /* Get the description. */
13300         if (get_pg(SCF_PG_TM_DESCRIPTION, pg) == 0) {
13301             description = safe_malloc(max_scf_value_len + 1);

13303             /* Try both the current locale and the "C" locale. */
13304             if (scf_pg_get_property(pg, locale, prop) == 0 ||
13305                 (scf_error() == SCF_ERROR_NOT_FOUND &&
13306                  scf_pg_get_property(pg, "C", prop) == 0)) {

```

```

13307         if (prop_get_val(prop, val) == 0 &&
13308             scf_value_get_ustring(val, description,
13309                                   max_scf_value_len + 1) != -1) {
13310             safe_printf("%s%s: %s\n", TEMPL_INDENT,
13311                         gettext("description"),
13312                         description);
13313         }
13314     }
13315 }

13317 /* Process doc_link & manpage elements. */
13318 if (cur_level != NULL) {
13319     r = scf_iter_snaplevel_pgs_typed(iter, cur_level,
13320                                       SCF_GROUP_TEMPLATE);
13321 } else if (cur_inst != NULL) {
13322     r = scf_iter_instance_pgs_typed(iter, cur_inst,
13323                                       SCF_GROUP_TEMPLATE);
13324 } else {
13325     r = scf_iter_service_pgs_typed(iter, cur_svc,
13326                                       SCF_GROUP_TEMPLATE);
13327 }
13328 if (r == 0) {
13329     display_documentation(iter, pg);
13330 }
13331 }

13333     free(common_name);
13334     free(description);
13335     scf_pg_destroy(pg);
13336     scf_property_destroy(prop);
13337     scf_value_destroy(val);
13338     scf_iter_destroy(iter);
13339 }

13341 static void
13342 listtmpl(const char *pattern, int templates)
13343 {
13344     scf_pg_tmpl_t *pgt;
13345     scf_prop_tmpl_t *prt;
13346     char *snapbuf = NULL;
13347     char *fmribuf;
13348     char *pg_name = NULL, *prop_name = NULL;
13349     ssize_t prop_name_size;
13350     char *qual_prop_name;
13351     char *search_name;
13352     int listed = 0;

13354     if ((pgt = scf_tmpl_pg_create(g_hndl)) == NULL ||
13355         (prt = scf_tmpl_prop_create(g_hndl)) == NULL)
13356         scfdie();

13358     fmribuf = safe_malloc(max_scf_name_len + 1);
13359     qual_prop_name = safe_malloc(max_scf_name_len + 1);

13361     if (cur_snap != NULL) {
13362         snapbuf = safe_malloc(max_scf_name_len + 1);
13363         if (scf_snapshot_get_name(cur_snap, snapbuf,
13364                                   max_scf_name_len + 1) < 0)
13365             scfdie();
13366     }

13368     if (cur_inst != NULL) {
13369         if (scf_instance_to_fmri(cur_inst, fmribuf,
13370                                   max_scf_name_len + 1) < 0)
13371             scfdie();
13372     } else if (cur_svc != NULL) {

```

```

13373         if (scf_service_to_fmri(cur_svc, fmribuf,
13374             max_scf_name_len + 1) < 0)
13375             scfdie();
13376     } else
13377         abort();

13379     /* If pattern is specified, we want to list only those items. */
13380     while (scf_tmpl_iter_pgs(pgt, fmribuf, snapbuf, NULL, 0) == 1) {
13381         listed = 0;
13382         if (pattern == NULL || (scf_tmpl_pg_name(pgt, &pg_name) > 0 &&
13383             fnmatch(pattern, pg_name, 0) == 0)) {
13384             list_pg_tmpl(pgt, NULL, templates);
13385             listed++;
13386         }
13387     }

13388     scf_tmpl_prop_reset(prt);

13390     while (scf_tmpl_iter_props(pgt, prt, 0) == 0) {
13391         search_name = NULL;
13392         prop_name_size = scf_tmpl_prop_name(prt, &prop_name);
13393         if ((prop_name_size > 0) && (pg_name != NULL)) {
13394             if (snprintf(qual_prop_name,
13395                 max_scf_name_len + 1, "%s/%s",
13396                 pg_name, prop_name) >=
13397                 max_scf_name_len + 1) {
13398                 prop_name_size = -1;
13399             } else {
13400                 search_name = qual_prop_name;
13401             }
13402         }
13403         if (listed > 0 || pattern == NULL ||
13404             (prop_name_size > 0 &&
13405             fnmatch(pattern, search_name,
13406             FNM_PATHNAME) == 0))
13407             list_prop_tmpl(prt, NULL, templates);
13408         if (prop_name != NULL) {
13409             free(prop_name);
13410             prop_name = NULL;
13411         }
13412     }
13413     if (pg_name != NULL) {
13414         free(pg_name);
13415         pg_name = NULL;
13416     }
13417 }

13419     scf_tmpl_prop_destroy(prt);
13420     scf_tmpl_pg_destroy(pgt);
13421     free(snapbuf);
13422     free(fmribuf);
13423     free(qual_prop_name);
13424 }

13426 static void
13427 listprop(const char *pattern, int only_pgs, int templates)
13428 {
13429     scf_propertygroup_t *pg;
13430     scf_property_t *prop;
13431     scf_iter_t *iter, *piter;
13432     char *pgnbuf, *prnbuf, *ppnbuf;
13433     scf_pg_tmpl_t *pgt, *pgtp;
13434     scf_prop_tmpl_t *prt;

13436     void **objects;
13437     char **names;
13438     void **tpls;

```

```

13439     int allocd, i;

13441     int ret;
13442     ssize_t pgnlen, prnlen, szret;
13443     size_t max_len = 0;

13445     if (cur_svc == NULL && cur_inst == NULL) {
13446         semerr(msg_entity_not_selected);
13447         return;
13448     }

13450     if ((pg = scf_pg_create(g_hndl)) == NULL ||
13451         (prop = scf_property_create(g_hndl)) == NULL ||
13452         (iter = scf_iter_create(g_hndl)) == NULL ||
13453         (piter = scf_iter_create(g_hndl)) == NULL ||
13454         (prt = scf_tmpl_prop_create(g_hndl)) == NULL ||
13455         (pgt = scf_tmpl_pg_create(g_hndl)) == NULL)
13456         scfdie();

13458     prnbuf = safe_malloc(max_scf_name_len + 1);

13460     if (cur_level != NULL)
13461         ret = scf_iter_snaplevel_pgs(iter, cur_level);
13462     else if (cur_inst != NULL)
13463         ret = scf_iter_instance_pgs(iter, cur_inst);
13464     else
13465         ret = scf_iter_service_pgs(iter, cur_svc);
13466     if (ret != 0) {
13467         return;
13468     }

13470     /*
13471     * We want to only list items which match pattern, and we want the
13472     * second column to line up, so during the first pass we'll save
13473     * matching items, their names, and their templates in objects,
13474     * names, and tpls, computing the maximum name length as we go,
13475     * and then we'll print them out.
13476     *
13477     * Note: We always keep an extra slot available so the array can be
13478     * NULL-terminated.
13479     */
13480     i = 0;
13481     allocd = 1;
13482     objects = safe_malloc(sizeof (*objects));
13483     names = safe_malloc(sizeof (*names));
13484     tpls = safe_malloc(sizeof (*tpls));

13486     while ((ret = scf_iter_next_pg(iter, pg)) == 1) {
13487         int new_pg = 0;
13488         int print_props = 0;
13489         pgtp = NULL;

13491         pgnlen = scf_pg_get_name(pg, NULL, 0);
13492         if (pgnlen < 0)
13493             scfdie();

13495         pgnbuf = safe_malloc(pgnlen + 1);

13497         szret = scf_pg_get_name(pg, pgnbuf, pgnlen + 1);
13498         if (szret < 0)
13499             scfdie();
13500         assert(szret <= pgnlen);

13502         if (scf_tmpl_get_by_pg(pg, pgt, 0) == -1) {
13503             if (scf_error() != SCF_ERROR_NOT_FOUND)
13504                 scfdie();

```

```

13505         pgtp = NULL;
13506     } else {
13507         pgtp = pgt;
13508     }

13510     if (pattern == NULL ||
13511         fnmatch(pattern, pgnbuf, 0) == 0) {
13512         if (i+1 >= allocd) {
13513             allocd *= 2;
13514             objects = realloc(objects,
13515                 sizeof(*objects) * allocd);
13516             names =
13517                 realloc(names, sizeof(*names) * allocd);
13518             tmpls = realloc(tmpls,
13519                 sizeof(*tmpls) * allocd);
13520             if (objects == NULL || names == NULL ||
13521                 tmpls == NULL)
13522                 uu_die(gettext("Out of memory"));
13523         }
13524         objects[i] = pg;
13525         names[i] = pgnbuf;

13527         if (pgtp == NULL)
13528             tmpls[i] = NULL;
13529         else
13530             tmpls[i] = pgt;

13532         ++i;

13534         if (pgnlen > max_len)
13535             max_len = pgnlen;

13537         new_pg = 1;
13538         print_props = 1;
13539     }

13541     if (only_pgs) {
13542         if (new_pg) {
13543             pg = scf_pg_create(g_hdl);
13544             if (pg == NULL)
13545                 scfdie();
13546             pgt = scf_tmpl_pg_create(g_hdl);
13547             if (pgt == NULL)
13548                 scfdie();
13549         } else
13550             free(pgnbuf);

13552         continue;
13553     }

13555     if (scf_iter_pg_properties(piter, pg) != SCF_SUCCESS)
13556         scfdie();

13558     while ((ret = scf_iter_next_property(piter, prop)) == 1) {
13559         prnlen = scf_property_get_name(prop, prnbuf,
13560             max_scf_name_len + 1);
13561         if (prnlen < 0)
13562             scfdie();

13564         /* Will prepend the property group name and a slash. */
13565         prnlen += pgnlen + 1;

13567         ppnbuf = safe_malloc(prnlen + 1);

13569         if (snprintf(ppnbuf, prnlen + 1, "%s/%s", pgnbuf,
13570             prnbuf) < 0)

```

```

13571         uu_die("snprintf");

13573         if (pattern == NULL || print_props == 1 ||
13574             fnmatch(pattern, ppnbuf, 0) == 0) {
13575             if (i+1 >= allocd) {
13576                 allocd *= 2;
13577                 objects = realloc(objects,
13578                     sizeof(*objects) * allocd);
13579                 names = realloc(names,
13580                     sizeof(*names) * allocd);
13581                 tmpls = realloc(tmpls,
13582                     sizeof(*tmpls) * allocd);
13583                 if (objects == NULL || names == NULL ||
13584                     tmpls == NULL)
13585                     uu_die(gettext(
13586                         "Out of memory"));
13587             }

13589             objects[i] = prop;
13590             names[i] = ppnbuf;

13592             if (pgtp != NULL) {
13593                 if (scf_tmpl_get_by_prop(pgt, prnbuf,
13594                     prt, 0) < 0) {
13595                     if (scf_error() !=
13596                         SCF_ERROR_NOT_FOUND)
13597                         scfdie();
13598                     tmpls[i] = NULL;
13599                 } else {
13600                     tmpls[i] = prt;
13601                 }
13602             } else {
13603                 tmpls[i] = NULL;
13604             }

13606             ++i;

13608             if (prnlen > max_len)
13609                 max_len = prnlen;

13611             prop = scf_property_create(g_hdl);
13612             prt = scf_tmpl_prop_create(g_hdl);
13613         } else {
13614             free(ppnbuf);
13615         }
13616     }

13618     if (new_pg) {
13619         pg = scf_pg_create(g_hdl);
13620         if (pg == NULL)
13621             scfdie();
13622         pgt = scf_tmpl_pg_create(g_hdl);
13623         if (pgt == NULL)
13624             scfdie();
13625     } else
13626         free(pgnbuf);
13627 }
13628 if (ret != 0)
13629     scfdie();

13631     objects[i] = NULL;

13633     scf_pg_destroy(pg);
13634     scf_tmpl_pg_destroy(pgt);
13635     scf_property_destroy(prop);
13636     scf_tmpl_prop_destroy(prt);

```

```

13638     for (i = 0; objects[i] != NULL; ++i) {
13639         if (strchr(names[i], '/') == NULL) {
13640             /* property group */
13641             pg = (scf_propertygroup_t *)objects[i];
13642             pgt = (scf_pg_tmpl_t *)tpls[i];
13643             list_pg_info(pg, names[i], max_len);
13644             list_pg_tmpl(pgt, pg, templates);
13645             free(names[i]);
13646             scf_pg_destroy(pg);
13647             if (pgt != NULL)
13648                 scf_tmpl_pg_destroy(pgt);
13649         } else {
13650             /* property */
13651             prop = (scf_property_t *)objects[i];
13652             prt = (scf_prop_tmpl_t *)tpls[i];
13653             list_prop_info(prop, names[i], max_len);
13654             list_prop_tmpl(prt, prop, templates);
13655             free(names[i]);
13656             scf_property_destroy(prop);
13657             if (prt != NULL)
13658                 scf_tmpl_prop_destroy(prt);
13659         }
13660     }

13662     free(names);
13663     free(objects);
13664     free(tpls);
13665 }

13667 void
13668 lscf_listpg(const char *pattern)
13669 {
13670     lscf_prep_hdl();

13672     listprop(pattern, 1, 0);
13673 }

13675 /*
13676  * Property group and property creation, setting, and deletion. setprop (and
13677  * its alias, addprop) can either create a property group of a given type, or
13678  * it can create or set a property to a given type and list of values.
13679  */
13680 void
13681 lscf_addpg(const char *name, const char *type, const char *flags)
13682 {
13683     scf_propertygroup_t *pg;
13684     int ret;
13685     uint32_t flgs = 0;
13686     const char *cp;

13689     lscf_prep_hdl();

13691     if (cur_snap != NULL) {
13692         semerr(msg_cant_modify_snapshots);
13693         return;
13694     }

13696     if (cur_inst == NULL && cur_svc == NULL) {
13697         semerr(msg_entity_not_selected);
13698         return;
13699     }

13701     if (flags != NULL) {
13702         for (cp = flags; *cp != '\0'; ++cp) {

```

```

13703         switch (*cp) {
13704             case 'P':
13705                 flgs |= SCF_PG_FLAG_NONPERSISTENT;
13706                 break;

13708             case 'p':
13709                 flgs &= ~SCF_PG_FLAG_NONPERSISTENT;
13710                 break;

13712             default:
13713                 semerr(gettext("Invalid property group flag "
13714                                "%c."), *cp);
13715                 return;
13716         }
13717     }

13720     pg = scf_pg_create(g_hdl);
13721     if (pg == NULL)
13722         scfdie();

13724     if (cur_inst != NULL)
13725         ret = scf_instance_add_pg(cur_inst, name, type, flgs, pg);
13726     else
13727         ret = scf_service_add_pg(cur_svc, name, type, flgs, pg);

13729     if (ret != SCF_SUCCESS) {
13730         switch (scf_error()) {
13731             case SCF_ERROR_INVALID_ARGUMENT:
13732                 semerr(gettext("Name, type, or flags are invalid.\n"));
13733                 break;

13735             case SCF_ERROR_EXISTS:
13736                 semerr(gettext("Property group already exists.\n"));
13737                 break;

13739             case SCF_ERROR_PERMISSION_DENIED:
13740                 semerr(msg_permission_denied);
13741                 break;

13743             case SCF_ERROR_BACKEND_ACCESS:
13744                 semerr(gettext("Backend refused access.\n"));
13745                 break;

13747             default:
13748                 scfdie();
13749         }
13750     }

13752     scf_pg_destroy(pg);

13754     private_refresh();
13755 }

13757 void
13758 lscf_delpg(char *name)
13759 {
13760     lscf_prep_hdl();

13762     if (cur_snap != NULL) {
13763         semerr(msg_cant_modify_snapshots);
13764         return;
13765     }

13767     if (cur_inst == NULL && cur_svc == NULL) {
13768         semerr(msg_entity_not_selected);

```

```

13769         return;
13770     }

13772     if (strchr(name, '/') != NULL) {
13773         semerr(msg_invalid_pg_name, name);
13774         return;
13775     }

13777     lscf_delprop(name);
13778 }

13780 /*
13781  * scf_delhash() is used to remove the property group related to the
13782  * hash entry for a specific manifest in the repository. pgname will be
13783  * constructed from the location of the manifest file. If deathrow isn't 0,
13784  * manifest file doesn't need to exist (manifest string will be used as
13785  * an absolute path).
13786  */
13787 void
13788 lscf_delhash(char *manifest, int deathrow)
13789 {
13790     char *pgname;

13792     if (cur_snap != NULL ||
13793         cur_inst != NULL || cur_svc != NULL) {
13794         warn(gettext("error, an entity is selected\n"));
13795         return;
13796     }

13798     /* select smf/manifest */
13799     lscf_select(HASH_SVC);
13800     /*
13801      * Translate the manifest file name to property name. In the deathrow
13802      * case, the manifest file does not need to exist.
13803      */
13804     pgname = mhash_filename_to_propname(manifest,
13805         deathrow ? B_TRUE : B_FALSE);
13806     if (pgname == NULL) {
13807         warn(gettext("cannot resolve pathname for %s\n"), manifest);
13808         return;
13809     }
13810     /* delete the hash property name */
13811     lscf_delpg(pgname);
13812 }

13814 void
13815 lscf_listprop(const char *pattern)
13816 {
13817     lscf_prep_hdl();

13819     listprop(pattern, 0, 0);
13820 }

13822 int
13823 lscf_setprop(const char *pgname, const char *type, const char *value,
13824     const uu_list_t *values)
13825 {
13826     scf_type_t ty, current_ty;
13827     scf_service_t *svc;
13828     scf_propertygroup_t *pg, *parent_pg;
13829     scf_property_t *prop, *parent_prop;
13830     scf_pg_tmpl_t *pgt;
13831     scf_prop_tmpl_t *prt;
13832     int ret, result = 0;
13833     scf_transaction_t *tx;
13834     scf_transaction_entry_t *e;

```

```

13835     scf_value_t *v;
13836     uu_list_walk_t *walk;
13837     string_list_t *sp;
13838     char *propname;
13839     int req_quotes = 0;

13841     lscf_prep_hdl();

13843     if ((e = scf_entry_create(g_hdl)) == NULL ||
13844         (svc = scf_service_create(g_hdl)) == NULL ||
13845         (parent_pg = scf_pg_create(g_hdl)) == NULL ||
13846         (pg = scf_pg_create(g_hdl)) == NULL ||
13847         (parent_prop = scf_property_create(g_hdl)) == NULL ||
13848         (prop = scf_property_create(g_hdl)) == NULL ||
13849         (pgt = scf_tmpl_pg_create(g_hdl)) == NULL ||
13850         (prt = scf_tmpl_prop_create(g_hdl)) == NULL ||
13851         (tx = scf_transaction_create(g_hdl)) == NULL)
13852         scfdie();

13854     if (cur_snap != NULL) {
13855         semerr(msg_cant_modify_snapshots);
13856         goto fail;
13857     }

13859     if (cur_inst == NULL && cur_svc == NULL) {
13860         semerr(msg_entity_not_selected);
13861         goto fail;
13862     }

13864     propname = strchr(pgname, '/');
13865     if (propname == NULL) {
13866         semerr(gettext("Property names must contain a '/'.\n"));
13867         goto fail;
13868     }

13870     *propname = '\0';
13871     ++propname;

13873     if (type != NULL) {
13874         ty = string_to_type(type);
13875         if (ty == SCF_TYPE_INVALID) {
13876             semerr(gettext("Unknown type \"%s\".\n"), type);
13877             goto fail;
13878         }
13879     }

13881     if (cur_inst != NULL)
13882         ret = scf_instance_get_pg(cur_inst, pgname, pg);
13883     else
13884         ret = scf_service_get_pg(cur_svc, pgname, pg);
13885     if (ret != SCF_SUCCESS) {
13886         switch (scf_error()) {
13887             case SCF_ERROR_NOT_FOUND:
13888                 semerr(msg_no_such_pg, pgname);
13889                 goto fail;

13891             case SCF_ERROR_INVALID_ARGUMENT:
13892                 semerr(msg_invalid_pg_name, pgname);
13893                 goto fail;

13895             default:
13896                 scfdie();
13897                 break;
13898         }
13899     }

```

```

13901     do {
13902         if (scf_pg_update(pg) == -1)
13903             scfdie();
13904         if (scf_transaction_start(tx, pg) != SCF_SUCCESS) {
13905             if (scf_error() != SCF_ERROR_PERMISSION_DENIED)
13906                 scfdie();
13907
13908             semerr(msg_permission_denied);
13909             goto fail;
13910         }
13911
13912         ret = scf_pg_get_property(pg, propname, prop);
13913         if (ret == SCF_SUCCESS) {
13914             if (scf_property_type(prop, &current_ty) != SCF_SUCCESS)
13915                 scfdie();
13916
13917             if (type == NULL)
13918                 ty = current_ty;
13919             if (scf_transaction_property_change_type(tx, e,
13920                 propname, ty) == -1)
13921                 scfdie();
13922
13923         } else if (scf_error() == SCF_ERROR_NOT_FOUND) {
13924             /* Infer the type, if possible. */
13925             if (type == NULL) {
13926                 /*
13927                  * First check if we're an instance and the
13928                  * property is set on the service.
13929                  */
13930                 if (cur_inst != NULL &&
13931                     scf_instance_get_parent(cur_inst,
13932                     svc) == 0 &&
13933                     scf_service_get_pg(cur_svc, pgname,
13934                     parent_pg) == 0 &&
13935                     scf_pg_get_property(parent_pg, propname,
13936                     parent_prop) == 0 &&
13937                     scf_property_type(parent_prop,
13938                     &current_ty) == 0) {
13939                     ty = current_ty;
13940
13941                     /* Then check for a type set in a template. */
13942                 } else if (scf_tmpl_get_by_pg(pg, pgt,
13943                     0) == 0 &&
13944                     scf_tmpl_get_by_prop(pgt, propname, prt,
13945                     0) == 0 &&
13946                     scf_tmpl_prop_type(prt, &current_ty) == 0) {
13947                     ty = current_ty;
13948
13949                     /* If type can't be inferred, fail. */
13950                 } else {
13951                     semerr(gettext("Type required for new "
13952                     "properties.\n"));
13953                     goto fail;
13954                 }
13955             }
13956             if (scf_transaction_property_new(tx, e, propname,
13957                 ty) == -1)
13958                 scfdie();
13959         } else if (scf_error() == SCF_ERROR_INVALID_ARGUMENT) {
13960             semerr(msg_invalid_prop_name, propname);
13961             goto fail;
13962         } else {
13963             scfdie();
13964         }
13965
13966         if (ty == SCF_TYPE_ASTRING || ty == SCF_TYPE_USTRING)

```

```

13967             req_quotes = 1;
13968
13969             if (value != NULL) {
13970                 v = string_to_value(value, ty, 0);
13971
13972                 if (v == NULL)
13973                     goto fail;
13974
13975                 ret = scf_entry_add_value(e, v);
13976                 assert(ret == SCF_SUCCESS);
13977             } else {
13978                 assert(values != NULL);
13979
13980                 walk = uu_list_walk_start((uu_list_t *)values,
13981                     UU_DEFAULT);
13982                 if (walk == NULL)
13983                     uu_die(gettext("Could not walk list"));
13984
13985                 for (sp = uu_list_walk_next(walk); sp != NULL;
13986                     sp = uu_list_walk_next(walk)) {
13987                     v = string_to_value(sp->str, ty, req_quotes);
13988
13989                     if (v == NULL) {
13990                         scf_entry_destroy_children(e);
13991                         goto fail;
13992                     }
13993
13994                     ret = scf_entry_add_value(e, v);
13995                     assert(ret == SCF_SUCCESS);
13996                 }
13997                 uu_list_walk_end(walk);
13998             }
13999             result = scf_transaction_commit(tx);
14000
14001             scf_transaction_reset(tx);
14002             scf_entry_destroy_children(e);
14003         } while (result == 0);
14004
14005         if (result < 0) {
14006             if (scf_error() != SCF_ERROR_PERMISSION_DENIED)
14007                 scfdie();
14008
14009             semerr(msg_permission_denied);
14010             goto fail;
14011         }
14012
14013         ret = 0;
14014
14015         private_refresh();
14016
14017         goto cleanup;
14018
14019 fail:
14020     ret = -1;
14021
14022 cleanup:
14023     scf_transaction_destroy(tx);
14024     scf_entry_destroy(e);
14025     scf_service_destroy(svc);
14026     scf_pg_destroy(parent_pg);
14027     scf_pg_destroy(pg);
14028     scf_property_destroy(parent_prop);
14029     scf_property_destroy(prop);
14030     scf_tmpl_pg_destroy(pgt);
14031     scf_tmpl_prop_destroy(prt);

```



```

14033     return (ret);
14034 }

14036 void
14037 lscf_delprop(char *pgn)
14038 {
14039     char *slash, *pn;
14040     scf_propertygroup_t *pg;
14041     scf_transaction_t *tx;
14042     scf_transaction_entry_t *e;
14043     int ret;

14046     lscf_prep_hdl();

14048     if (cur_snap != NULL) {
14049         semerr(msg_cant_modify_snapshots);
14050         return;
14051     }

14053     if (cur_inst == NULL && cur_svc == NULL) {
14054         semerr(msg_entity_not_selected);
14055         return;
14056     }

14058     pg = scf_pg_create(g_hdl);
14059     if (pg == NULL)
14060         scfdie();

14062     slash = strchr(pgn, '/');
14063     if (slash == NULL) {
14064         pn = NULL;
14065     } else {
14066         *slash = '\0';
14067         pn = slash + 1;
14068     }

14070     if (cur_inst != NULL)
14071         ret = scf_instance_get_pg(cur_inst, pgn, pg);
14072     else
14073         ret = scf_service_get_pg(cur_svc, pgn, pg);
14074     if (ret != SCF_SUCCESS) {
14075         switch (scf_error()) {
14076             case SCF_ERROR_NOT_FOUND:
14077                 semerr(msg_no_such_pg, pgn);
14078                 break;

14080             case SCF_ERROR_INVALID_ARGUMENT:
14081                 semerr(msg_invalid_pg_name, pgn);
14082                 break;

14084             default:
14085                 scfdie();
14086         }

14088         scf_pg_destroy(pg);

14090     }
14091     return;

14093     if (pn == NULL) {
14094         /* Try to delete the property group. */
14095         if (scf_pg_delete(pg) != SCF_SUCCESS) {
14096             if (scf_error() != SCF_ERROR_PERMISSION_DENIED)
14097                 scfdie();

```

```

14099         semerr(msg_permission_denied);
14100     } else {
14101         private_refresh();
14102     }

14104     scf_pg_destroy(pg);
14105     return;
14106 }

14108     e = scf_entry_create(g_hdl);
14109     tx = scf_transaction_create(g_hdl);

14111     do {
14112         if (scf_pg_update(pg) == -1)
14113             scfdie();
14114         if (scf_transaction_start(tx, pg) != SCF_SUCCESS) {
14115             if (scf_error() != SCF_ERROR_PERMISSION_DENIED)
14116                 scfdie();

14118             semerr(msg_permission_denied);
14119             break;
14120         }

14122         if (scf_transaction_property_delete(tx, e, pn) != SCF_SUCCESS) {
14123             if (scf_error() == SCF_ERROR_NOT_FOUND) {
14124                 semerr(gettext("No such property %s/%s.\n"),
14125                     pgn, pn);
14126                 break;
14127             } else if (scf_error() == SCF_ERROR_INVALID_ARGUMENT) {
14128                 semerr(msg_invalid_prop_name, pn);
14129                 break;
14130             } else {
14131                 scfdie();
14132             }
14133         }

14135         ret = scf_transaction_commit(tx);

14137         if (ret == 0)
14138             scf_transaction_reset(tx);
14139     } while (ret == 0);

14141     if (ret < 0) {
14142         if (scf_error() != SCF_ERROR_PERMISSION_DENIED)
14143             scfdie();

14145         semerr(msg_permission_denied);
14146     } else {
14147         private_refresh();
14148     }

14150     scf_transaction_destroy(tx);
14151     scf_entry_destroy(e);
14152     scf_pg_destroy(pg);
14153 }

14155 /*
14156  * Property editing.
14157  */

14159 static int
14160 write_edit_script(FILE *strm)
14161 {
14162     char *fmribuf;
14163     ssize_t fmrlen;

```

```

14165     scf_propertygroup_t *pg;
14166     scf_property_t *prop;
14167     scf_value_t *val;
14168     scf_type_t ty;
14169     int ret, result = 0;
14170     scf_iter_t *iter, *piter, *viter;
14171     char *buf, *tybuf, *pname;
14172     const char *emsg_write_error;

14175     emsg_write_error = gettext("Error writing temporary file: %s.\n");

14178     /* select fmri */
14179     if (cur_inst != NULL) {
14180         fmrlen = scf_instance_to_fmri(cur_inst, NULL, 0);
14181         if (fmrlen < 0)
14182             scfdie();
14183         fmribuf = safe_malloc(fmrlen + 1);
14184         if (scf_instance_to_fmri(cur_inst, fmribuf, fmrlen + 1) < 0)
14185             scfdie();
14186     } else {
14187         assert(cur_svc != NULL);
14188         fmrlen = scf_service_to_fmri(cur_svc, NULL, 0);
14189         if (fmrlen < 0)
14190             scfdie();
14191         fmribuf = safe_malloc(fmrlen + 1);
14192         if (scf_service_to_fmri(cur_svc, fmribuf, fmrlen + 1) < 0)
14193             scfdie();
14194     }

14196     if (fprintf(strm, "select %s\n\n", fmribuf) < 0) {
14197         warn(emsg_write_error, strerror(errno));
14198         free(fmribuf);
14199         return (-1);
14200     }

14202     free(fmribuf);

14205     if ((pg = scf_pg_create(g_hndl)) == NULL ||
14206         (prop = scf_property_create(g_hndl)) == NULL ||
14207         (val = scf_value_create(g_hndl)) == NULL ||
14208         (iter = scf_iter_create(g_hndl)) == NULL ||
14209         (piter = scf_iter_create(g_hndl)) == NULL ||
14210         (viter = scf_iter_create(g_hndl)) == NULL)
14211         scfdie();

14213     buf = safe_malloc(max_scf_name_len + 1);
14214     tybuf = safe_malloc(max_scf_pg_type_len + 1);
14215     pname = safe_malloc(max_scf_name_len + 1);

14217     if (cur_inst != NULL)
14218         ret = scf_iter_instance_pgs(iter, cur_inst);
14219     else
14220         ret = scf_iter_service_pgs(iter, cur_svc);
14221     if (ret != SCF_SUCCESS)
14222         scfdie();

14224     while ((ret = scf_iter_next_pg(iter, pg)) == 1) {
14225         int ret2;

14227         /*
14228          * # delprop pg
14229          * # addpg pg type
14230          */

```

```

14231         if (scf_pg_get_name(pg, buf, max_scf_name_len + 1) < 0)
14232             scfdie();

14234         if (scf_pg_get_type(pg, tybuf, max_scf_pg_type_len + 1) < 0)
14235             scfdie();

14237         if (fprintf(strm, "# Property group \"%s\"\n"
14238                     "# delprop %s\n"
14239                     "# addpg %s %s\n", buf, buf, buf, tybuf) < 0) {
14240             warn(emsg_write_error, strerror(errno));
14241             result = -1;
14242             goto out;
14243         }

14245         /* # setprop pg/prop = (values) */

14247         if (scf_iter_pg_properties(piter, pg) != SCF_SUCCESS)
14248             scfdie();

14250         while ((ret2 = scf_iter_next_property(piter, prop)) == 1) {
14251             int first = 1;
14252             int ret3;
14253             int multiple;
14254             int is_str;
14255             scf_type_t bty;

14257             if (scf_property_get_name(prop, pname,
14258                                         max_scf_name_len + 1) < 0)
14259                 scfdie();

14261             if (scf_property_type(prop, &ty) != 0)
14262                 scfdie();

14264             multiple = prop_has_multiple_values(prop, val);

14266             if (fprintf(strm, "# setprop %s/%s = %s: %s", buf,
14267                         pname, scf_type_to_string(ty), multiple ? "(" : "" )
14268                 < 0) {
14269                 warn(emsg_write_error, strerror(errno));
14270                 result = -1;
14271                 goto out;
14272             }

14274             (void) scf_type_base_type(ty, &bty);
14275             is_str = (bty == SCF_TYPE_ASTRING);

14277             if (scf_iter_property_values(viter, prop) !=
14278                 SCF_SUCCESS)
14279                 scfdie();

14281             while ((ret3 = scf_iter_next_value(viter, val)) == 1) {
14282                 char *buf;
14283                 ssize_t buflen;

14285                 buflen = scf_value_get_as_string(val, NULL, 0);
14286                 if (buflen < 0)
14287                     scfdie();

14289                 buf = safe_malloc(buflen + 1);

14291                 if (scf_value_get_as_string(val, buf,
14292                                             buflen + 1) < 0)
14293                     scfdie();

14295                 if (first)
14296                     first = 0;

```

```

14297     else {
14298         if (putc(' ', strm) != ' ') {
14299             warn(msg_write_error,
14300                 strerror(errno));
14301             result = -1;
14302             goto out;
14303         }
14304     }
14305
14306     if ((is_str && multiple) ||
14307         strpbrk(buf, CHARS_TO_QUOTE) != NULL) {
14308         (void) putc('"', strm);
14309         (void) quote_and_print(buf, strm, 1);
14310         (void) putc('"', strm);
14311
14312         if (ferror(strm)) {
14313             warn(msg_write_error,
14314                 strerror(errno));
14315             result = -1;
14316             goto out;
14317         }
14318     } else {
14319         if (fprintf(strm, "%s", buf) < 0) {
14320             warn(msg_write_error,
14321                 strerror(errno));
14322             result = -1;
14323             goto out;
14324         }
14325     }
14326
14327     free(buf);
14328 }
14329 if (ret3 < 0 &&
14330     scf_error() != SCF_ERROR_PERMISSION_DENIED)
14331     scfdie();
14332
14333 /* Write closing paren if mult-value property */
14334 if ((multiple && putc(')', strm) == EOF) ||
14335
14336     /* Write final newline */
14337     fputc('\n', strm) == EOF) {
14338     warn(msg_write_error, strerror(errno));
14339     result = -1;
14340     goto out;
14341 }
14342 if (ret2 < 0)
14343     scfdie();
14344
14345 if (fputc('\n', strm) == EOF) {
14346     warn(msg_write_error, strerror(errno));
14347     result = -1;
14348     goto out;
14349 }
14350 }
14351 if (ret < 0)
14352     scfdie();
14353
14354 out:
14355 free(pname);
14356 free(tybuf);
14357 free(buf);
14358 scf_iter_destroy(viter);
14359 scf_iter_destroy(piter);
14360 scf_iter_destroy(iter);
14361 scf_value_destroy(val);

```

```

14363     scf_property_destroy(prop);
14364     scf_pg_destroy(pg);
14365
14366     if (result == 0) {
14367         if (fflush(strm) != 0) {
14368             warn(msg_write_error, strerror(errno));
14369             return (-1);
14370         }
14371     }
14372
14373     return (result);
14374 }
14375
14376 int
14377 lscf_editprop()
14378 {
14379     char *buf, *editor;
14380     size_t bufsz;
14381     int tmpfd;
14382     char tempname[] = TEMP_FILE_PATTERN;
14383
14384     lscf_prep_hdl();
14385
14386     if (cur_snap != NULL) {
14387         semerr(msg_cant_modify_snapshots);
14388         return (-1);
14389     }
14390
14391     if (cur_svc == NULL && cur_inst == NULL) {
14392         semerr(msg_entity_not_selected);
14393         return (-1);
14394     }
14395
14396     tmpfd = mkstemp(tempname);
14397     if (tmpfd == -1) {
14398         semerr(gettext("Could not create temporary file.\n"));
14399         return (-1);
14400     }
14401
14402     (void) strcpy(tempfilename, tempname);
14403
14404     tempfile = fdopen(tmpfd, "r+");
14405     if (tempfile == NULL) {
14406         warn(gettext("Could not create temporary file.\n"));
14407         if (close(tmpfd) == -1)
14408             warn(gettext("Could not close temporary file: %s.\n"),
14409                 strerror(errno));
14410     }
14411
14412     remove_tempfile();
14413
14414     return (-1);
14415 }
14416
14417 if (write_edit_script(tempfile) == -1) {
14418     remove_tempfile();
14419     return (-1);
14420 }
14421
14422 editor = getenv("EDITOR");
14423 if (editor == NULL)
14424     editor = "vi";
14425
14426 bufsz = strlen(editor) + 1 + strlen(tempname) + 1;
14427 buf = safe_malloc(bufsz);
14428
14429 if (snprintf(buf, bufsz, "%s %s", editor, tempname) < 0)

```

```

14429         uu_die(gettext("Error creating editor command"));
14431
14431     if (system(buf) == -1) {
14432         semerr(gettext("Could not launch editor %s: %s\n"), editor,
14433             strerror(errno));
14434         free(buf);
14435         remove_tempfile();
14436         return (-1);
14437     }
14439
14439     free(buf);
14441
14441     (void) engine_source(tempname, est->sc_cmd_flags & SC_CMD_IACTIVE);
14443
14443     remove_tempfile();
14445
14445     return (0);
14446 }
14448
14448 static void
14449 add_string(uu_list_t *strlist, const char *str)
14450 {
14451     string_list_t *elem;
14452     elem = safe_malloc(sizeof (*elem));
14453     uu_list_node_init(elem, &elem->node, string_pool);
14454     elem->str = safe_strdup(str);
14455     if (uu_list_append(strlist, elem) != 0)
14456         uu_die(gettext("libuutil error: %s\n"),
14457             uu_strerror(uu_error()));
14458 }
14460
14460 static int
14461 remove_string(uu_list_t *strlist, const char *str)
14462 {
14463     uu_list_walk_t *elems;
14464     string_list_t *sp;
14466
14466     /*
14467      * Find the element that needs to be removed.
14468      */
14469     elems = uu_list_walk_start(strlist, UU_DEFAULT);
14470     while ((sp = uu_list_walk_next(elems)) != NULL) {
14471         if (strcmp(sp->str, str) == 0)
14472             break;
14473     }
14474     uu_list_walk_end(elems);
14476
14476     /*
14477      * Returning 1 here as the value was not found, this
14478      * might not be an error. Leave it to the caller to
14479      * decide.
14480      */
14481     if (sp == NULL) {
14482         return (1);
14483     }
14485
14485     uu_list_remove(strlist, sp);
14487
14487     free(sp->str);
14488     free(sp);
14490
14490     return (0);
14491 }
14493 /*
14494 * Get all property values that don't match the given glob pattern,

```

```

14495     * if a pattern is specified.
14496     */
14497     static void
14498     get_prop_values(scf_property_t *prop, uu_list_t *values,
14499         const char *pattern)
14500     {
14501         scf_iter_t *iter;
14502         scf_value_t *val;
14503         int ret;
14505
14505         if ((iter = scf_iter_create(g_hndl)) == NULL ||
14506             (val = scf_value_create(g_hndl)) == NULL)
14507             scfdie();
14509
14509         if (scf_iter_property_values(iter, prop) != 0)
14510             scfdie();
14512
14512         while ((ret = scf_iter_next_value(iter, val)) == 1) {
14513             char *buf;
14514             ssize_t vlen, szret;
14516
14516             vlen = scf_value_get_as_string(val, NULL, 0);
14517             if (vlen < 0)
14518                 scfdie();
14520
14520             buf = safe_malloc(vlen + 1);
14522
14522             szret = scf_value_get_as_string(val, buf, vlen + 1);
14523             if (szret < 0)
14524                 scfdie();
14525             assert(szret <= vlen);
14527
14527             if (pattern == NULL || fnmatch(pattern, buf, 0) != 0)
14528                 add_string(values, buf);
14530
14530             free(buf);
14531         }
14533
14533         if (ret == -1)
14534             scfdie();
14536
14536         scf_value_destroy(val);
14537         scf_iter_destroy(iter);
14538     }
14540
14540 static int
14541 lscf_setpropvalue(const char *pgname, const char *type,
14542     const char *arg, int isadd, int isnotfoundok)
14543 {
14544     scf_type_t ty;
14545     scf_propertygroup_t *pg;
14546     scf_property_t *prop;
14547     int ret, result = 0;
14548     scf_transaction_t *tx;
14549     scf_transaction_entry_t *e;
14550     scf_value_t *v;
14551     string_list_t *sp;
14552     char *propname;
14553     uu_list_t *values;
14554     uu_list_walk_t *walk;
14555     void *cookie = NULL;
14556     char *pattern = NULL;
14558
14558     lscf_prep_hndl();
14560
14560     if ((values = uu_list_create(string_pool, NULL, 0)) == NULL)

```

```

14561     uu_die(gettext("Could not create property list: %s\n"),
14562           uu_strerror(uu_error()));

14564     if (!isadd)
14565         pattern = safe_strdup(arg);

14567     if ((e = scf_entry_create(g_hndl)) == NULL ||
14568         (pg = scf_pg_create(g_hndl)) == NULL ||
14569         (prop = scf_property_create(g_hndl)) == NULL ||
14570         (tx = scf_transaction_create(g_hndl)) == NULL)
14571         scfdie();

14573     if (cur_snap != NULL) {
14574         semerr(msg_cant_modify_snapshots);
14575         goto fail;
14576     }

14578     if (cur_inst == NULL && cur_svc == NULL) {
14579         semerr(msg_entity_not_selected);
14580         goto fail;
14581     }

14583     propname = strchr(pgname, '/');
14584     if (propname == NULL) {
14585         semerr(gettext("Property names must contain a '/'.\n"));
14586         goto fail;
14587     }

14589     *propname = '\0';
14590     ++propname;

14592     if (type != NULL) {
14593         ty = string_to_type(type);
14594         if (ty == SCF_TYPE_INVALID) {
14595             semerr(gettext("Unknown type \"%s\".\n"), type);
14596             goto fail;
14597         }
14598     }

14600     if (cur_inst != NULL)
14601         ret = scf_instance_get_pg(cur_inst, pgname, pg);
14602     else
14603         ret = scf_service_get_pg(cur_svc, pgname, pg);
14604     if (ret != 0) {
14605         switch (scf_error()) {
14606             case SCF_ERROR_NOT_FOUND:
14607                 if (isnotfoundok) {
14608                     result = 0;
14609                 } else {
14610                     semerr(msg_no_such_pg, pgname);
14611                     result = -1;
14612                 }
14613                 goto out;

14615             case SCF_ERROR_INVALID_ARGUMENT:
14616                 semerr(msg_invalid_pg_name, pgname);
14617                 goto fail;

14619             default:
14620                 scfdie();
14621         }
14622     }

14624     do {
14625         if (scf_pg_update(pg) == -1)
14626             scfdie();

```

```

14627         if (scf_transaction_start(tx, pg) != 0) {
14628             if (scf_error() != SCF_ERROR_PERMISSION_DENIED)
14629                 scfdie();

14631         semerr(msg_permission_denied);
14632         goto fail;
14633     }

14635     ret = scf_pg_get_property(pg, propname, prop);
14636     if (ret == 0) {
14637         scf_type_t ptype;
14638         char *pat = pattern;

14640         if (scf_property_type(prop, &ptype) != 0)
14641             scfdie();

14643         if (isadd) {
14644             if (type != NULL && ptype != ty) {
14645                 semerr(gettext("Property \"%s\" is not "
14646                               "of type \"%s\".\n"), propname,
14647                           type);
14648                 goto fail;
14649             }

14651             pat = NULL;
14652         } else {
14653             size_t len = strlen(pat);
14654             if (len > 0 && pat[len - 1] == '\\')
14655                 pat[len - 1] = '\0';
14656             if (len > 0 && pat[0] == '\\')
14657                 pat++;
14658         }

14660         ty = ptype;

14662         get_prop_values(prop, values, pat);

14664         if (isadd)
14665             add_string(values, arg);

14667         if (scf_transaction_property_change(tx, e,
14668                                             propname, ty) == -1)
14669             scfdie();
14670     } else if (scf_error() == SCF_ERROR_NOT_FOUND) {
14671         if (isadd) {
14672             if (type == NULL) {
14673                 semerr(gettext("Type required "
14674                               "for new properties.\n"));
14675                 goto fail;
14676             }

14678             add_string(values, arg);

14680             if (scf_transaction_property_new(tx, e,
14681                                             propname, ty) == -1)
14682                 scfdie();
14683         } else if (isnotfoundok) {
14684             result = 0;
14685             goto out;
14686         } else {
14687             semerr(gettext("No such property %s/%s.\n"),
14688                     pgname, propname);
14689             result = -1;
14690             goto out;
14691         }
14692     } else if (scf_error() == SCF_ERROR_INVALID_ARGUMENT) {

```

```

14693         semerr(msg_invalid_prop_name, propname);
14694         goto fail;
14695     } else {
14696         scfdie();
14697     }

14699     walk = uu_list_walk_start(values, UU_DEFAULT);
14700     if (walk == NULL)
14701         uu_die(gettext("Could not walk property list.\n"));

14703     for (sp = uu_list_walk_next(walk); sp != NULL;
14704          sp = uu_list_walk_next(walk)) {
14705         v = string_to_value(sp->str, ty, 0);

14707         if (v == NULL) {
14708             scf_entry_destroy_children(e);
14709             goto fail;
14710         }
14711         ret = scf_entry_add_value(e, v);
14712         assert(ret == 0);
14713     }
14714     uu_list_walk_end(walk);

14716     result = scf_transaction_commit(tx);

14718     scf_transaction_reset(tx);
14719     scf_entry_destroy_children(e);
14720 } while (result == 0);

14722 if (result < 0) {
14723     if (scf_error() != SCF_ERROR_PERMISSION_DENIED)
14724         scfdie();

14726     semerr(msg_permission_denied);
14727     goto fail;
14728 }

14730 result = 0;

14732 private_refresh();

14734 out:
14735     scf_transaction_destroy(tx);
14736     scf_entry_destroy(e);
14737     scf_pg_destroy(pg);
14738     scf_property_destroy(prop);
14739     free(pattern);

14741     while ((sp = uu_list_takedown(values, &cookie)) != NULL) {
14742         free(sp->str);
14743         free(sp);
14744     }

14746     uu_list_destroy(values);

14748     return (result);

14750 fail:
14751     result = -1;
14752     goto out;
14753 }

14755 int
14756 lscf_addpropvalue(const char *pgname, const char *type, const char *value)
14757 {
14758     return (lscf_setpropvalue(pgname, type, value, 1, 0));

```

```

14759 }

14761 int
14762 lscf_delpropvalue(const char *pgname, const char *pattern, int isnotfoundok)
14763 {
14764     return (lscf_setpropvalue(pgname, NULL, pattern, 0, isnotfoundok));
14765 }

14767 /*
14768  * Look for a standard start method, first in the instance (if any),
14769  * then the service.
14770  */
14771 static const char *
14772 start_method_name(int *in_instance)
14773 {
14774     scf_propertygroup_t *pg;
14775     char **p;
14776     int ret;
14777     scf_instance_t *inst = cur_inst;

14779     if ((pg = scf_pg_create(g_hndl)) == NULL)
14780         scfdie();

14782 again:
14783     for (p = start_method_names; *p != NULL; p++) {
14784         if (inst != NULL)
14785             ret = scf_instance_get_pg(inst, *p, pg);
14786         else
14787             ret = scf_service_get_pg(cur_svc, *p, pg);

14789         if (ret == 0) {
14790             size_t bufsz = strlen(SCF_GROUP_METHOD) + 1;
14791             char *buf = safe_malloc(bufsz);

14793             if ((ret = scf_pg_get_type(pg, buf, bufsz)) < 0) {
14794                 free(buf);
14795                 continue;
14796             }
14797             if (strcmp(buf, SCF_GROUP_METHOD) != 0) {
14798                 free(buf);
14799                 continue;
14800             }

14802             free(buf);
14803             *in_instance = (inst != NULL);
14804             scf_pg_destroy(pg);
14805             return (*p);
14806         }

14808         if (scf_error() == SCF_ERROR_NOT_FOUND)
14809             continue;

14811         scfdie();
14812     }

14814     if (inst != NULL) {
14815         inst = NULL;
14816         goto again;
14817     }

14819     scf_pg_destroy(pg);
14820     return (NULL);
14821 }

14823 static int
14824 addpg(const char *name, const char *type)

```

```

14825 {
14826     scf_propertygroup_t *pg;
14827     int ret;

14829     pg = scf_pg_create(g_hndl);
14830     if (pg == NULL)
14831         scfdie();

14833     if (cur_inst != NULL)
14834         ret = scf_instance_add_pg(cur_inst, name, type, 0, pg);
14835     else
14836         ret = scf_service_add_pg(cur_svc, name, type, 0, pg);

14838     if (ret != 0) {
14839         switch (scf_error()) {
14840             case SCF_ERROR_EXISTS:
14841                 ret = 0;
14842                 break;

14844             case SCF_ERROR_PERMISSION_DENIED:
14845                 semerr(msg_permission_denied);
14846                 break;

14848             default:
14849                 scfdie();
14850         }
14851     }

14853     scf_pg_destroy(pg);
14854     return (ret);
14855 }

14857 int
14858 lscf_setenv(uu_list_t *args, int isunset)
14859 {
14860     int ret = 0;
14861     size_t i;
14862     int argc;
14863     char **argv = NULL;
14864     string_list_t *slp;
14865     char *pattern;
14866     char *prop;
14867     int do_service = 0;
14868     int do_instance = 0;
14869     const char *method = NULL;
14870     const char *name = NULL;
14871     const char *value = NULL;
14872     scf_instance_t *saved_cur_inst = cur_inst;

14874     lscf_prep_hndl();

14876     argc = uu_list_numnodes(args);
14877     if (argc < 1)
14878         goto usage;

14880     argv = calloc(argc + 1, sizeof (char *));
14881     if (argv == NULL)
14882         uu_die(gettext("Out of memory.\n"));

14884     for (slp = uu_list_first(args), i = 0;
14885         slp != NULL;
14886         slp = uu_list_next(args, slp), ++i)
14887         argv[i] = slp->str;

14889     argv[i] = NULL;

```

```

14891     opterr = 0;
14892     optind = 0;
14893     for (;;) {
14894         ret = getopt(argc, argv, "sim:");
14895         if (ret == -1)
14896             break;

14898         switch (ret) {
14899             case 's':
14900                 do_service = 1;
14901                 cur_inst = NULL;
14902                 break;

14904             case 'i':
14905                 do_instance = 1;
14906                 break;

14908             case 'm':
14909                 method = optarg;
14910                 break;

14912             case '?':
14913                 goto usage;

14915             default:
14916                 bad_error("getopt", ret);
14917         }
14918     }

14920     argc -= optind;
14921     if ((do_service && do_instance) ||
14922         (isunset && argc != 1) ||
14923         (!isunset && argc != 2))
14924         goto usage;

14926     name = argv[optind];
14927     if (!isunset)
14928         value = argv[optind + 1];

14930     if (cur_snap != NULL) {
14931         semerr(msg_cant_modify_snapshots);
14932         ret = -1;
14933         goto out;
14934     }

14936     if (cur_inst == NULL && cur_svc == NULL) {
14937         semerr(msg_entity_not_selected);
14938         ret = -1;
14939         goto out;
14940     }

14942     if (do_instance && cur_inst == NULL) {
14943         semerr(gettext("No instance is selected.\n"));
14944         ret = -1;
14945         goto out;
14946     }

14948     if (do_service && cur_svc == NULL) {
14949         semerr(gettext("No service is selected.\n"));
14950         ret = -1;
14951         goto out;
14952     }

14954     if (method == NULL) {
14955         if (do_instance || do_service) {
14956             method = "method_context";

```

```

14957         if (!isunset) {
14958             ret = addpg("method_context",
14959                 SCF_GROUP_FRAMEWORK);
14960             if (ret != 0)
14961                 goto out;
14962         }
14963     } else {
14964         int in_instance;
14965         method = start_method_name(&in_instance);
14966         if (method == NULL) {
14967             semerr(gettext(
14968                 "Couldn't find start method; please "
14969                 "specify a method with '-m'.\n"));
14970             ret = -1;
14971             goto out;
14972         }
14973         if (!in_instance)
14974             cur_inst = NULL;
14975     }
14976 } else {
14977     scf_propertygroup_t *pg;
14978     size_t bufisz;
14979     char *buf;
14980     int ret;
14981
14982     if ((pg = scf_pg_create(g_hndl)) == NULL)
14983         scfdie();
14984
14985     if (cur_inst != NULL)
14986         ret = scf_instance_get_pg(cur_inst, method, pg);
14987     else
14988         ret = scf_service_get_pg(cur_svc, method, pg);
14989
14990     if (ret != 0) {
14991         scf_pg_destroy(pg);
14992         switch (scf_error()) {
14993             case SCF_ERROR_NOT_FOUND:
14994                 semerr(gettext("Couldn't find the method "
14995                     "\"%s\".\n"), method);
14996                 goto out;
14997
14998             case SCF_ERROR_INVALID_ARGUMENT:
14999                 semerr(gettext("Invalid method name \"%s\".\n"),
15000                     method);
15001                 goto out;
15002
15003             default:
15004                 scfdie();
15005         }
15006     }
15007
15008     bufisz = strlen(SCF_GROUP_METHOD) + 1;
15009     buf = safe_malloc(bufisz);
15010
15011     if (scf_pg_get_type(pg, buf, bufisz) < 0 ||
15012         strcmp(buf, SCF_GROUP_METHOD) != 0) {
15013         semerr(gettext("Property group \"%s\" is not of type "
15014             "\"method\".\n"), method);
15015         ret = -1;
15016         free(buf);
15017         scf_pg_destroy(pg);
15018         goto out;
15019     }
15020
15021     free(buf);
15022     scf_pg_destroy(pg);

```

```

15023     }
15024
15025     prop = uu_msprintf("%s/environment", method);
15026     pattern = uu_msprintf("%s=*", name);
15027
15028     if (prop == NULL || pattern == NULL)
15029         uu_die(gettext("Out of memory.\n"));
15030
15031     ret = lscf_delpropvalue(prop, pattern, !isunset);
15032
15033     if (ret == 0 && !isunset) {
15034         uu_free(pattern);
15035         uu_free(prop);
15036         prop = uu_msprintf("%s/environment", method);
15037         pattern = uu_msprintf("%s=%s", name, value);
15038         if (prop == NULL || pattern == NULL)
15039             uu_die(gettext("Out of memory.\n"));
15040         ret = lscf_addpropvalue(prop, "astring:", pattern);
15041     }
15042     uu_free(pattern);
15043     uu_free(prop);
15044
15045 out:
15046     cur_inst = saved_cur_inst;
15047
15048     free(argv);
15049     return (ret);
15050 usage:
15051     ret = -2;
15052     goto out;
15053 }
15054
15055 /*
15056  * Snapshot commands
15057  */
15058
15059 void
15060 lscf_listsnap()
15061 {
15062     scf_snapshot_t *snap;
15063     scf_iter_t *iter;
15064     char *nb;
15065     int r;
15066
15067     lscf_prep_hndl();
15068
15069     if (cur_inst == NULL) {
15070         semerr(gettext("Instance not selected.\n"));
15071         return;
15072     }
15073
15074     if ((snap = scf_snapshot_create(g_hndl)) == NULL ||
15075         (iter = scf_iter_create(g_hndl)) == NULL)
15076         scfdie();
15077
15078     if (scf_iter_instance_snapshots(iter, cur_inst) != SCF_SUCCESS)
15079         scfdie();
15080
15081     nb = safe_malloc(max_scf_name_len + 1);
15082
15083     while ((r = scf_iter_next_snapshot(iter, snap)) == 1) {
15084         if (scf_snapshot_get_name(snap, nb, max_scf_name_len + 1) < 0)
15085             scfdie();
15086
15087         (void) puts(nb);
15088     }

```



```

15089     if (r < 0)
15090         scfdie();

15092     free(nb);
15093     scf_iter_destroy(iter);
15094     scf_snapshot_destroy(snap);
15095 }

15097 void
15098 lscf_selectsnap(const char *name)
15099 {
15100     scf_snapshot_t *snap;
15101     scf_snaplevel_t *level;

15103     lscf_prep_hdl();

15105     if (cur_inst == NULL) {
15106         semerr(gettext("Instance not selected.\n"));
15107         return;
15108     }

15110     if (cur_snap != NULL) {
15111         if (name != NULL) {
15112             char *cur_snap_name;
15113             boolean_t nochange;

15115             cur_snap_name = safe_malloc(max_scf_name_len + 1);

15117             if (scf_snapshot_get_name(cur_snap, cur_snap_name,
15118                 max_scf_name_len + 1) < 0)
15119                 scfdie();

15121             nochange = strcmp(name, cur_snap_name) == 0;

15123             free(cur_snap_name);

15125             if (nochange)
15126                 return;
15127         }

15129         unselect_cursnap();
15130     }

15132     if (name == NULL)
15133         return;

15135     if ((snap = scf_snapshot_create(g_hdl)) == NULL ||
15136         (level = scf_snaplevel_create(g_hdl)) == NULL)
15137         scfdie();

15139     if (scf_instance_get_snapshot(cur_inst, name, snap) !=
15140         SCF_SUCCESS) {
15141         switch (scf_error()) {
15142             case SCF_ERROR_INVALID_ARGUMENT:
15143                 semerr(gettext("Invalid name \"%s\".\n"), name);
15144                 break;

15146             case SCF_ERROR_NOT_FOUND:
15147                 semerr(gettext("No such snapshot \"%s\".\n"), name);
15148                 break;

15150             default:
15151                 scfdie();
15152         }

15154         scf_snaplevel_destroy(level);

```

```

15155         scf_snapshot_destroy(snap);
15156         return;
15157     }

15159     /* Load the snaplevels into our list. */
15160     cur_levels = uu_list_create(snaplevel_pool, NULL, 0);
15161     if (cur_levels == NULL)
15162         uu_die(gettext("Could not create list: %s\n"),
15163             uu_strerror(uu_error()));

15165     if (scf_snapshot_get_base_snaplevel(snap, level) != SCF_SUCCESS) {
15166         if (scf_error() != SCF_ERROR_NOT_FOUND)
15167             scfdie();

15169         semerr(gettext("Snapshot has no snaplevels.\n"));

15171         scf_snaplevel_destroy(level);
15172         scf_snapshot_destroy(snap);
15173         return;
15174     }

15176     cur_snap = snap;

15178     for (;;) {
15179         cur_elt = safe_malloc(sizeof(*cur_elt));
15180         uu_list_node_init(cur_elt, &cur_elt->list_node,
15181             snaplevel_pool);
15182         cur_elt->sl = level;
15183         if (uu_list_insert_after(cur_levels, NULL, cur_elt) != 0)
15184             uu_die(gettext("libutil error: %s\n"),
15185                 uu_strerror(uu_error()));

15187         level = scf_snaplevel_create(g_hdl);
15188         if (level == NULL)
15189             scfdie();

15191         if (scf_snaplevel_get_next_snaplevel(cur_elt->sl,
15192             level) != SCF_SUCCESS) {
15193             if (scf_error() != SCF_ERROR_NOT_FOUND)
15194                 scfdie();

15196             scf_snaplevel_destroy(level);
15197             break;
15198         }
15199     }

15201     cur_elt = uu_list_last(cur_levels);
15202     cur_level = cur_elt->sl;
15203 }

15205 /*
15206  * Copies the properties & values in src to dst. Assumes src won't change.
15207  * Returns -1 if permission is denied, -2 if another transaction interrupts,
15208  * and 0 on success.
15209  *
15210  * If enabled is 0 or 1, its value is used for the SCF_PROPERTY_ENABLED
15211  * property, if it is copied and has type boolean. (See comment in
15212  * lscf_revert()).
15213  */
15214 static int
15215 pg_copy(const scf_propertygroup_t *src, scf_propertygroup_t *dst,
15216     uint8_t enabled)
15217 {
15218     scf_transaction_t *tx;
15219     scf_iter_t *iter, *viter;
15220     scf_property_t *prop;

```

```

15221     scf_value_t *v;
15222     char *nbuf;
15223     int r;

15225     tx = scf_transaction_create(g_hndl);
15226     if (tx == NULL)
15227         scfdie();

15229     if (scf_transaction_start(tx, dst) != SCF_SUCCESS) {
15230         if (scf_error() != SCF_ERROR_PERMISSION_DENIED)
15231             scfdie();

15233         scf_transaction_destroy(tx);

15235         return (-1);
15236     }

15238     if ((iter = scf_iter_create(g_hndl)) == NULL ||
15239         (prop = scf_property_create(g_hndl)) == NULL ||
15240         (viter = scf_iter_create(g_hndl)) == NULL)
15241         scfdie();

15243     nbuf = safe_malloc(max_scf_name_len + 1);

15245     if (scf_iter_pg_properties(iter, src) != SCF_SUCCESS)
15246         scfdie();

15248     for (;;) {
15249         scf_transaction_entry_t *e;
15250         scf_type_t ty;

15252         r = scf_iter_next_property(iter, prop);
15253         if (r == -1)
15254             scfdie();
15255         if (r == 0)
15256             break;

15258         e = scf_entry_create(g_hndl);
15259         if (e == NULL)
15260             scfdie();

15262         if (scf_property_type(prop, &ty) != SCF_SUCCESS)
15263             scfdie();

15265         if (scf_property_get_name(prop, nbuf, max_scf_name_len + 1) < 0)
15266             scfdie();

15268         if (scf_transaction_property_new(tx, e, nbuf,
15269             ty) != SCF_SUCCESS)
15270             scfdie();

15272         if ((enabled == 0 || enabled == 1) &&
15273             strcmp(nbuf, scf_property_enabled) == 0 &&
15274             ty == SCF_TYPE_BOOLEAN) {
15275             v = scf_value_create(g_hndl);
15276             if (v == NULL)
15277                 scfdie();

15279             scf_value_set_boolean(v, enabled);

15281             if (scf_entry_add_value(e, v) != 0)
15282                 scfdie();
15283         } else {
15284             if (scf_iter_property_values(viter, prop) != 0)
15285                 scfdie();

```

```

15287         for (;;) {
15288             v = scf_value_create(g_hndl);
15289             if (v == NULL)
15290                 scfdie();

15292             r = scf_iter_next_value(viter, v);
15293             if (r == -1)
15294                 scfdie();
15295             if (r == 0) {
15296                 scf_value_destroy(v);
15297                 break;
15298             }

15300             if (scf_entry_add_value(e, v) != SCF_SUCCESS)
15301                 scfdie();
15302         }
15303     }
15304 }

15306     free(nbuf);
15307     scf_iter_destroy(viter);
15308     scf_property_destroy(prop);
15309     scf_iter_destroy(iter);

15311     r = scf_transaction_commit(tx);
15312     if (r == -1 && scf_error() != SCF_ERROR_PERMISSION_DENIED)
15313         scfdie();

15315     scf_transaction_destroy_children(tx);
15316     scf_transaction_destroy(tx);

15318     switch (r) {
15319     case 1:         return (0);
15320     case 0:         return (-2);
15321     case -1:        return (-1);

15323     default:
15324         abort();
15325     }

15327     /* NOTREACHED */
15328 }

15330 void
15331 lscf_revert(const char *snapname)
15332 {
15333     scf_snapshot_t *snap, *prev;
15334     scf_snaplevel_t *level, *nlevel;
15335     scf_iter_t *iter;
15336     scf_propertygroup_t *pg, *npg;
15337     scf_property_t *prop;
15338     scf_value_t *val;
15339     char *nbuf, *tbuf;
15340     uint8_t enabled;

15342     lscf_prep_hndl();

15344     if (cur_inst == NULL) {
15345         semerr(gettext("Instance not selected.\n"));
15346         return;
15347     }

15349     if (snapname != NULL) {
15350         snap = scf_snapshot_create(g_hndl);
15351         if (snap == NULL)
15352             scfdie();

```



```

15485         r = pg_copy(pg, npg, enabled);
15486     else
15487         r = pg_copy(pg, npg, 2);

15489     switch (r) {
15490     case 0:
15491         break;

15493     case -1:
15494         semerr(msg_permission_denied);
15495         goto out;

15497     case -2:
15498         semerr(gettext(
15499             "Interrupted by another change.\n"));
15500         goto out;

15502     default:
15503         abort();
15504     }
15505 }
15506 if (r == -1)
15507     scfdie();

15509 /* Get next level. */
15510 nlevel = scf_snaplevel_create(g_hndl);
15511 if (nlevel == NULL)
15512     scfdie();

15514 if (scf_snaplevel_get_next_snaplevel(level, nlevel) !=
15515     SCF_SUCCESS) {
15516     if (scf_error() != SCF_ERROR_NOT_FOUND)
15517         scfdie();

15519     scf_snaplevel_destroy(nlevel);
15520     break;
15521 }

15523     scf_snaplevel_destroy(level);
15524     level = nlevel;
15525 }

15527 if (snapname == NULL) {
15528     lscf_selectsnap(NULL);
15529     snap = NULL; /* cur_snap has been destroyed */
15530 }

15532 out:
15533     free(tbuf);
15534     free(nbuf);
15535     scf_value_destroy(val);
15536     scf_property_destroy(prop);
15537     scf_pg_destroy(npg);
15538     scf_pg_destroy(pg);
15539     scf_iter_destroy(iter);
15540     scf_snaplevel_destroy(level);
15541     scf_snapshot_destroy(prev);
15542     if (snap != cur_snap)
15543         scf_snapshot_destroy(snap);
15544 }

15546 void
15547 lscf_refresh(void)
15548 {
15549     ssize_t fmrlen;
15550     size_t bufsz;

```

```

15551     char *fmribuf;
15552     int r;

15554     lscf_prep_hndl();

15556     if (cur_inst == NULL) {
15557         semerr(gettext("Instance not selected.\n"));
15558         return;
15559     }

15561     bufsz = max_scf_fmri_len + 1;
15562     fmribuf = safe_malloc(bufsz);
15563     fmrlen = scf_instance_to_fmri(cur_inst, fmribuf, bufsz);
15564     if (fmrlen < 0) {
15565         free(fmribuf);
15566         if (scf_error() != SCF_ERROR_DELETED)
15567             scfdie();
15568         scf_instance_destroy(cur_inst);
15569         cur_inst = NULL;
15570         warn(msg_deleted);
15571         return;
15572     }
15573     assert(fmrlen < bufsz);

15575     r = refresh_entity(0, cur_inst, fmribuf, NULL, NULL, NULL);
15576     switch (r) {
15577     case 0:
15578         break;

15580     case ECONNABORTED:
15581         warn(gettext("Could not refresh %s "
15582             "(repository connection broken).\n"), fmribuf);
15583         break;

15585     case ECANCELED:
15586         warn(msg_deleted);
15587         break;

15589     case EPERM:
15590         warn(gettext("Could not refresh %s "
15591             "(permission denied).\n"), fmribuf);
15592         break;

15594     case ENOSPC:
15595         warn(gettext("Could not refresh %s "
15596             "(repository server out of resources).\n"),
15597             fmribuf);
15598         break;

15600     case EACCES:
15601     default:
15602         bad_error("refresh_entity", scf_error());
15603     }

15605     free(fmribuf);
15606 }

15608 /*
15609  * describe [-v] [-t] [pg/prop]
15610  */
15611 int
15612 lscf_describe(uu_list_t *args, int hasargs)
15613 {
15614     int ret = 0;
15615     size_t i;
15616     int argc;

```

```

15617     char **argv = NULL;
15618     string_list_t *slp;
15619     int do_verbose = 0;
15620     int do_templates = 0;
15621     char *pattern = NULL;
15623     lscf_prep_hdl();
15625     if (hasargs != 0) {
15626         argc = uu_list_numnodes(args);
15627         if (argc < 1)
15628             goto usage;
15630         argv = calloc(argc + 1, sizeof (char *));
15631         if (argv == NULL)
15632             uu_die(gettext("Out of memory.\n"));
15634         for (slp = uu_list_first(args), i = 0;
15635             slp != NULL;
15636             slp = uu_list_next(args, slp), ++i)
15637             argv[i] = slp->str;
15639         argv[i] = NULL;
15641         /*
15642          * We start optind = 0 because our list of arguments
15643          * starts at argv[0]
15644          */
15645         optind = 0;
15646         opterr = 0;
15647         for (;;) {
15648             ret = getopt(argc, argv, "vt");
15649             if (ret == -1)
15650                 break;
15652             switch (ret) {
15653                 case 'v':
15654                     do_verbose = 1;
15655                     break;
15657                 case 't':
15658                     do_templates = 1;
15659                     break;
15661                 case '?':
15662                     goto usage;
15664                 default:
15665                     bad_error("getopt", ret);
15666             }
15667         }
15669         pattern = argv[optind];
15670     }
15672     if (cur_inst == NULL && cur_svc == NULL) {
15673         semerr(msg_entity_not_selected);
15674         ret = -1;
15675         goto out;
15676     }
15678     /*
15679     * list_entity_tmpl(), listprop() and listtmpl() produce verbose
15680     * output if their last parameter is set to 2. Less information is
15681     * produced if the parameter is set to 1.
15682     */

```

```

15683     if (pattern == NULL) {
15684         if (do_verbose == 1)
15685             list_entity_tmpl(2);
15686         else
15687             list_entity_tmpl(1);
15688     }
15690     if (do_templates == 0) {
15691         if (do_verbose == 1)
15692             listprop(pattern, 0, 2);
15693         else
15694             listprop(pattern, 0, 1);
15695     } else {
15696         if (do_verbose == 1)
15697             listtmpl(pattern, 2);
15698         else
15699             listtmpl(pattern, 1);
15700     }
15702     ret = 0;
15703 out:
15704     if (argv != NULL)
15705         free(argv);
15706     return (ret);
15707 usage:
15708     ret = -2;
15709     goto out;
15710 }
15712 #define PARAM_ACTIVE ((const char *) "active")
15713 #define PARAM_INACTIVE ((const char *) "inactive")
15714 #define PARAM_SMTP_TO ((const char *) "to")
15716 /*
15717 * tokenize()
15718 * Breaks down the string according to the tokens passed.
15719 * Caller is responsible for freeing array of pointers returned.
15720 * Returns NULL on failure
15721 */
15722 char **
15723 tokenize(char *str, const char *sep)
15724 {
15725     char *token, *lasts;
15726     char **buf;
15727     int n = 0; /* number of elements */
15728     int size = 8; /* size of the array (initial) */
15730     buf = safe_malloc(size * sizeof (char *));
15732     for (token = strtok_r(str, sep, &lasts); token != NULL;
15733         token = strtok_r(NULL, sep, &lasts), ++n) {
15734         if (n + 1 >= size) {
15735             size *= 2;
15736             if ((buf = realloc(buf, size * sizeof (char *))) ==
15737                 NULL) {
15738                 uu_die(gettext("Out of memory"));
15739             }
15740         }
15741         buf[n] = token;
15742     }
15743     /* NULL terminate the pointer array */
15744     buf[n] = NULL;
15746     return (buf);
15747 }

```

```

15749 int32_t
15750 check_tokens(char **p)
15751 {
15752     int32_t smf = 0;
15753     int32_t fma = 0;
15754
15755     while (*p) {
15756         int32_t t = string_to_tset(*p);
15757
15758         if (t == 0) {
15759             if (is_fma_token(*p) == 0)
15760                 return (INVALID_TOKENS);
15761             fma = 1; /* this token is an fma event */
15762         } else {
15763             smf |= t;
15764         }
15765
15766         if (smf != 0 && fma == 1)
15767             return (MIXED_TOKENS);
15768         ++p;
15769     }
15770
15771     if (smf > 0)
15772         return (smf);
15773     else if (fma == 1)
15774         return (FMA_TOKENS);
15775
15776     return (INVALID_TOKENS);
15777 }
15778
15779 static int
15780 get_selection_str(char *fmri, size_t sz)
15781 {
15782     if (g_hndl == NULL) {
15783         semerr(msg_entity_not_selected);
15784         return (-1);
15785     } else if (cur_level != NULL) {
15786         semerr(msg_invalid_for_snapshot);
15787         return (-1);
15788     } else {
15789         lscf_get_selection_str(fmri, sz);
15790     }
15791
15792     return (0);
15793 }
15794
15795 void
15796 lscf_delnotify(const char *set, int global)
15797 {
15798     char *str = strdup(set);
15799     char **pgs;
15800     char **p;
15801     int32_t tset;
15802     char *fmri = NULL;
15803
15804     if (str == NULL)
15805         uu_die(gettext("Out of memory.\n"));
15806
15807     pgs = tokenize(str, ",");
15808
15809     if ((tset = check_tokens(pgs)) > 0) {
15810         size_t sz = max_scf_fmri_len + 1;
15811
15812         fmri = safe_malloc(sz);
15813         if (global) {
15814             (void) strcpy(fmri, SCF_INSTANCE_GLOBAL, sz);

```

```

15815     } else if (get_selection_str(fmri, sz) != 0) {
15816         goto out;
15817     }
15818
15819     if (smf_notify_del_params(SCF_SVC_TRANSITION_CLASS, fmri,
15820                             tset) != SCF_SUCCESS) {
15821         uu_warn(gettext("Failed smf_notify_del_params: %s\n"),
15822                scf_strerror(scf_error()));
15823     }
15824     } else if (tset == FMA_TOKENS) {
15825         if (global) {
15826             semerr(gettext("Can't use option '-g' with FMA event "
15827                            "definitions\n"));
15828             goto out;
15829         }
15830
15831         for (p = pgs; *p; ++p) {
15832             if (smf_notify_del_params(de_tag(*p), NULL, 0) !=
15833                 SCF_SUCCESS) {
15834                 uu_warn(gettext("Failed for \"%s\": %s\n"), *p,
15835                        scf_strerror(scf_error()));
15836                 goto out;
15837             }
15838         }
15839     } else if (tset == MIXED_TOKENS) {
15840         semerr(gettext("Can't mix SMF and FMA event definitions\n"));
15841         goto out;
15842     } else {
15843         uu_die(gettext("Invalid input.\n"));
15844     }
15845
15846 out:
15847     free(fmri);
15848     free(pgs);
15849     free(str);
15850 }
15851
15852 void
15853 lscf_listnotify(const char *set, int global)
15854 {
15855     char *str = safe_strdup(set);
15856     char **pgs;
15857     char **p;
15858     int32_t tset;
15859     nvlist_t *nvl;
15860     char *fmri = NULL;
15861
15862     if (nvlist_alloc(&nvl, NV_UNIQUE_NAME, 0) != 0)
15863         uu_die(gettext("Out of memory.\n"));
15864
15865     pgs = tokenize(str, ",");
15866
15867     if ((tset = check_tokens(pgs)) > 0) {
15868         size_t sz = max_scf_fmri_len + 1;
15869
15870         fmri = safe_malloc(sz);
15871         if (global) {
15872             (void) strcpy(fmri, SCF_INSTANCE_GLOBAL, sz);
15873         } else if (get_selection_str(fmri, sz) != 0) {
15874             goto out;
15875         }
15876
15877         if (_scf_get_svc_notify_params(fmri, nvl, tset, 1, 1) !=
15878             SCF_SUCCESS) {
15879             if (scf_error() != SCF_ERROR_NOT_FOUND &&
15880                 scf_error() != SCF_ERROR_DELETED)

```

```

15881         uu_warn(gettext(
15882             "Failed listnotify: %s\n"),
15883             scf_strerror(scf_error()));
15884         goto out;
15885     }
15887     listnotify_print(nvl, NULL);
15888 } else if (tset == FMA_TOKENS) {
15889     if (global) {
15890         semerr(gettext("Can't use option '-g' with FMA event "
15891             "definitions\n"));
15892         goto out;
15893     }
15895     for (p = pgs; *p; ++p) {
15896         if (_scf_get_fma_notify_params(de_tag(*p), nvl, 1) !=
15897             SCF_SUCCESS) {
15898             /*
15899              * if the preferences have just been deleted
15900              * or does not exist, just skip.
15901              */
15902             if (scf_error() == SCF_ERROR_NOT_FOUND ||
15903                 scf_error() == SCF_ERROR_DELETED)
15904                 continue;
15905             uu_warn(gettext(
15906                 "Failed listnotify: %s\n"),
15907                 scf_strerror(scf_error()));
15908             goto out;
15909         }
15910         listnotify_print(nvl, re_tag(*p));
15911     }
15912 } else if (tset == MIXED_TOKENS) {
15913     semerr(gettext("Can't mix SMF and FMA event definitions\n"));
15914     goto out;
15915 } else {
15916     semerr(gettext("Invalid input.\n"));
15917 }
15919 out:
15920     nvl_free(nvl);
15921     free(fmri);
15922     free(pgs);
15923     free(str);
15924 }
15926 static char *
15927 strip_quotes_and_blanks(char *s)
15928 {
15929     char *start = s;
15930     char *end = strchr(s, '\0');
15932     if (s[0] == '\'' && end != NULL && *(end + 1) == '\0') {
15933         start = s + 1;
15934         while (isblank(*start))
15935             start++;
15936         while (isblank(*(end - 1)) && end > start) {
15937             end--;
15938         }
15939         *end = '\0';
15940     }
15942     return (start);
15943 }
15945 static int
15946 set_active(nvlist_t *mech, const char *hier_part)

```

```

15947 {
15948     boolean_t b;
15950     if (*hier_part == '\0' || strcmp(hier_part, PARAM_ACTIVE) == 0) {
15951         b = B_TRUE;
15952     } else if (strcmp(hier_part, PARAM_INACTIVE) == 0) {
15953         b = B_FALSE;
15954     } else {
15955         return (-1);
15956     }
15958     if (nvlist_add_boolean_value(mech, PARAM_ACTIVE, b) != 0)
15959         uu_die(gettext("Out of memory.\n"));
15961     return (0);
15962 }
15964 static int
15965 add_snmp_params(nvlist_t *mech, char *hier_part)
15966 {
15967     return (set_active(mech, hier_part));
15968 }
15970 static int
15971 add_syslog_params(nvlist_t *mech, char *hier_part)
15972 {
15973     return (set_active(mech, hier_part));
15974 }
15976 /*
15977  * add_mailto_params()
15978  * parse the hier_part of mailto URI
15979  * mailto:<addr>[?<header1>=<value1>[&<header2>=<value2>]]
15980  * or mailto:{{[active]|inactive}}
15981  */
15982 static int
15983 add_mailto_params(nvlist_t *mech, char *hier_part)
15984 {
15985     const char *tok = "?&";
15986     char *p;
15987     char *lasts;
15988     char *param;
15989     char *val;
15991     /*
15992      * If the notification parameters are in the form of
15993      *
15994      * mailto:{{[active]|inactive}}
15995      *
15996      * we set the property accordingly and return.
15997      * Otherwise, we make the notification type active and
15998      * process the hier_part.
15999      */
16000     if (set_active(mech, hier_part) == 0)
16001         return (0);
16002     else if (set_active(mech, PARAM_ACTIVE) != 0)
16003         return (-1);
16005     if ((p = strtok_r(hier_part, tok, &lasts)) == NULL) {
16006         /*
16007          * sanity check: we only get here if hier_part = "", but
16008          * that's handled by set_active
16009          */
16010         uu_die("strtok_r");
16011     }

```



```

16146         if (smf_notify_set_params(SCF_SVC_TRANSITION_CLASS,
16147             nvl) != SCF_SUCCESS) {
16148             r = -1;
16149             uu_warn(gettext(
16150                 "Failed smf_notify_set_params(3SCF): %s\n"),
16151                 scf_strerror(scf_error()));
16152         }
16153     } else if (tset == FMA_TOKENS) {
16154         /* FMA event parameters */
16155         if (global) {
16156             semerr(gettext("Can't use option '-g' with FMA event "
16157                 "definitions\n"));
16158             goto out;
16159         }
16160     }
16162     if ((r = set_params(params, p)) != 0)
16163         goto out;
16165     if (nvlist_add_nvlist(nvl, SCF_NOTIFY_PARAMS, params) != 0)
16166         uu_die(gettext("Out of memory.\n"));
16168     while (*events) {
16169         if (smf_notify_set_params(de_tag(*events), nvl) !=
16170             SCF_SUCCESS)
16171             uu_warn(gettext(
16172                 "Failed smf_notify_set_params(3SCF) for "
16173                 "event %s: %s\n"), *events,
16174                 scf_strerror(scf_error()));
16175         events++;
16176     }
16177 } else if (tset == MIXED_TOKENS) {
16178     semerr(gettext("Can't mix SMF and FMA event definitions\n"));
16179 } else {
16180     /* Sanity check */
16181     uu_die(gettext("Invalid input.\n"));
16182 }
16184 out:
16185     nvlist_free(nvl);
16186     nvlist_free(params);
16187     free(fmri);
16188     free(str);
16190     return (r);
16191 }
16193 int
16194 lscf_setnotify(uu_list_t *args)
16195 {
16196     int argc;
16197     char **argv = NULL;
16198     string_list_t *slp;
16199     int global;
16200     char *events;
16201     char **p;
16202     int i;
16203     int ret;
16205     if ((argc = uu_list_numnodes(args)) < 2)
16206         goto usage;
16208     argv = calloc(argc + 1, sizeof(char *));
16209     if (argv == NULL)
16210         uu_die(gettext("Out of memory.\n"));

```

```

16212     for (slp = uu_list_first(args), i = 0;
16213         slp != NULL;
16214         slp = uu_list_next(args, slp), ++i)
16215         argv[i] = slp->str;
16217     argv[i] = NULL;
16219     if (strcmp(argv[0], "-g") == 0) {
16220         global = 1;
16221         events = argv[1];
16222         p = argv + 2;
16223     } else {
16224         global = 0;
16225         events = argv[0];
16226         p = argv + 1;
16227     }
16229     ret = setnotify(events, p, global);
16231 out:
16232     free(argv);
16233     return (ret);
16235 usage:
16236     ret = -2;
16237     goto out;
16238 }
16240 /*
16241  * Creates a list of instance name strings associated with a service. If
16242  * wohandcrafted flag is set, get only instances that have a last-import
16243  * snapshot, instances that were imported via svccfg.
16244  */
16245 static uu_list_t *
16246 create_instance_list(scf_service_t *svc, int wohandcrafted)
16247 {
16248     scf_snapshot_t *snap = NULL;
16249     scf_instance_t *inst;
16250     scf_iter_t *inst_iter;
16251     uu_list_t *instances;
16252     char *instname;
16253     int r;
16255     inst_iter = scf_iter_create(g_hdl);
16256     inst = scf_instance_create(g_hdl);
16257     if (inst_iter == NULL || inst == NULL) {
16258         uu_warn(gettext("Could not create instance or iterator\n"));
16259         scfdie();
16260     }
16262     if ((instances = uu_list_create(string_pool, NULL, 0)) == NULL)
16263         return (instances);
16265     if (scf_iter_service_instances(inst_iter, svc) != 0) {
16266         switch (scf_error()) {
16267             case SCF_ERROR_CONNECTION_BROKEN:
16268             case SCF_ERROR_DELETED:
16269                 uu_list_destroy(instances);
16270                 instances = NULL;
16271                 goto out;
16273             case SCF_ERROR_HANDLE_MISMATCH:
16274             case SCF_ERROR_NOT_BOUND:
16275             case SCF_ERROR_NOT_SET:
16276             default:

```

```

16277         bad_error("scf_iter_service_instances", scf_error());
16278     }
16279 }

16281 instname = safe_malloc(max_scf_name_len + 1);
16282 while ((r = scf_iter_next_instance(inst_iter, inst)) != 0) {
16283     if (r == -1) {
16284         (void) uu_warn(gettext("Unable to iterate through "
16285             "instances to create instance list : %s\n"),
16286             scf_strerror(scf_error()));

16288         uu_list_destroy(instances);
16289         instances = NULL;
16290         goto out;
16291     }

16293     /*
16294     * If the instance does not have a last-import snapshot
16295     * then do not add it to the list as it is a hand-crafted
16296     * instance that should not be managed.
16297     */
16298     if (wohandcrafted) {
16299         if (snap == NULL &&
16300             (snap = scf_snapshot_create(g_hdl)) == NULL) {
16301             uu_warn(gettext("Unable to create snapshot "
16302                 "entity\n"));
16303             scfdie();
16304         }

16306         if (scf_instance_get_snapshot(inst,
16307             snap_lastimport, snap) != 0) {
16308             switch (scf_error()) {
16309                 case SCF_ERROR_NOT_FOUND :
16310                 case SCF_ERROR_DELETED:
16311                     continue;

16313                 case SCF_ERROR_CONNECTION_BROKEN:
16314                     uu_list_destroy(instances);
16315                     instances = NULL;
16316                     goto out;

16318                 case SCF_ERROR_HANDLE_MISMATCH:
16319                 case SCF_ERROR_NOT_BOUND:
16320                 case SCF_ERROR_NOT_SET:
16321                 default:
16322                     bad_error("scf_iter_service_instances",
16323                         scf_error());
16324             }
16325         }
16326     }

16328     if (scf_instance_get_name(inst, instname,
16329         max_scf_name_len + 1) < 0) {
16330         switch (scf_error()) {
16331             case SCF_ERROR_NOT_FOUND :
16332                 continue;

16334             case SCF_ERROR_CONNECTION_BROKEN:
16335             case SCF_ERROR_DELETED:
16336                 uu_list_destroy(instances);
16337                 instances = NULL;
16338                 goto out;

16340             case SCF_ERROR_HANDLE_MISMATCH:
16341             case SCF_ERROR_NOT_BOUND:
16342             case SCF_ERROR_NOT_SET:

```

```

16343         default:
16344             bad_error("scf_iter_service_instances",
16345                 scf_error());
16346     }
16347 }

16349     add_string(instances, instname);
16350 }

16352 out:
16353     if (snap)
16354         scf_snapshot_destroy(snap);

16356     scf_instance_destroy(inst);
16357     scf_iter_destroy(inst_iter);
16358     free(instname);
16359     return (instances);
16360 }

16362 /*
16363 * disable an instance but wait for the instance to
16364 * move out of the running state.
16365 *
16366 * Returns 0 : if the instance did not disable
16367 * Returns non-zero : if the instance disabled.
16368 *
16369 */
16370 static int
16371 disable_instance(scf_instance_t *instance)
16372 {
16373     char    *fmribuf;
16374     int     enabled = 10000;

16376     if (inst_is_running(instance)) {
16377         fmribuf = safe_malloc(max_scf_name_len + 1);
16378         if (scf_instance_to_fmri(instance, fmribuf,
16379             max_scf_name_len + 1) < 0) {
16380             free(fmribuf);
16381             return (0);
16382         }

16384         /*
16385         * If the instance cannot be disabled then return
16386         * failure to disable and let the caller decide
16387         * if that is of importance.
16388         */
16389         if (smf_disable_instance(fmribuf, 0) != 0) {
16390             free(fmribuf);
16391             return (0);
16392         }

16394         while (enabled) {
16395             if (!inst_is_running(instance))
16396                 break;

16398             (void) poll(NULL, 0, 5);
16399             enabled = enabled - 5;
16400         }

16402         free(fmribuf);
16403     }

16405     return (enabled);
16406 }

16408 /*

```

```

16409 * Function to compare two service_manifest structures.
16410 */
16411 /* ARGSUSED2 */
16412 static int
16413 service_manifest_compare(const void *left, const void *right, void *unused)
16414 {
16415     service_manifest_t *l = (service_manifest_t *)left;
16416     service_manifest_t *r = (service_manifest_t *)right;
16417     int rc;

16419     rc = strcmp(l->servicename, r->servicename);

16421     return (rc);
16422 }

16424 /*
16425 * Look for the provided service in the service to manifest
16426 * tree. If the service exists, and a manifest was provided
16427 * then add the manifest to that service. If the service
16428 * does not exist, then add the service and manifest to the
16429 * list.
16430 *
16431 * If the manifest is NULL, return the element if found. If
16432 * the service is not found return NULL.
16433 */
16434 service_manifest_t *
16435 find_add_svc_mfst(const char *svnbuf, const char *mfst)
16436 {
16437     service_manifest_t     elem;
16438     service_manifest_t     *fnelem;
16439     uu_avl_index_t         marker;

16441     elem.servicename = svnbuf;
16442     fnelem = uu_avl_find(service_manifest_tree, &elem, NULL, &marker);

16444     if (mfst) {
16445         if (fnelem) {
16446             add_string(fnelem->mfstlist, strdup(mfst));
16447         } else {
16448             fnelem = safe_malloc(sizeof(*fnelem));
16449             fnelem->servicename = safe_strdup(svnbuf);
16450             if ((fnelem->mfstlist =
16451                 uu_list_create(string_pool, NULL, 0)) == NULL)
16452                 uu_die(gettext("Could not create property "
16453                     "list: %s\n"), uu_strerror(uu_error()));
16455             add_string(fnelem->mfstlist, safe_strdup(mfst));

16457             uu_avl_insert(service_manifest_tree, fnelem, marker);
16458         }
16459     }

16461     return (fnelem);
16462 }

16464 /*
16465 * Create the service to manifest avl tree.
16466 *
16467 * Walk each of the manifests currently installed in the supported
16468 * directories, /lib/svc/manifests and /var/svc/manifests. For
16469 * each of the manifests, inventory the services and add them to
16470 * the tree.
16471 *
16472 * Code that calls this function should make sure filesystem/minimal is online,
16473 * /var is available, since this function walks the /var/svc/manifest directory.
16474 */

```

```

16475 static void
16476 create_manifest_tree(void)
16477 {
16478     manifest_info_t **entry;
16479     manifest_info_t **manifests;
16480     uu_list_walk_t *svcs;
16481     bundle_t *b;
16482     entity_t *mfsvc;
16483     char *dirs[] = {LIBSVC_DIR, VARSVC_DIR, NULL};
16484     int c, status;

16486     if (service_manifest_pool)
16487         return;

16489     /*
16490     * Create the list pool for the service manifest list
16491     */
16492     service_manifest_pool = uu_avl_pool_create("service_manifest",
16493         sizeof(service_manifest_t),
16494         offsetof(service_manifest_t, svcmfst_node),
16495         service_manifest_compare, UU_DEFAULT);
16496     if (service_manifest_pool == NULL)
16497         uu_die(gettext("service_manifest pool creation failed: %s\n"),
16498             uu_strerror(uu_error()));

16500     /*
16501     * Create the list
16502     */
16503     service_manifest_tree = uu_avl_create(service_manifest_pool, NULL,
16504         UU_DEFAULT);
16505     if (service_manifest_tree == NULL)
16506         uu_die(gettext("service_manifest tree creation failed: %s\n"),
16507             uu_strerror(uu_error()));

16509     /*
16510     * Walk the manifests adding the service(s) from each manifest.
16511     *
16512     * If a service already exists add the manifest to the manifest
16513     * list for that service. This covers the case of a service that
16514     * is supported by multiple manifest files.
16515     */
16516     for (c = 0; dirs[c]; c++) {
16517         status = find_manifests(g_hndl, dirs[c], &manifests, CHECKEXT);
16518         if (status < 0) {
16519             uu_warn(gettext("file tree walk of %s encountered "
16520                 "error %s\n"), dirs[c], strerror(errno));

16522             uu_avl_destroy(service_manifest_tree);
16523             service_manifest_tree = NULL;
16524             return;
16525         }

16527         /*
16528         * If a manifest that was in the list is not found
16529         * then skip and go to the next manifest file.
16530         */
16531         if (manifests != NULL) {
16532             for (entry = manifests; *entry != NULL; entry++) {
16533                 b = internal_bundle_new();
16534                 if (lxml_get_bundle_file(b, (*entry)->mi_path,
16535                     SVCCFG_OP_IMPORT) != 0) {
16536                     internal_bundle_free(b);
16537                     continue;
16538                 }

16540                 svcs = uu_list_walk_start(b->sc_bundle_services,

```

```

16541     0);
16542     if (svcs == NULL) {
16543         internal_bundle_free(b);
16544         continue;
16545     }
16547     while ((mfsvc = uu_list_walk_next(svcs)) !=
16548            NULL) {
16549         /* Add manifest to service */
16550         (void) find_add_svc_mfst(mfsvc->sc_name,
16551                                (*entry)->mi_path);
16552     }
16554     uu_list_walk_end(svcs);
16555     internal_bundle_free(b);
16556 }
16558     free_manifest_array(manifests);
16559 }
16560 }
16561 }
16563 /*
16564 * Check the manifest history file to see
16565 * if the service was ever installed from
16566 * one of the supported directories.
16567 *
16568 * Return Values :
16569 *   -1 - if there's error reading manifest history file
16570 *    1 - if the service is not found
16571 *    0 - if the service is found
16572 */
16573 static int
16574 check_mfst_history(const char *svcname)
16575 {
16576     struct stat      st;
16577     caddr_t          mfsthist_start;
16578     char             *svnbuf;
16579     int              fd;
16580     int              r = 1;
16582     fd = open(MFSTHISTFILE, O_RDONLY);
16583     if (fd == -1) {
16584         uu_warn(gettext("Unable to open the history file\n"));
16585         return (-1);
16586     }
16588     if (fstat(fd, &st) == -1) {
16589         uu_warn(gettext("Unable to stat the history file\n"));
16590         return (-1);
16591     }
16593     mfsthist_start = mmap(0, st.st_size, PROT_READ,
16594                          MAP_PRIVATE, fd, 0);
16596     (void) close(fd);
16597     if (mfsthist_start == MAP_FAILED ||
16598         *(mfsthist_start + st.st_size) != '\0') {
16599         (void) munmap(mfsthist_start, st.st_size);
16600         return (-1);
16601     }
16603     /*
16604     * The manifest history file is a space delimited list
16605     * of service and instance to manifest linkage. Adding
16606     * a space to the end of the service name so to get only

```

```

16607     * the service that is being searched for.
16608     */
16609     svnbuf = uu_msprintf("%s ", svcname);
16610     if (svnbuf == NULL)
16611         uu_die(gettext("Out of memory"));
16613     if (strstr(mfsthist_start, svnbuf) != NULL)
16614         r = 0;
16616     (void) munmap(mfsthist_start, st.st_size);
16617     uu_free(svnbuf);
16618     return (r);
16619 }
16621 /*
16622 * Take down each of the instances in the service
16623 * and remove them, then delete the service.
16624 */
16625 static void
16626 teardown_service(scf_service_t *svc, const char *svnbuf)
16627 {
16628     scf_instance_t *instance;
16629     scf_iter_t      *iter;
16630     int             r;
16632     safe_printf(gettext("Delete service %s as there are no "
16633                        "supporting manifests\n"), svnbuf);
16635     instance = scf_instance_create(g_hdl);
16636     iter = scf_iter_create(g_hdl);
16637     if (iter == NULL || instance == NULL) {
16638         uu_warn(gettext("Unable to create supporting entities to "
16639                        "teardown the service\n"));
16640         uu_warn(gettext("scf error is : %s\n"),
16641                scf_strerror(scf_error()));
16642         scfdie();
16643     }
16645     if (scf_iter_service_instances(iter, svc) != 0) {
16646         switch (scf_error()) {
16647             case SCF_ERROR_CONNECTION_BROKEN:
16648             case SCF_ERROR_DELETED:
16649                 goto out;
16651             case SCF_ERROR_HANDLE_MISMATCH:
16652             case SCF_ERROR_NOT_BOUND:
16653             case SCF_ERROR_NOT_SET:
16654                 default:
16655                     bad_error("scf_iter_service_instances",
16656                               scf_error());
16657         }
16658     }
16660     while ((r = scf_iter_next_instance(iter, instance)) != 0) {
16661         if (r == -1) {
16662             uu_warn(gettext("Error - %s\n"),
16663                    scf_strerror(scf_error()));
16664             goto out;
16665         }
16667         (void) disable_instance(instance);
16668     }
16670     /*
16671     * Delete the service... forcing the deletion in case
16672     * any of the instances did not disable.

```

```

16673     */
16674     (void) lscf_service_delete(svc, 1);
16675 out:
16676     scf_instance_destroy(instance);
16677     scf_iter_destroy(iter);
16678 }

16680 /*
16681  * Get the list of instances supported by the manifest
16682  * file.
16683  *
16684  * Return 0 if there are no instances.
16685  *
16686  * Return -1 if there are errors attempting to collect instances.
16687  *
16688  * Return the count of instances found if there are no errors.
16689  */
16690 static int
16691 check_instance_support(char *mfstfile, const char *svcname,
16692     uu_list_t *instances)
16693 {
16694     {
16695         uu_list_walk_t *svcs, *insts;
16696         uu_list_t *ilist;
16697         bundle_t *b;
16698         entity_t *mfsvc, *mfinst;
16699         const char *svcn;
16700         int rminstcnt = 0;
16703
16704         b = internal_bundle_new();
16705
16706         if (lxml_get_bundle_file(b, mfstfile, SVCCFG_OP_IMPORT) != 0) {
16707             /*
16708              * Unable to process the manifest file for
16709              * instance support, so just return as
16710              * don't want to remove instances that could
16711              * not be accounted for that might exist here.
16712              */
16713             internal_bundle_free(b);
16714             return (0);
16715         }
16716
16717         svcs = uu_list_walk_start(b->sc_bundle_services, 0);
16718         if (svcs == NULL) {
16719             internal_bundle_free(b);
16720             return (0);
16721         }
16722
16723         svcn = svcname + (sizeof (SCF_FMRI_SVC_PREFIX) - 1) +
16724             (sizeof (SCF_FMRI_SERVICE_PREFIX) - 1);
16725
16726         while ((mfsvc = uu_list_walk_next(svcs)) != NULL) {
16727             if (strcmp(mfsvc->sc_name, svcn) == 0)
16728                 break;
16729         }
16730         uu_list_walk_end(svcs);
16731
16732         if (mfsvc == NULL) {
16733             internal_bundle_free(b);
16734             return (-1);
16735         }
16736
16737         ilist = mfsvc->sc_u.sc_service.sc_service_instances;
16738         if ((insts = uu_list_walk_start(ilist, 0)) == NULL) {
16739             internal_bundle_free(b);

```

```

16739         return (0);
16740     }
16741
16742     while ((mfinst = uu_list_walk_next(insts)) != NULL) {
16743         /*
16744          * Remove the instance from the instances list.
16745          * The unaccounted for instances will be removed
16746          * from the service once all manifests are
16747          * processed.
16748          */
16749         (void) remove_string(instances,
16750             mfinst->sc_name);
16751         rminstcnt++;
16752     }
16753
16754     uu_list_walk_end(insts);
16755     internal_bundle_free(b);
16756
16757     return (rminstcnt);
16758 }

16759 /*
16760  * For the given service, set its SCF_PG_MANIFESTFILES/SUPPORT property to
16761  * 'false' to indicate there's no manifest file(s) found for the service.
16762  */
16763 static void
16764 svc_add_no_support(scf_service_t *svc)
16765 {
16766     {
16767         char *pname;
16768
16769         /* Add no support */
16770         cur_svc = svc;
16771         if (addpg(SCF_PG_MANIFESTFILES, SCF_GROUP_FRAMEWORK))
16772             return;
16773
16774         pname = uu_msprintf("%s/%s", SCF_PG_MANIFESTFILES, SUPPORTPROP);
16775         if (pname == NULL)
16776             uu_die(gettext("Out of memory.\n"));
16777
16778         (void) lscf_addpropvalue(pname, "boolean:", "0");
16779
16780         uu_free(pname);
16781         cur_svc = NULL;
16782     }
16783
16784 /*
16785  * This function handles all upgrade scenarios for a service that doesn't have
16786  * SCF_PG_MANIFESTFILES pg. The function creates and populates
16787  * SCF_PG_MANIFESTFILES pg for the given service to keep track of service to
16788  * manifest(s) mapping. Manifests under supported directories are inventoried
16789  * and a property is added for each file that delivers configuration to the
16790  * service. A service that has no corresponding manifest files (deleted) are
16791  * removed from repository.
16792  *
16793  * Unsupported services:
16794  *
16795  * A service is considered unsupported if there is no corresponding manifest
16796  * in the supported directories for that service and the service isn't in the
16797  * history file list. The history file, MFSTHISTFILE, contains a list of all
16798  * services and instances that were delivered by Solaris before the introduction
16799  * of the SCF_PG_MANIFESTFILES property group. The history file also contains
16800  * the path to the manifest file that defined the service or instance.
16801  *
16802  * Another type of unsupported services is 'handcrafted' services,
16803  * programmatically created services or services created by dependent entries
16804  * in other manifests. A handcrafted service is identified by its lack of any

```

```

16805 * instance containing last-import snapshot which is created during svccfg
16806 * import.
16807 *
16808 * This function sets a flag for unsupported services by setting services'
16809 * SCF_PG_MANIFESTFILES/support property to false.
16810 */
16811 static void
16812 upgrade_svc_mfst_connection(scf_service_t *svc, const char *svcname)
16813 {
16814     service_manifest_t    *elem;
16815     uu_list_walk_t        *mfwalk;
16816     string_list_t         *mfile;
16817     uu_list_t             *instances;
16818     const char            *sname;
16819     char                  *pname;
16820     int                   r;
16821
16822     /*
16823     * Since there's no guarantee manifests under /var are available during
16824     * early import, don't perform any upgrade during early import.
16825     */
16826     if (IGNORE_VAR)
16827         return;
16828
16829     if (service_manifest_tree == NULL) {
16830         create_manifest_tree();
16831     }
16832
16833     /*
16834     * Find service's supporting manifest(s) after
16835     * stripping off the svc:/ prefix that is part
16836     * of the fmri that is not used in the service
16837     * manifest bundle list.
16838     */
16839     sname = svcname + strlen(SCF_FMRI_SVC_PREFIX) +
16840           strlen(SCF_FMRI_SERVICE_PREFIX);
16841     elem = find_add_svc_mfst(sname, NULL);
16842     if (elem == NULL) {
16843
16844         /*
16845         * A handcrafted service, one that has no instance containing
16846         * last-import snapshot, should get unsupported flag.
16847         */
16848         instances = create_instance_list(svc, 1);
16849         if (instances == NULL) {
16850             uu_warn(gettext("Unable to create instance list %s\n"),
16851                   svcname);
16852             return;
16853         }
16854
16855         if (uu_list_numnodes(instances) == 0) {
16856             svc_add_no_support(svc);
16857             return;
16858         }
16859
16860         /*
16861         * If the service is in the history file, and its supporting
16862         * manifests are not found, we can safely delete the service
16863         * because its manifests are removed from the system.
16864         *
16865         * Services not found in the history file are not delivered by
16866         * Solaris and/or delivered outside supported directories, set
16867         * unsupported flag for these services.
16868         */
16869         r = check_mfst_history(svcname);
16870         if (r == -1)

```

```

16871         return;
16872
16873         if (r) {
16874             /* Set unsupported flag for service */
16875             svc_add_no_support(svc);
16876         } else {
16877             /* Delete the service */
16878             teardown_service(svc, svcname);
16879         }
16880
16881         return;
16882     }
16883
16884     /*
16885     * Walk through the list of manifests and add them
16886     * to the service.
16887     *
16888     * Create a manifestfiles pg and add the property.
16889     */
16890     mfwalk = uu_list_walk_start(elem->mfstlist, 0);
16891     if (mfwalk == NULL)
16892         return;
16893
16894     cur_svc = svc;
16895     r = addpg(SCF_PG_MANIFESTFILES, SCF_GROUP_FRAMEWORK);
16896     if (r != 0) {
16897         cur_svc = NULL;
16898         return;
16899     }
16900
16901     while ((mfile = uu_list_walk_next(mfwalk)) != NULL) {
16902         pname = uu_msprintf("%s/%s", SCF_PG_MANIFESTFILES,
16903                             mhash_filename_to_propname(mfile->str, 0));
16904         if (pname == NULL)
16905             uu_die(gettext("Out of memory.\n"));
16906
16907         (void) lscf_addpropvalue(pname, "astring:", mfile->str);
16908         uu_free(pname);
16909     }
16910     uu_list_walk_end(mfwalk);
16911     cur_svc = NULL;
16912 }
16913
16914 /*
16915 * Take a service and process the manifest file entires to see if
16916 * there is continued support for the service and instances. If
16917 * not cleanup as appropriate.
16918 *
16919 * If a service does not have a manifest files entry flag it for
16920 * upgrade and return.
16921 *
16922 * For each manifestfiles property check if the manifest file is
16923 * under the supported /lib/svc/manifest or /var/svc/manifest path
16924 * and if not then return immediately as this service is not supported
16925 * by the cleanup mechanism and should be ignored.
16926 *
16927 * For each manifest file that is supported, check to see if the
16928 * file exists. If not then remove the manifest file property
16929 * from the service and the smf/manifest hash table. If the manifest
16930 * file exists then verify that it supports the instances that are
16931 * part of the service.
16932 *
16933 * Once all manifest files have been accounted for remove any instances
16934 * that are no longer supported in the service.
16935 */
16936

```

```

16937 * Return values :
16938 * 0 - Successfully processed the service
16939 * non-zero - failed to process the service
16940 *
16941 * On most errors, will just return to wait and get the next service,
16942 * unless in case of unable to create the needed structures which is
16943 * most likely a fatal error that is not going to be recoverable.
16944 */
16945 int
16946 lscf_service_cleanup(void *act, scf_walkinfo_t *wip)
16947 {
16948     struct mpg_mfile      *mpntov;
16949     struct mpg_mfile      *mpvarry = NULL;
16950     scf_service_t         *svc;
16951     scf_propertygroup_t   *mpg;
16952     scf_property_t        *mp;
16953     scf_value_t           *mv;
16954     scf_iter_t            *mi;
16955     scf_instance_t        *instance;
16956     uu_list_walk_t        *insts;
16957     uu_list_t             *instances = NULL;
16958     boolean_t             activity = (boolean_t)act;
16959     char                   *mpnbuf;
16960     char                   *mpvbuf;
16961     char                   *pgpropbuf;
16962     int                    mfstcnt, rminstct, instct, mfstmax;
16963     int                    index;
16964     int                    r = 0;

16966     assert(g_hndl != NULL);
16967     assert(wip->svc != NULL);
16968     assert(wip->fmri != NULL);

16970     svc = wip->svc;

16972     mpg = scf_pg_create(g_hndl);
16973     mp = scf_property_create(g_hndl);
16974     mi = scf_iter_create(g_hndl);
16975     mv = scf_value_create(g_hndl);
16976     instance = scf_instance_create(g_hndl);

16978     if (mpg == NULL || mp == NULL || mi == NULL || mv == NULL ||
16979         instance == NULL) {
16980         uu_warn(gettext("Unable to create the supporting entities\n"));
16981         uu_warn(gettext("scf error is : %s\n"),
16982             scf_strerror(scf_error()));
16983         scfdie();
16984     }

16986     /*
16987     * Get the manifestfiles property group to be parsed for
16988     * files existence.
16989     */
16990     if (scf_service_get_pg(svc, SCF_PG_MANIFESTFILES, mpg) != SCF_SUCCESS) {
16991         switch (scf_error()) {
16992             case SCF_ERROR_NOT_FOUND:
16993                 upgrade_svc_mfst_connection(svc, wip->fmri);
16994                 break;
16995             case SCF_ERROR_DELETED:
16996             case SCF_ERROR_CONNECTION_BROKEN:
16997                 goto out;

16999             case SCF_ERROR_HANDLE_MISMATCH:
17000             case SCF_ERROR_NOT_BOUND:
17001             case SCF_ERROR_NOT_SET:
17002             default:

```

```

17003         bad_error("scf_iter_pg_properties",
17004             scf_error());
17005     }
17007     goto out;
17008 }

17010 /*
17011 * Iterate through each of the manifestfiles properties
17012 * to determine what manifestfiles are available.
17013 *
17014 * If a manifest file is supported then increment the
17015 * count and therefore the service is safe.
17016 */
17017 if (scf_iter_pg_properties(mi, mpg) != 0) {
17018     switch (scf_error()) {
17019         case SCF_ERROR_DELETED:
17020         case SCF_ERROR_CONNECTION_BROKEN:
17021             goto out;

17023         case SCF_ERROR_HANDLE_MISMATCH:
17024         case SCF_ERROR_NOT_BOUND:
17025         case SCF_ERROR_NOT_SET:
17026         default:
17027             bad_error("scf_iter_pg_properties",
17028                 scf_error());
17029     }
17030 }

17032 mfstcnt = 0;
17033 mfstmax = MFSTFILE_MAX;
17034 mpvarry = safe_malloc(sizeof (struct mpg_file *) * MFSTFILE_MAX);
17035 while ((r = scf_iter_next_property(mi, mp)) != 0) {
17036     if (r == -1)
17037         bad_error(gettext("Unable to iterate through "
17038             "manifestfiles properties : %s"),
17039             scf_error());

17041     mpntov = safe_malloc(sizeof (struct mpg_mfile));
17042     mpnbuf = safe_malloc(max_scf_name_len + 1);
17043     mpvbuf = safe_malloc(max_scf_value_len + 1);
17044     mpntov->mpg = mpnbuf;
17045     mpntov->mfile = mpvbuf;
17046     mpntov->access = 1;
17047     if (scf_property_get_name(mp, mpnbuf,
17048         max_scf_name_len + 1) < 0) {
17049         uu_warn(gettext("Unable to get manifest file "
17050             "property : %s\n"),
17051             scf_strerror(scf_error()));

17053         switch (scf_error()) {
17054             case SCF_ERROR_DELETED:
17055             case SCF_ERROR_CONNECTION_BROKEN:
17056                 r = scferror2errno(scf_error());
17057                 goto out_free;

17059             case SCF_ERROR_HANDLE_MISMATCH:
17060             case SCF_ERROR_NOT_BOUND:
17061             case SCF_ERROR_NOT_SET:
17062             default:
17063                 bad_error("scf_iter_pg_properties",
17064                     scf_error());
17065         }
17066     }
17068     /*

```

```

17069     * The support property is a boolean value that indicates
17070     * if the service is supported for manifest file deletion.
17071     * Currently at this time there is no code that sets this
17072     * value to true. So while we could just let this be caught
17073     * by the support check below, in the future this by be set
17074     * to true and require processing. So for that, go ahead
17075     * and check here, and just return if false. Otherwise,
17076     * fall through expecting that other support checks will
17077     * handle the entries.
17078     */
17079     if (strcmp(mpnbuf, SUPPORTPROP) == 0) {
17080         uint8_t support;
17081
17082         if (scf_property_get_value(mp, mv) != 0 ||
17083             scf_value_get_boolean(mv, &support) != 0) {
17084             uu_warn(gettext("Unable to get the manifest ")
17085                 "support value: %s\n"),
17086                 scf_strerror(scf_error()));
17087
17088             switch (scf_error()) {
17089             case SCF_ERROR_DELETED:
17090             case SCF_ERROR_CONNECTION_BROKEN:
17091                 r = scferror2errno(scf_error());
17092                 goto out_free;
17093
17094             case SCF_ERROR_HANDLE_MISMATCH:
17095             case SCF_ERROR_NOT_BOUND:
17096             case SCF_ERROR_NOT_SET:
17097             default:
17098                 bad_error("scf_iter_pg_properties",
17099                     scf_error());
17100             }
17101         }
17102
17103         if (support == B_FALSE)
17104             goto out_free;
17105     }
17106
17107     /*
17108     * Anything with a manifest outside of the supported
17109     * directories, immediately bail out because that makes
17110     * this service non-supported. We don't even want
17111     * to do instance processing in this case because the
17112     * instances could be part of the non-supported manifest.
17113     */
17114     if (strncmp(mpnbuf, LIBSVC_PR, strlen(LIBSVC_PR)) != 0) {
17115         /*
17116         * Manifest is not in /lib/svc, so we need to
17117         * consider the /var/svc case.
17118         */
17119         if (strncmp(mpnbuf, VARSVC_PR,
17120             strlen(VARSVC_PR)) != 0 || IGNORE_VAR) {
17121             /*
17122             * Either the manifest is not in /var/svc or
17123             * /var is not yet mounted. We ignore the
17124             * manifest either because it is not in a
17125             * standard location or because we cannot
17126             * currently access the manifest.
17127             */
17128             goto out_free;
17129         }
17130     }
17131
17132     /*
17133     * Get the value to of the manifest file for this entry
17134     * for access verification and instance support

```

```

17135     * verification if it still exists.
17136     *
17137     * During Early Manifest Import if the manifest is in
17138     * /var/svc then it may not yet be available for checking
17139     * so we must determine if /var/svc is available. If not
17140     * then defer until Late Manifest Import to cleanup.
17141     */
17142     if (scf_property_get_value(mp, mv) != 0) {
17143         uu_warn(gettext("Unable to get the manifest file ")
17144             "value: %s\n"),
17145             scf_strerror(scf_error()));
17146
17147         switch (scf_error()) {
17148         case SCF_ERROR_DELETED:
17149         case SCF_ERROR_CONNECTION_BROKEN:
17150             r = scferror2errno(scf_error());
17151             goto out_free;
17152
17153         case SCF_ERROR_HANDLE_MISMATCH:
17154         case SCF_ERROR_NOT_BOUND:
17155         case SCF_ERROR_NOT_SET:
17156         default:
17157             bad_error("scf_property_get_value",
17158                 scf_error());
17159         }
17160     }
17161
17162     if (scf_value_get_astring(mv, mpvbuf,
17163         max_scf_value_len + 1) < 0) {
17164         uu_warn(gettext("Unable to get the manifest ")
17165             "file : %s\n"),
17166             scf_strerror(scf_error()));
17167
17168         switch (scf_error()) {
17169         case SCF_ERROR_DELETED:
17170         case SCF_ERROR_CONNECTION_BROKEN:
17171             r = scferror2errno(scf_error());
17172             goto out_free;
17173
17174         case SCF_ERROR_HANDLE_MISMATCH:
17175         case SCF_ERROR_NOT_BOUND:
17176         case SCF_ERROR_NOT_SET:
17177         default:
17178             bad_error("scf_value_get_astring",
17179                 scf_error());
17180         }
17181     }
17182
17183     mpvarry[mfstcnt] = mpntov;
17184     mfstcnt++;
17185
17186     /*
17187     * Check for the need to reallocate array
17188     */
17189     if (mfstcnt >= (mfstmax - 1)) {
17190         struct mpg_mfile **newmpvarry;
17191
17192         mfstmax = mfstmax * 2;
17193         newmpvarry = realloc(mpvarry,
17194             sizeof (struct mpg_mfile *) * mfstmax);
17195
17196         if (newmpvarry == NULL)
17197             goto out_free;
17198
17199         mpvarry = newmpvarry;
17200     }

```



```

17202         mpvarry[mfstcnt] = NULL;
17203     }

17205     for (index = 0; mpvarry[index]; index++) {
17206         mpntov = mpvarry[index];

17208         /*
17209          * Check to see if the manifestfile is accessible, if so hand
17210          * this service and manifestfile off to be processed for
17211          * instance support.
17212          */
17213         mpnbuf = mpntov->mpg;
17214         mpvbuf = mpntov->mfile;
17215         if (access(mpvbuf, F_OK) != 0) {
17216             mpntov->access = 0;
17217             activity++;
17218             mfstcnt--;
17219             /* Remove the entry from the service */
17220             cur_svc = svc;
17221             pgpropbuf = uu_msprintf("%s/%s", SCF_PG_MANIFESTFILES,
17222                 mpnbuf);
17223             if (pgpropbuf == NULL)
17224                 uu_die(gettext("Out of memory.\n"));

17226             lscf_delprop(pgpropbuf);
17227             cur_svc = NULL;

17229             uu_free(pgpropbuf);
17230         }
17231     }

17233     /*
17234      * If mfstcnt is 0, none of the manifests that supported the service
17235      * existed so remove the service.
17236      */
17237     if (mfstcnt == 0) {
17238         teardown_service(svc, wip->fmri);

17240         goto out_free;
17241     }

17243     if (activity) {
17244         int    nosvcsupport = 0;

17246         /*
17247          * If the list of service instances is NULL then
17248          * create the list.
17249          */
17250         instances = create_instance_list(svc, 1);
17251         if (instances == NULL) {
17252             uu_warn(gettext("Unable to create instance list %s\n"),
17253                 wip->fmri);
17254             goto out_free;
17255         }

17257         rminstct = uu_list_numnodes(instances);
17258         instct = rminstct;

17260         for (index = 0; mpvarry[index]; index++) {
17261             mpntov = mpvarry[index];
17262             if (mpntov->access == 0)
17263                 continue;

17265             mpnbuf = mpntov->mpg;
17266             mpvbuf = mpntov->mfile;

```

```

17267         r = check_instance_support(mpvbuf, wip->fmri,
17268             instances);
17269         if (r == -1) {
17270             nosvcsupport++;
17271         } else {
17272             rminstct -= r;
17273         }
17274     }

17276     if (instct && instct == rminstct && nosvcsupport == mfstcnt) {
17277         teardown_service(svc, wip->fmri);

17279         goto out_free;
17280     }
17281 }

17283 /*
17284  * If there are instances left on the instance list, then
17285  * we must remove them.
17286  */
17287 if (instances != NULL && uu_list_numnodes(instances)) {
17288     string_list_t *sp;

17290     insts = uu_list_walk_start(instances, 0);
17291     while ((sp = uu_list_walk_next(insts)) != NULL) {
17292         /*
17293          * Remove the instance from the instances list.
17294          */
17295         safe_printf(gettext("Delete instance %s from "
17296             "service %s\n"), sp->str, wip->fmri);
17297         if (scf_service_get_instance(svc, sp->str,
17298             instance) != SCF_SUCCESS) {
17299             (void) uu_warn("scf_error - %s\n",
17300                 scf_strerror(scf_error()));

17302             continue;
17303         }

17305         (void) disable_instance(instance);

17307         (void) lscf_instance_delete(instance, 1);
17308     }
17309     scf_instance_destroy(instance);
17310     uu_list_walk_end(insts);
17311 }

17313 out_free:
17314     if (mpvarry) {
17315         struct mpg_mfile *fmpntov;

17317         for (index = 0; mpvarry[index]; index++) {
17318             fmpntov = mpvarry[index];
17319             if (fmpntov->mpg == mpnbuf)
17320                 mpnbuf = NULL;
17321             free(fmpntov->mpg);

17323             if (fmpntov->mfile == mpvbuf)
17324                 mpvbuf = NULL;
17325             free(fmpntov->mfile);

17327             if (fmpntov == mpntov)
17328                 mpntov = NULL;
17329             free(fmpntov);
17330         }
17331         if (mpnbuf)
17332             free(mpnbuf);

```

```

17333         if (mpvbuf)
17334             free(mpvbuf);
17335         if (mpntov)
17336             free(mpntov);
17337     }
17338     free(mpvary);
17339 }
17340 out:
17341     scf_pg_destroy(mpg);
17342     scf_property_destroy(mp);
17343     scf_iter_destroy(mi);
17344     scf_value_destroy(mv);
17345
17346     return (0);
17347 }
17348
17349 /*
17350  * Take the service and search for the manifestfiles property
17351  * in each of the property groups.  If the manifest file
17352  * associated with the property does not exist then remove
17353  * the property group.
17354  */
17355 int
17356 lscf_hash_cleanup()
17357 {
17358     scf_service_t      *svc;
17359     scf_scope_t        *scope;
17360     scf_propertygroup_t *pg;
17361     scf_property_t     *prop;
17362     scf_value_t        *val;
17363     scf_iter_t         *iter;
17364     char               *pgname = NULL;
17365     char               *mfile = NULL;
17366     int                r;
17367
17368     svc = scf_service_create(g_hndl);
17369     scope = scf_scope_create(g_hndl);
17370     pg = scf_pg_create(g_hndl);
17371     prop = scf_property_create(g_hndl);
17372     val = scf_value_create(g_hndl);
17373     iter = scf_iter_create(g_hndl);
17374     if (pg == NULL || prop == NULL || val == NULL || iter == NULL ||
17375         svc == NULL || scope == NULL) {
17376         uu_warn(gettext("Unable to create a property group, or "
17377             "property\n"));
17378         uu_warn("%s\n", pg == NULL ? "pg is NULL" :
17379             "pg is not NULL");
17380         uu_warn("%s\n", prop == NULL ? "prop is NULL" :
17381             "prop is not NULL");
17382         uu_warn("%s\n", val == NULL ? "val is NULL" :
17383             "val is not NULL");
17384         uu_warn("%s\n", iter == NULL ? "iter is NULL" :
17385             "iter is not NULL");
17386         uu_warn("%s\n", svc == NULL ? "svc is NULL" :
17387             "svc is not NULL");
17388         uu_warn("%s\n", scope == NULL ? "scope is NULL" :
17389             "scope is not NULL");
17390         uu_warn(gettext("scf error is : %s\n"),
17391             scf_strerror(scf_error()));
17392         scfdie();
17393     }
17394
17395     if (scf_handle_get_scope(g_hndl, SCF_SCOPE_LOCAL, scope) != 0) {
17396         switch (scf_error()) {
17397             case SCF_ERROR_CONNECTION_BROKEN:
17398             case SCF_ERROR_NOT_FOUND:

```

```

17399         goto out;
17400     }
17401     case SCF_ERROR_HANDLE_MISMATCH:
17402     case SCF_ERROR_NOT_BOUND:
17403     case SCF_ERROR_INVALID_ARGUMENT:
17404     default:
17405         bad_error("scf_handle_get_scope", scf_error());
17406     }
17407 }
17408
17409 if (scf_scope_get_service(scope, HASH_SVC, svc) != 0) {
17410     uu_warn(gettext("Unable to process the hash service, %s\n"),
17411         HASH_SVC);
17412     goto out;
17413 }
17414
17415 pgname = safe_malloc(max_scf_name_len + 1);
17416 mfile = safe_malloc(max_scf_value_len + 1);
17417
17418 if (scf_iter_service_pgs(iter, svc) != SCF_SUCCESS) {
17419     uu_warn(gettext("Unable to cleanup smf hash table : %s\n"),
17420         scf_strerror(scf_error()));
17421     goto out;
17422 }
17423
17424 while ((r = scf_iter_next_pg(iter, pg)) != 0) {
17425     if (r == -1)
17426         goto out;
17427
17428     if (scf_pg_get_name(pg, pgname, max_scf_name_len + 1) < 0) {
17429         switch (scf_error()) {
17430             case SCF_ERROR_DELETED:
17431                 return (ENODEV);
17432
17433             case SCF_ERROR_CONNECTION_BROKEN:
17434                 return (ECONNABORTED);
17435
17436             case SCF_ERROR_NOT_SET:
17437             case SCF_ERROR_NOT_BOUND:
17438             default:
17439                 bad_error("scf_pg_get_name", scf_error());
17440         }
17441     }
17442     if (IGNORE_VAR) {
17443         if (strncmp(pgname, VARSVC_PR, strlen(VARSVC_PR)) == 0)
17444             continue;
17445     }
17446     /*
17447      * If unable to get the property continue as this is an
17448      * entry that has no location to check against.
17449      */
17450     if (scf_pg_get_property(pg, MFSTFILEPR, prop) != SCF_SUCCESS) {
17451         continue;
17452     }
17453
17454     if (scf_property_get_value(prop, val) != SCF_SUCCESS) {
17455         uu_warn(gettext("Unable to get value from %s\n"),
17456             pgname);
17457     }
17458
17459     switch (scf_error()) {
17460     case SCF_ERROR_DELETED:
17461     case SCF_ERROR_CONSTRAINT_VIOLATED:
17462     case SCF_ERROR_NOT_FOUND:
17463     case SCF_ERROR_NOT_SET:
17464         continue;

```

```

17466         case SCF_ERROR_CONNECTION_BROKEN:
17467             r = scferror2errno(scf_error());
17468             goto out;

17470         case SCF_ERROR_HANDLE_MISMATCH:
17471         case SCF_ERROR_NOT_BOUND:
17472         default:
17473             bad_error("scf_property_get_value",
17474                     scf_error());
17475     }
17476 }

17478     if (scf_value_get_astring(val, mfile, max_scf_value_len + 1)
17479         == -1) {
17480         uu_warn(gettext("Unable to get astring from %s : %s\n"),
17481                pgname, scf_strerror(scf_error()));

17483         switch (scf_error()) {
17484         case SCF_ERROR_NOT_SET:
17485         case SCF_ERROR_TYPE_MISMATCH:
17486             continue;

17488         default:
17489             bad_error("scf_value_get_astring", scf_error());
17490         }
17491     }

17493     if (access(mfile, F_OK) == 0)
17494         continue;

17496     (void) scf_pg_delete(pg);
17497 }

17499 out:
17500     scf_scope_destroy(scope);
17501     scf_service_destroy(svc);
17502     scf_pg_destroy(pg);
17503     scf_property_destroy(prop);
17504     scf_value_destroy(val);
17505     scf_iter_destroy(iter);
17506     free(pgname);
17507     free(mfile);

17509     return (0);
17510 }

17512 #ifndef NATIVE_BUILD
17513 /* ARGSUSED */
17514 CPL_MATCH_FN(complete_select)
17515 {
17516     const char *arg0, *arg1, *arglend;
17517     int word_start, err = 0, r;
17518     size_t len;
17519     char *buf;

17521     lscf_prep_hdl();

17523     arg0 = line + strspn(line, " \t");
17524     assert(strncmp(arg0, "select", sizeof("select") - 1) == 0);

17526     arg1 = arg0 + sizeof("select") - 1;
17527     arg1 += strspn(arg1, " \t");
17528     word_start = arg1 - line;

17530     arglend = arg1 + strcspn(arg1, " \t");

```

```

17531     if (arglend < line + word_end)
17532         return (0);

17534     len = line + word_end - arg1;

17536     buf = safe_malloc(max_scf_name_len + 1);

17538     if (cur_snap != NULL) {
17539         return (0);
17540     } else if (cur_inst != NULL) {
17541         return (0);
17542     } else if (cur_svc != NULL) {
17543         scf_instance_t *inst;
17544         scf_iter_t *iter;

17546         if ((inst = scf_instance_create(g_hdl)) == NULL ||
17547             (iter = scf_iter_create(g_hdl)) == NULL)
17548             scfdie();

17550         if (scf_iter_service_instances(iter, cur_svc) != 0)
17551             scfdie();

17553         for (;;) {
17554             r = scf_iter_next_instance(iter, inst);
17555             if (r == 0)
17556                 break;
17557             if (r != 1)
17558                 scfdie();

17560             if (scf_instance_get_name(inst, buf,
17561                                     max_scf_name_len + 1) < 0)
17562                 scfdie();

17564             if (strncmp(buf, arg1, len) == 0) {
17565                 err = cpl_add_completion(cpl, line, word_start,
17566                                       word_end, buf + len, " ", " ");
17567                 if (err != 0)
17568                     break;
17569             }
17570         }

17572         scf_iter_destroy(iter);
17573         scf_instance_destroy(inst);

17575     } else {
17576         return (err);
17577         scf_service_t *svc;
17578         scf_iter_t *iter;

17580         assert(cur_scope != NULL);

17582         if ((svc = scf_service_create(g_hdl)) == NULL ||
17583             (iter = scf_iter_create(g_hdl)) == NULL)
17584             scfdie();

17586         if (scf_iter_scope_services(iter, cur_scope) != 0)
17587             scfdie();

17589         for (;;) {
17590             r = scf_iter_next_service(iter, svc);
17591             if (r == 0)
17592                 break;
17593             if (r != 1)
17594                 scfdie();

17596             if (scf_service_get_name(svc, buf,

```

```
17597         max_scf_name_len + 1) < 0)
17598         scfdie();
17600         if (strncmp(buf, arg1, len) == 0) {
17601             err = cpl_add_completion(cpl, line, word_start,
17602                 word_end, buf + len, "", " ");
17603             if (err != 0)
17604                 break;
17605         }
17606     }
17608     scf_iter_destroy(iter);
17609     scf_service_destroy(svc);
17611     return (err);
17612 }
17613 }
17615 /* ARGSUSED */
17616 CPL_MATCH_FN(complete_command)
17617 {
17618     uint32_t scope = 0;
17620     if (cur_snap != NULL)
17621         scope = CS_SNAP;
17622     else if (cur_inst != NULL)
17623         scope = CS_INST;
17624     else if (cur_svc != NULL)
17625         scope = CS_SVC;
17626     else
17627         scope = CS_SCOPE;
17629     return (scope ? add_cmd_matches(cpl, line, word_end, scope) : 0);
17630 }
17631 #endif /* NATIVE_BUILD */
```



```

1122     case SC_METHOD_CONTEXT:
1123         (void) lxml_get_method_context(pg, cursor);
1124         break;

1126     case SC_PROPVAL:
1127         (void) lxml_get_propval(pg, cursor);
1128         break;

1130     case SC_PROPERTY:
1131         (void) lxml_get_property(pg, cursor);
1132         break;

1134     default:
1135         uu_die(gettext("illegal element \"%s\" on "
1136                     "execution method \"%s\"\n"), cursor->name,
1137              pg->sc_pgroup_name);
1138         break;
1139     }
1140 }

1142 delete = xmlGetProp(emeth, (xmlChar *)delete_attr);
1143 pg->sc_pgroup_delete = (xmlStrcmp(delete, (xmlChar *)true) == 0);
1144 xmlFree(delete);

1146     return (0);
1147 }

```

unchanged portion omitted

```

3343 /*
3344  * Check to see if the service should allow the upgrade
3345  * process to handle adding of the manifestfiles linkage.
3346  *
3347  * If the service exists and does not have a manifestfiles
3348  * property group then the upgrade process should handle
3349  * the service.
3350  *
3351  * If the service doesn't exist or the service exists
3352  * and has a manifestfiles property group then the import
3353  * process can handle the manifestfiles property group
3354  * work.
3355  *
3356  * This prevents potential cleanup of unaccounted for instances
3357  * in early manifest import due to upgrade process needing
3358  * information that has not yet been supplied by manifests
3359  * that are still located in the /var/svc manifests directory.
3360  */

```

```

3361 static int
3362 lxml_check_upgrade(const char *service)
3363 {
3364     lxml_check_upgrade(const char *service) {
3364         scf_handle_t    *h = NULL;
3365         scf_scope_t     *sc = NULL;
3366         scf_service_t   *svc = NULL;
3367         scf_propertygroup_t *pg = NULL;
3368         int rc = SCF_FAILED;

3370         if ((h = scf_handle_create(SCF_VERSION)) == NULL ||
3371             (sc = scf_scope_create(h)) == NULL ||
3372             (svc = scf_service_create(h)) == NULL ||
3373             (pg = scf_pg_create(h)) == NULL)
3374             goto out;

3376         if (scf_handle_bind(h) != 0)
3377             goto out;

```

```

3379         if (scf_handle_get_scope(h, SCF_FMRI_LOCAL_SCOPE, sc) == -1)
3380             goto out;

3382         if (scf_scope_get_service(sc, service, svc) != SCF_SUCCESS) {
3383             if (scf_error() == SCF_ERROR_NOT_FOUND)
3384                 rc = SCF_SUCCESS;

3386             goto out;
3387         }

3389         if (scf_service_get_pg(svc, SCF_PG_MANIFESTFILES, pg) != SCF_SUCCESS)
3390             goto out;

3392         rc = SCF_SUCCESS;
3393     out:
3394         scf_pg_destroy(pg);
3395         scf_service_destroy(svc);
3396         scf_scope_destroy(sc);
3397         scf_handle_destroy(h);

3399         return (rc);
3400 }

```

unchanged portion omitted

new/usr/src/cmd/truss/print.c

1

```
*****
69675 Thu Jun 30 22:01:45 2016
new/usr/src/cmd/truss/print.c
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
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 /*
23  * Copyright (c) 1989, 2010, Oracle and/or its affiliates. All rights reserved.
24  * Copyright (c) 2015, Joyent, Inc. All rights reserved.
25  */
26
27 /*      Copyright (c) 1984, 1986, 1987, 1988, 1989 AT&T */
28 /*      All Rights Reserved      */
29
30 /* Copyright (c) 2013, OmniTI Computer Consulting, Inc. All rights reserved. */
31
32 #define _SYSCALL32      /* make 32-bit compat headers visible */
33
34 #include <stdio.h>
35 #include <stdlib.h>
36 #include <unistd.h>
37 #include <string.h>
38 #include <signal.h>
39 #include <termio.h>
40 #include <stddef.h>
41 #include <limits.h>
42 #include <fcntl.h>
43 #include <ctype.h>
44 #include <sys/types.h>
45 #include <sys/mman.h>
46 #include <sys/resource.h>
47 #include <sys/ulimit.h>
48 #include <sys/utsname.h>
49 #include <sys/kstat.h>
50 #include <sys/modctl.h>
51 #include <sys/acl.h>
52 #include <stropts.h>
53 #include <sys/isa_defs.h>
54 #include <sys/systeminfo.h>
55 #include <sys/cladm.h>
56 #include <sys/lwp.h>
57 #include <bsm/audit.h>
58 #include <libproc.h>
```

new/usr/src/cmd/truss/print.c

2

```
59 #include <priv.h>
60 #include <sys/aio.h>
61 #include <sys/aiocb.h>
62 #include <sys/corectl.h>
63 #include <sys/cpc_impl.h>
64 #include <sys/priocntl.h>
65 #include <sys/tspriocntl.h>
66 #include <sys/iapriocntl.h>
67 #include <sys/rtpriocntl.h>
68 #include <sys/fsspriocntl.h>
69 #include <sys/fxpriocntl.h>
70 #include <sys/proc.h>
71 #endif /* ! codereview */
72 #include <netdb.h>
73 #include <nss_dbdefs.h>
74 #include <sys/socketvar.h>
75 #include <netinet/in.h>
76 #include <netinet/tcp.h>
77 #include <netinet/udp.h>
78 #include <netinet/sctp.h>
79 #include <net/route.h>
80 #include <sys/utrap.h>
81 #include <sys/lgrp_user.h>
82 #include <sys/door.h>
83 #include <sys/tsol/tndb.h>
84 #include <sys/rctl.h>
85 #include <sys/rctl_impl.h>
86 #include <sys/fork.h>
87 #include <sys/task.h>
88 #include <sys/random.h>
89 #include "ramdata.h"
90 #include "print.h"
91 #include "proto.h"
92 #include "systable.h"
93
94 void grow(private_t *, int nbyte);
95
96 #define GROW(nb) if (pri->sys_leng + (nb) >= pri->sys_ssize) grow(pri, (nb))
97
98
99 /*ARGSUSED*/
100 void
101 prt_nov(private_t *pri, int raw, long val)      /* print nothing */
102 {
103 }
104
105 /*ARGSUSED*/
106 void
107 prt_dec(private_t *pri, int raw, long val)      /* print as decimal */
108 {
109     GROW(24);
110     if (data_model == PR_MODEL_ILP32)
111         pri->sys_leng += sprintf(pri->sys_string + pri->sys_leng,
112                                 "%d", (int)val);
113     else
114         pri->sys_leng += sprintf(pri->sys_string + pri->sys_leng,
115                                 "%ld", val);
116 }
117
118 /*ARGSUSED*/
119 void
120 prt_uns(private_t *pri, int raw, long val)      /* print as unsigned decimal */
121 {
122     GROW(24);
123     if (data_model == PR_MODEL_ILP32)
124         pri->sys_leng += sprintf(pri->sys_string + pri->sys_leng,
```

```

125         "%u", (int)val);
126     else
127         pri->sys_leng += sprintf(pri->sys_string + pri->sys_leng,
128             "%lu", val);
129 }

131 /* print as unsigned decimal, except for -1 */
132 void
133 prt_unl(private_t *pri, int raw, long val)
134 {
135     if ((int)val == -1)
136         prt_dec(pri, raw, val);
137     else
138         prt_uns(pri, raw, val);
139 }

141 /*ARGSUSED*/
142 void
143 prt_oct(private_t *pri, int raw, long val) /* print as octal */
144 {
145     GROW(24);
146     if (data_model == PR_MODEL_ILP32)
147         pri->sys_leng += sprintf(pri->sys_string + pri->sys_leng,
148             "%#o", (int)val);
149     else
150         pri->sys_leng += sprintf(pri->sys_string + pri->sys_leng,
151             "%#lo", val);
152 }

154 /*ARGSUSED*/
155 void
156 prt_hex(private_t *pri, int raw, long val) /* print as hexadecimal */
157 {
158     GROW(20);
159     if (data_model == PR_MODEL_ILP32)
160         pri->sys_leng += sprintf(pri->sys_string + pri->sys_leng,
161             "0x%.8X", (int)val);
162     else
163         pri->sys_leng += sprintf(pri->sys_string + pri->sys_leng,
164             "0x%.8lX", val);
165 }

167 /* print as hexadecimal (half size) */
168 /*ARGSUSED*/
169 void
170 prt_hhx(private_t *pri, int raw, long val)
171 {
172     GROW(20);
173     if (data_model == PR_MODEL_ILP32)
174         pri->sys_leng += sprintf(pri->sys_string + pri->sys_leng,
175             "0x%.4X", (int)val);
176     else
177         pri->sys_leng += sprintf(pri->sys_string + pri->sys_leng,
178             "0x%.4lX", val);
179 }

181 /* print as decimal if small, else hexadecimal */
182 /*ARGSUSED*/
183 void
184 prt_dex(private_t *pri, int raw, long val)
185 {
186     if (val & 0xff000000)
187         prt_hex(pri, 0, val);
188     else
189         prt_dec(pri, 0, val);
190 }

```

```

192 /* print long long offset */
193 /*ARGSUSED*/
194 void
195 prt_lll(private_t *pri, int raw, long val1, long val2)
196 {
197     int hival;
198     int loval;

200 #ifdef LONG_LONG_LTOH
201     hival = (int)val2;
202     loval = (int)val1;
203 #else
204     hival = (int)val1;
205     loval = (int)val2;
206 #endif

208     if (hival == 0) {
209         prt_dex(pri, 0, loval);
210     } else {
211         GROW(18);
212         pri->sys_leng +=
213             sprintf(pri->sys_string + pri->sys_leng, "0x%.8X%.8X",
214                 hival, loval);
215     }
216 }

218 void
219 escape_string(private_t *pri, const char *s)
220 {
221     /*
222      * We want to avoid outputting unprintable characters that may
223      * destroy the user's terminal. So we do one pass to find any
224      * unprintable characters, size the array appropriately, and
225      * then walk each character by hand. Those that are unprintable
226      * are replaced by a hex escape (\xNN). We also escape quotes for
227      * completeness.
228      */
229     int i, unprintable, quotes;
230     size_t len = strlen(s);
231     for (i = 0, unprintable = 0, quotes = 0; i < len; i++) {
232         if (!isprint(s[i]))
233             unprintable++;
234         if (s[i] == '"')
235             quotes++;
236     }

238     GROW(len + 3 * unprintable + quotes + 2);

240     pri->sys_string[pri->sys_leng++] = '"';
241     for (i = 0; i < len; i++) {
242         if (s[i] == '"')
243             pri->sys_string[pri->sys_leng++] = '\\';

245         if (isprint(s[i])) {
246             pri->sys_string[pri->sys_leng++] = s[i];
247         } else {
248             pri->sys_leng += sprintf(pri->sys_string +
249                 pri->sys_leng, "\\x%02x", (uint8_t)s[i]);
250         }
251     }
252     pri->sys_string[pri->sys_leng++] = '"';
253 }

255 void
256 prt_stg(private_t *pri, int raw, long val) /* print as string */

```



```

257 {
258     char *s = raw? NULL : fetchstring(pri, (long)val, PATH_MAX);
260     if (s == NULL)
261         prt_hex(pri, 0, val);
262     else
263         escape_string(pri, s);
264 }
266 /* print as string returned from syscall */
267 void
268 prt_rst(private_t *pri, int raw, long val)
269 {
270     char *s = (raw || pri->Errno)? NULL :
271         fetchstring(pri, (long)val, PATH_MAX);
273     if (s == NULL)
274         prt_hex(pri, 0, val);
275     else {
276         GROW((int)strlen(s) + 2);
277         pri->sys_leng += snprintf(pri->sys_string + pri->sys_leng,
278             pri->sys_ssize - pri->sys_leng, "\"%s\"", s);
279     }
280 }
282 /* print contents of readlink() buffer */
283 void
284 prt_rlk(private_t *pri, int raw, long val)
285 {
286     char *s = (raw || pri->Errno || pri->Rvall <= 0)? NULL :
287         fetchstring(pri, (long)val,
288             (pri->Rvall > PATH_MAX)? PATH_MAX : (int)pri->Rvall);
290     if (s == NULL)
291         prt_hex(pri, 0, val);
292     else {
293         GROW((int)strlen(s) + 2);
294         pri->sys_leng += snprintf(pri->sys_string + pri->sys_leng,
295             pri->sys_ssize - pri->sys_leng, "\"%s\"", s);
296     }
297 }
299 void
300 prt_ioc(private_t *pri, int raw, long val) /* print ioctl code */
301 {
302     const char *s = raw? NULL : ioctlname(pri, (int)val);
304     if (s == NULL)
305         prt_hex(pri, 0, val);
306     else
307         outstring(pri, s);
308 }
310 void
311 prt_ioa(private_t *pri, int raw, long val) /* print ioctl argument */
312 {
313     const char *s;
315     /* cheating -- look at the ioctl() code */
316     switch (pri->sys_args[1]) {
318         /* kstat ioctl()s */
319         case KSTAT_IOC_READ:
320         case KSTAT_IOC_WRITE:
321 #ifdef _LP64
322         if (data_model == PR_MODEL_ILP32)

```

```

323         prt_stg(pri, raw,
324             val + offsetof(kstat32_t, ks_name[0]));
325     else
326 #endif
327         prt_stg(pri, raw,
328             val + offsetof(kstat_t, ks_name[0]));
329     break;
331     /* streams ioctl()s */
332     case I_LOOK:
333         prt_rst(pri, raw, val);
334         break;
335     case I_PUSH:
336     case I_FIND:
337         prt_stg(pri, raw, val);
338         break;
339     case I_LINK:
340     case I_UNLINK:
341     case I_SENDFD:
342         prt_dec(pri, 0, val);
343         break;
344     case I_SRDOPT:
345         if (raw || (s = strropt(val)) == NULL)
346             prt_dec(pri, 0, val);
347         else
348             outstring(pri, s);
349         break;
350     case I_SETSIG:
351         if (raw || (s = strevents(pri, val)) == NULL)
352             prt_hex(pri, 0, val);
353         else
354             outstring(pri, s);
355         break;
356     case I_FLUSH:
357         if (raw || (s = strflush(val)) == NULL)
358             prt_dec(pri, 0, val);
359         else
360             outstring(pri, s);
361         break;
363     /* tty ioctl()s */
364     case TCSBRK:
365     case TCXONC:
366     case TCFLSH:
367     case TCDSET:
368         prt_dec(pri, 0, val);
369         break;
371     default:
372         prt_hex(pri, 0, val);
373         break;
374 }
375 }
377 void
378 prt_pip(private_t *pri, int raw, long val) /* print pipe code */
379 {
380     const char *s = NULL;
382     if (!raw) {
383         switch (val) {
384             case O_CLOEXEC:
385                 s = "O_CLOEXEC";
386                 break;
387             case O_NONBLOCK:
388                 s = "O_NONBLOCK";

```

```

389         break;
390         case O_CLOEXEC|O_NONBLOCK:
391             s = "O_CLOEXEC|O_NONBLOCK";
392             break;
393     }
394 }
396 if (s == NULL)
397     prt_dex(pri, 0, val);
398 else
399     outstring(pri, s);
400 }

402 void
403 prt_pfd(private_t *pri, int raw, long val)    /* print pipe code */
404 {
405     int fds[2];
406     char str[32];

408     /* the fds only have meaning if the return value is 0 */
409     if (!raw &&
410         pri->Rvall >= 0 &&
411         Pread(Proc, fds, sizeof (fds), (long)val) == sizeof (fds)) {
412         (void) snprintf(str, sizeof (str), "[%d,%d]", fds[0], fds[1]);
413         outstring(pri, str);
414     } else {
415         prt_hex(pri, 0, val);
416     }
417 }

419 void
420 prt_fcn(private_t *pri, int raw, long val)    /* print fcntl code */
421 {
422     const char *s = raw? NULL : fcntlname(val);

424     if (s == NULL)
425         prt_dec(pri, 0, val);
426     else
427         outstring(pri, s);
428 }

430 void
431 prt_s86(private_t *pri, int raw, long val)    /* print sysi86 code */
432 {
434     const char *s = raw? NULL : si86name(val);

436     if (s == NULL)
437         prt_dec(pri, 0, val);
438     else
439         outstring(pri, s);
440 }

442 void
443 prt_uts(private_t *pri, int raw, long val)    /* print utssys code */
444 {
445     const char *s = raw? NULL : utscode(val);

447     if (s == NULL)
448         prt_dec(pri, 0, val);
449     else
450         outstring(pri, s);
451 }

453 void
454 prt_msc(private_t *pri, int raw, long val)    /* print msgsys command */

```

```

455 {
456     const char *s = raw? NULL : msgcmd(val);

458     if (s == NULL)
459         prt_dec(pri, 0, val);
460     else
461         outstring(pri, s);
462 }

464 void
465 prt_msf(private_t *pri, int raw, long val)    /* print msgsys flags */
466 {
467     const char *s = raw? NULL : msgflags(pri, (int)val);

469     if (s == NULL)
470         prt_oct(pri, 0, val);
471     else
472         outstring(pri, s);
473 }

475 void
476 prt_smc(private_t *pri, int raw, long val)    /* print semsys command */
477 {
478     const char *s = raw? NULL : semcmd(val);

480     if (s == NULL)
481         prt_dec(pri, 0, val);
482     else
483         outstring(pri, s);
484 }

486 void
487 prt_sef(private_t *pri, int raw, long val)    /* print semsys flags */
488 {
489     const char *s = raw? NULL : semflags(pri, (int)val);

491     if (s == NULL)
492         prt_oct(pri, 0, val);
493     else
494         outstring(pri, s);
495 }

497 void
498 prt_shc(private_t *pri, int raw, long val)    /* print shmsys command */
499 {
500     const char *s = raw? NULL : shmcmd(val);

502     if (s == NULL)
503         prt_dec(pri, 0, val);
504     else
505         outstring(pri, s);
506 }

508 void
509 prt_shf(private_t *pri, int raw, long val)    /* print shmsys flags */
510 {
511     const char *s = raw? NULL : shmflags(pri, (int)val);

513     if (s == NULL)
514         prt_oct(pri, 0, val);
515     else
516         outstring(pri, s);
517 }

519 void
520 prt_sfs(private_t *pri, int raw, long val)    /* print sysfs code */

```

```

521 {
522     const char *s = raw? NULL : sfsname(val);

524     if (s == NULL)
525         prt_dec(pri, 0, val);
526     else
527         outstring(pri, s);
528 }

530 void
531 prt_opn(private_t *pri, int raw, long val) /* print open code */
532 {
533     const char *s = raw? NULL : openarg(pri, val);

535     if (s == NULL)
536         prt_oct(pri, 0, val);
537     else
538         outstring(pri, s);
539 }

541 void
542 prt_sig(private_t *pri, int raw, long val) /* print signal name */
543 {
544     const char *s = raw? NULL : signame(pri, (int)val);

546     if (s == NULL)
547         prt_hex(pri, 0, val);
548     else
549         outstring(pri, s);
550 }

552 void
553 prt_smf(private_t *pri, int raw, long val) /* print streams message flags */
554 {
555     switch (val) {
556     case 0:
557         prt_dec(pri, 0, val);
558         break;
559     case RS_HIPRI:
560         if (raw)
561             prt_hhx(pri, 0, val);
562         else
563             outstring(pri, "RS_HIPRI");
564         break;
565     default:
566         prt_hhx(pri, 0, val);
567         break;
568     }
569 }

571 void
572 prt_mtf(private_t *pri, int raw, long val) /* print mount flags */
573 {
574     const char *s = raw? NULL : mountflags(pri, val);

576     if (s == NULL)
577         prt_hex(pri, 0, val);
578     else
579         outstring(pri, s);
580 }

582 void
583 prt_mft(private_t *pri, int raw, long val) /* print mount file system type */
584 {
585     if (val >= 0 && val < 256)
586         prt_dec(pri, 0, val);

```

```

587     else if (raw)
588         prt_hex(pri, 0, val);
589     else
590         prt_stg(pri, raw, val);
591 }

593 #define ISREAD(code) \
594 ((code) == SYS_read || (code) == SYS_pread || (code) == SYS_pread64 || \
595 (code) == SYS_recv || (code) == SYS_recvfrom)
596 #define ISWRITE(code) \
597 ((code) == SYS_write || (code) == SYS_pwrite || \
598 (code) == SYS_pwrite64 || (code) == SYS_send || (code) == SYS_sendto)

600 /* print contents of read() or write() I/O buffer */
601 void
602 prt_iob(private_t *pri, int raw, long val)
603 {
604     const lwpstatus_t *Lsp = pri->lwpstat;
605     int syscall = Lsp->pr_what;
606     int fdpl = pri->sys_args[0] + 1;
607     ssize_t nbyte = ISWRITE(syscall)? pri->sys_args[2] :
608         (pri->Errno? 0 : pri->Rvall);
609     int elsewhere = FALSE; /* TRUE iff dumped elsewhere */
610     char buffer[IOBSIZE];

612     pri->iob_buf[0] = '\0';

614     if (Lsp->pr_why == PR_SYSEXIT && nbyte > IOBSIZE) {
615         if (ISREAD(syscall))
616             elsewhere = prismember(&readfd, fdpl);
617         else
618             elsewhere = prismember(&writefd, fdpl);
619     }

621     if (nbyte <= 0 || elsewhere)
622         prt_hex(pri, 0, val);
623     else {
624         int nb = nbyte > IOBSIZE? IOBSIZE : (int)nbyte;

626         if (Pread(Proc, buffer, (size_t)nb, (long)val) != nb)
627             prt_hex(pri, 0, val);
628         else {
629             pri->iob_buf[0] = '';
630             showbytes(buffer, nb, pri->iob_buf + 1);
631             (void) strlcat(pri->iob_buf,
632                 (nb == nbyte)?
633                 (const char *)"\n" : (const char *)"\n..",
634                 sizeof (pri->iob_buf));
635             if (raw)
636                 prt_hex(pri, 0, val);
637             else
638                 outstring(pri, pri->iob_buf);
639         }
640     }
641 }
642 #undef ISREAD
643 #undef ISWRITE

645 void
646 prt_idt(private_t *pri, int raw, long val) /* print idtype_t, waitid() arg */
647 {
648     const char *s = raw? NULL : idtype_enum(pri, val);

650     if (s == NULL)
651         prt_dec(pri, 0, val);
652     else

```

```

653         outstring(pri, s);
654     }

656 void
657 prt_wop(private_t *pri, int raw, long val) /* print waitid() options */
658 {
659     const char *s = raw? NULL : woptions(pri, (int)val);

661     if (s == NULL)
662         prt_oct(pri, 0, val);
663     else
664         outstring(pri, s);
665 }

667 void
668 prt_whn(private_t *pri, int raw, long val) /* print lseek() whence argument */
669 {
670     const char *s = raw? NULL : whencearg(val);

672     if (s == NULL)
673         prt_dec(pri, 0, val);
674     else
675         outstring(pri, s);
676 }

678 /*ARGSUSED*/
679 void
680 prt_spm(private_t *pri, int raw, long val) /* print sigprocmask argument */
681 {
682     const char *s = NULL;

684     if (!raw) {
685         switch (val) {
686             case SIG_BLOCK:      s = "SIG_BLOCK";      break;
687             case SIG_UNBLOCK:    s = "SIG_UNBLOCK";    break;
688             case SIG_SETMASK:    s = "SIG_SETMASK";    break;
689         }
690     }

692     if (s == NULL)
693         prt_dec(pri, 0, val);
694     else
695         outstring(pri, s);
696 }

698 const char *
699 mmap_protect(private_t *pri, long arg)
700 {
701     char *str = pri->code_buf;

703     if (arg & ~(PROT_READ|PROT_WRITE|PROT_EXEC))
704         return ((char *)NULL);

706     if (arg == PROT_NONE)
707         return ("PROT_NONE");

709     *str = '\0';
710     if (arg & PROT_READ)
711         (void) strlcat(str, "|PROT_READ", sizeof (pri->code_buf));
712     if (arg & PROT_WRITE)
713         (void) strlcat(str, "|PROT_WRITE", sizeof (pri->code_buf));
714     if (arg & PROT_EXEC)
715         (void) strlcat(str, "|PROT_EXEC", sizeof (pri->code_buf));
716     return ((const char *) (str + 1));
717 }

```

```

719 const char *
720 mmap_type(private_t *pri, long arg)
721 {
722     char *str = pri->code_buf;
723     size_t used;

725 #define CBSIZE  sizeof (pri->code_buf)
726     switch (arg & MAP_TYPE) {
727     case MAP_SHARED:
728         used = strlcpy(str, "MAP_SHARED", CBSIZE);
729         break;
730     case MAP_PRIVATE:
731         used = strlcpy(str, "MAP_PRIVATE", CBSIZE);
732         break;
733     default:
734         used = snprintf(str, CBSIZE, "%ld", arg&MAP_TYPE);
735         break;
736     }

738     arg &= ~(_MAP_NEW|MAP_TYPE);

740     if (arg & ~(MAP_FIXED|MAP_RENAME|MAP_NORESERVE|MAP_ANON|MAP_ALIGN|
741               MAP_TEXT|MAP_INITDATA|MAP_32BIT))
742         (void) snprintf(str + used, sizeof (pri->code_buf) - used,
743                        "|0x%lx", arg);
744     else {
745         if (arg & MAP_FIXED)
746             (void) strlcat(str, "|MAP_FIXED", CBSIZE);
747         if (arg & MAP_RENAME)
748             (void) strlcat(str, "|MAP_RENAME", CBSIZE);
749         if (arg & MAP_NORESERVE)
750             (void) strlcat(str, "|MAP_NORESERVE", CBSIZE);
751         if (arg & MAP_ANON)
752             (void) strlcat(str, "|MAP_ANON", CBSIZE);
753         if (arg & MAP_ALIGN)
754             (void) strlcat(str, "|MAP_ALIGN", CBSIZE);
755         if (arg & MAP_TEXT)
756             (void) strlcat(str, "|MAP_TEXT", CBSIZE);
757         if (arg & MAP_INITDATA)
758             (void) strlcat(str, "|MAP_INITDATA", CBSIZE);
759         if (arg & MAP_32BIT)
760             (void) strlcat(str, "|MAP_32BIT", CBSIZE);
761     }

763     return ((const char *)str);
764 #undef CBSIZE
765 }

767 void
768 prt_mpr(private_t *pri, int raw, long val) /* print mmap()/mprotect() flags */
769 {
770     const char *s = raw? NULL : mmap_protect(pri, val);

772     if (s == NULL)
773         prt_hhx(pri, 0, val);
774     else
775         outstring(pri, s);
776 }

778 void
779 prt_mty(private_t *pri, int raw, long val) /* print mmap() mapping type flags */
780 {
781     const char *s = raw? NULL : mmap_type(pri, val);

783     if (s == NULL)
784         prt_hhx(pri, 0, val);

```

```

785     else
786         outstring(pri, s);
787 }

789 void
790 prt_mob(private_t *pri, int raw, long val) /* print mmapobj() flags */
791 {
792     if (val == 0)
793         prt_dec(pri, 0, val);
794     else if (raw || (val & ~(MMOBJ_PADDING|MMOBJ_INTERPRET)) != 0)
795         prt_hhx(pri, 0, val);
796     else {
797 #define CBSIZE    sizeof (pri->code_buf)
798         char *s = pri->code_buf;

800         *s = '\0';
801         if (val & MMOBJ_PADDING)
802             (void) strcat(s, "|MMOBJ_PADDING", CBSIZE);
803         if (val & MMOBJ_INTERPRET)
804             (void) strcat(s, "|MMOBJ_INTERPRET", CBSIZE);
805         outstring(pri, s + 1);
806 #undef CBSIZE
807     }
808 }

810 /*ARGSUSED*/
811 void
812 prt_mcf(private_t *pri, int raw, long val) /* print memcntl() function */
813 {
814     const char *s = NULL;

816     if (!raw) {
817         switch (val) {
818             case MC_SYNC:          s = "MC_SYNC";          break;
819             case MC_LOCK:         s = "MC_LOCK";          break;
820             case MC_UNLOCK:       s = "MC_UNLOCK";        break;
821             case MC_ADVISE:       s = "MC_ADVISE";        break;
822             case MC_LOCKAS:       s = "MC_LOCKAS";        break;
823             case MC_UNLOCKAS:     s = "MC_UNLOCKAS";      break;
824             case MC_HAT_ADVISE:   s = "MC_HAT_ADVISE";    break;
825         }
826     }

828     if (s == NULL)
829         prt_dec(pri, 0, val);
830     else
831         outstring(pri, s);
832 }

834 void
835 prt_mad(private_t *pri, int raw, long val) /* print madvise() argument */
836 {
837     const char *s = NULL;

839     if (!raw) {
840         switch (val) {
841             case MADV_NORMAL:      s = "MADV_NORMAL";      break;
842             case MADV_RANDOM:     s = "MADV_RANDOM";      break;
843             case MADV_SEQUENTIAL: s = "MADV_SEQUENTIAL";  break;
844             case MADV_WILLNEED:   s = "MADV_WILLNEED";   break;
845             case MADV_DONTNEED:   s = "MADV_DONTNEED";   break;
846             case MADV_FREE:       s = "MADV_FREE";        break;
847             case MADV_ACCESS_DEFAULT: s = "MADV_ACCESS_DEFAULT"; break;
848             case MADV_ACCESS_LWP: s = "MADV_ACCESS_LWP";  break;
849             case MADV_ACCESS_MANY: s = "MADV_ACCESS_MANY"; break;
850             case MADV_PURGE:      s = "MADV_PURGE";       break;

```

```

851     }
852 }

854     if (s == NULL)
855         prt_dec(pri, 0, val);
856     else
857         outstring(pri, s);
858 }

860 void
861 prt_mc4(private_t *pri, int raw, long val) /* print memcntl() (4th) argument */
862 {
863     if (val == 0)
864         prt_dec(pri, 0, val);
865     else if (raw)
866         prt_hhx(pri, 0, val);
867     else {
868         char *s = NULL;

870 #define CBSIZE    sizeof (pri->code_buf)
871         /* cheating -- look at memcntl func */
872         switch (pri->sys_args[2]) {
873             case MC_ADVISE:
874                 prt_mad(pri, 0, val);
875                 return;

877             case MC_SYNC:
878                 if ((val & ~(MS_SYNC|MS_ASYNC|MS_INVALIDATE)) == 0) {
879                     *(s = pri->code_buf) = '\0';
880                     if (val & MS_SYNC)
881                         (void) strcat(s, "|MS_SYNC", CBSIZE);
882                     if (val & MS_ASYNC)
883                         (void) strcat(s, "|MS_ASYNC", CBSIZE);
884                     if (val & MS_INVALIDATE)
885                         (void) strcat(s, "|MS_INVALIDATE",
886                                     CBSIZE);
887                 }
888                 break;

890             case MC_LOCKAS:
891             case MC_UNLOCKAS:
892                 if ((val & ~(MCL_CURRENT|MCL_FUTURE)) == 0) {
893                     *(s = pri->code_buf) = '\0';
894                     if (val & MCL_CURRENT)
895                         (void) strcat(s, "|MCL_CURRENT",
896                                     CBSIZE);
897                     if (val & MCL_FUTURE)
898                         (void) strcat(s, "|MCL_FUTURE",
899                                     CBSIZE);
900                 }
901                 break;
902             }
903 #undef CBSIZE

905         if (s == NULL || *s == '\0')
906             prt_hhx(pri, 0, val);
907         else
908             outstring(pri, ++s);
909     }
910 }

912 void
913 prt_mc5(private_t *pri, int raw, long val) /* print memcntl() (5th) argument */
914 {
915     char *s;

```

```

917 #define CBSIZE sizeof (pri->code_buf)
918     if (val == 0)
919         prt_dec(pri, 0, val);
920     else if (raw || (val & ~VALID_ATTR))
921         prt_hhx(pri, 0, val);
922     else {
923         s = pri->code_buf;
924         *s = '\0';
925         if (val & SHARED)
926             (void) strlcat(s, "|SHARED", CBSIZE);
927         if (val & PRIVATE)
928             (void) strlcat(s, "|PRIVATE", CBSIZE);
929         if (val & PROT_READ)
930             (void) strlcat(s, "|PROT_READ", CBSIZE);
931         if (val & PROT_WRITE)
932             (void) strlcat(s, "|PROT_WRITE", CBSIZE);
933         if (val & PROT_EXEC)
934             (void) strlcat(s, "|PROT_EXEC", CBSIZE);
935         if (*s == '\0')
936             prt_hhx(pri, 0, val);
937         else
938             outstring(pri, ++s);
939     }
940 #undef CBSIZE
941 }
942
943 void
944 prt_ulm(private_t *pri, int raw, long val) /* print ulimit() argument */
945 {
946     const char *s = NULL;
947
948     if (!raw) {
949         switch (val) {
950             case UL_GFILLIM:      s = "UL_GFILLIM";      break;
951             case UL_SFILLIM:      s = "UL_SFILLIM";      break;
952             case UL_GMEMLIM:      s = "UL_GMEMLIM";      break;
953             case UL_GDESLIM:      s = "UL_GDESLIM";      break;
954         }
955     }
956
957     if (s == NULL)
958         prt_dec(pri, 0, val);
959     else
960         outstring(pri, s);
961 }
962
963 void
964 prt_rlm(private_t *pri, int raw, long val) /* print get/setrlimit() argument */
965 {
966     const char *s = NULL;
967
968     if (!raw) {
969         switch (val) {
970             case RLIMIT_CPU:      s = "RLIMIT_CPU";      break;
971             case RLIMIT_FSIZE:    s = "RLIMIT_FSIZE";    break;
972             case RLIMIT_DATA:     s = "RLIMIT_DATA";     break;
973             case RLIMIT_STACK:    s = "RLIMIT_STACK";    break;
974             case RLIMIT_CORE:     s = "RLIMIT_CORE";     break;
975             case RLIMIT_NOFILE:   s = "RLIMIT_NOFILE";   break;
976             case RLIMIT_VMEM:     s = "RLIMIT_VMEM";     break;
977         }
978     }
979
980     if (s == NULL)
981         prt_dec(pri, 0, val);
982     else

```

```

983         outstring(pri, s);
984     }
985
986 void
987 prt_cnf(private_t *pri, int raw, long val) /* print sysconfig code */
988 {
989     const char *s = raw? NULL : sconfname(val);
990
991     if (s == NULL)
992         prt_dec(pri, 0, val);
993     else
994         outstring(pri, s);
995 }
996
997 void
998 prt_inf(private_t *pri, int raw, long val) /* print sysinfo code */
999 {
1000     const char *s = NULL;
1001
1002     if (!raw) {
1003         switch (val) {
1004             case SI_SYSNAME:      s = "SI_SYSNAME";      break;
1005             case SI_HOSTNAME:     s = "SI_HOSTNAME";     break;
1006             case SI_RELEASE:      s = "SI_RELEASE";      break;
1007             case SI_VERSION:      s = "SI_VERSION";      break;
1008             case SI_MACHINE:      s = "SI_MACHINE";      break;
1009             case SI_ARCHITECTURE: s = "SI_ARCHITECTURE"; break;
1010             case SI_ARCHITECTURE_32: s = "SI_ARCHITECTURE_32"; break;
1011             case SI_ARCHITECTURE_64: s = "SI_ARCHITECTURE_64"; break;
1012             case SI_ARCHITECTURE_K: s = "SI_ARCHITECTURE_K"; break;
1013             case SI_HW_SERIAL:    s = "SI_HW_SERIAL";    break;
1014             case SI_HW_PROVIDER:  s = "SI_HW_PROVIDER";  break;
1015             case SI_SRPC_DOMAIN:  s = "SI_SRPC_DOMAIN";  break;
1016             case SI_SET_HOSTNAME: s = "SI_SET_HOSTNAME"; break;
1017             case SI_SET_SRPC_DOMAIN: s = "SI_SET_SRPC_DOMAIN"; break;
1018             case SI_PLATFORM:    s = "SI_PLATFORM";    break;
1019             case SI_ISALIST:      s = "SI_ISALIST";      break;
1020             case SI_DHCP_CACHE:   s = "SI_DHCP_CACHE";   break;
1021         }
1022     }
1023
1024     if (s == NULL)
1025         prt_dec(pri, 0, val);
1026     else
1027         outstring(pri, s);
1028 }
1029
1030 void
1031 prt_ptc(private_t *pri, int raw, long val) /* print pathconf code */
1032 {
1033     const char *s = raw? NULL : pathconfname(val);
1034
1035     if (s == NULL)
1036         prt_dec(pri, 0, val);
1037     else
1038         outstring(pri, s);
1039 }
1040
1041 void
1042 prt_fui(private_t *pri, int raw, long val) /* print fusers() input argument */
1043 {
1044     const char *s = raw? NULL : fuiname(val);
1045
1046     if (s == NULL)
1047         prt_hhx(pri, 0, val);
1048     else

```

```

1049         outstring(pri, s);
1050     }

1052 void
1053 prt_lwf(private_t *pri, int raw, long val) /* print lwp_create() flags */
1054 {
1055     char *s;

1057     if (val == 0)
1058         prt_dec(pri, 0, val);
1059     else if (raw ||
1060             (val & ~(LWP_DAEMON|LWP_DETACHED|LWP_SUSPENDED)))
1061         prt_hhx(pri, 0, val);
1062     else {
1063 #define CBSIZE    sizeof (pri->code_buf)
1064                 s = pri->code_buf;
1065                 *s = '\0';
1066                 if (val & LWP_DAEMON)
1067                     (void) strlcat(s, "|LWP_DAEMON", CBSIZE);
1068                 if (val & LWP_DETACHED)
1069                     (void) strlcat(s, "|LWP_DETACHED", CBSIZE);
1070                 if (val & LWP_SUSPENDED)
1071                     (void) strlcat(s, "|LWP_SUSPENDED", CBSIZE);
1072                 outstring(pri, ++s);
1073 #undef CBSIZE
1074             }
1075     }

1077 void
1078 prt_itm(private_t *pri, int raw, long val) /* print [get|set|itimer() arg */
1079 {
1080     const char *s = NULL;

1082     if (!raw) {
1083         switch (val) {
1084             case ITIMER_REAL:      s = "ITIMER_REAL";      break;
1085             case ITIMER_VIRTUAL:  s = "ITIMER_VIRTUAL";    break;
1086             case ITIMER_PROF:     s = "ITIMER_PROF";       break;
1087 #ifndef ITIMER_REALPROF
1088             case ITIMER_REALPROF: s = "ITIMER_REALPROF";  break;
1089 #endif
1090         }
1091     }

1093     if (s == NULL)
1094         prt_dec(pri, 0, val);
1095     else
1096         outstring(pri, s);
1097 }

1099 void
1100 prt_mod(private_t *pri, int raw, long val) /* print modctl() code */
1101 {
1102     const char *s = NULL;

1104     if (!raw) {
1105         switch (val) {
1106             case MODLOAD:          s = "MODLOAD";          break;
1107             case MODUNLOAD:       s = "MODUNLOAD";        break;
1108             case MODINFO:         s = "MODINFO";          break;
1109             case MODRESERVED:     s = "MODRESERVED";      break;
1110             case MODSETMINIROOT:  s = "MODSETMINIROOT";   break;
1111             case MODADDMAJBIND:   s = "MODADDMAJBIND";   break;
1112             case MODGETPATH:     s = "MODGETPATH";       break;
1113             case MODGETPATHLEN:  s = "MODGETPATHLEN";    break;
1114             case MODREADSYSBIND:  s = "MODREADSYSBIND";  break;

```

```

1115         case MODGETMAJBIND:      s = "MODGETMAJBIND";    break;
1116         case MODGETNAME:        s = "MODGETNAME";        break;
1117         case MODSIZEOF_DEVID:   s = "MODSIZEOF_DEVID";   break;
1118         case MODGETDEVID:       s = "MODGETDEVID";       break;
1119         case MODSIZEOF_MINORNAME: s = "MODSIZEOF_MINORNAME"; break;
1120         case MODGETMINORNAME:   s = "MODGETMINORNAME";   break;
1121         case MODGETFBNAME:      s = "MODGETFBNAME";      break;
1122         case MODEVENTS:         s = "MODEVENTS";         break;
1123         case MODREREADDACF:     s = "MODREREADDACF";     break;
1124         case MODLOADDRVCONF:    s = "MODLOADDRVCONF";    break;
1125         case MODUNLOADDRVCONF:  s = "MODUNLOADDRVCONF";  break;
1126         case MODREMMAJBIND:     s = "MODREMMAJBIND";     break;
1127         case MODDEVT2INSTANCE:  s = "MODDEVT2INSTANCE";  break;
1128         case MODGETDEVFSPATH_LEN: s = "MODGETDEVFSPATH_LEN"; break;
1129         case MODGETDEVFSPATH:   s = "MODGETDEVFSPATH";   break;
1130         case MODDEVID2PATHS:    s = "MODDEVID2PATHS";    break;
1131         case MODSETDEVPOLICY:   s = "MODSETDEVPOLICY";   break;
1132         case MODGETDEVPOLICY:   s = "MODGETDEVPOLICY";   break;
1133         case MODALLOCPRIV:      s = "MODALLOCPRIV";      break;
1134         case MODGETDEVPOLICYBYNAME:
1135             s = "MODGETDEVPOLICYBYNAME"; break;
1136         case MODLOADMINORPERM:  s = "MODLOADMINORPERM";  break;
1137         case MODADDMINORPERM:   s = "MODADDMINORPERM";   break;
1138         case MODREMMINORPERM:  s = "MODREMMINORPERM";   break;
1139         case MODREMDRVCLEANUP:  s = "MODREMDRVCLEANUP";  break;
1140         case MODDEVEXISTS:      s = "MODDEVEXISTS";      break;
1141         case MODDEVREADDIR:     s = "MODDEVREADDIR";     break;
1142         case MODDEVEMPTYDIR:   s = "MODDEVEMPTYDIR";    break;
1143         case MODDEVNAME:        s = "MODDEVNAME";        break;
1144         case MODGETDEVFSPATH_MI_LEN:
1145             s = "MODGETDEVFSPATH_MI_LEN"; break;
1146         case MODGETDEVFSPATH_MI:
1147             s = "MODGETDEVFSPATH_MI"; break;
1148         case MODREMDRVALIAS:    s = "MODREMDRVALIAS";    break;
1149         case MODHPOPS:          s = "MODHPOPS";          break;
1150     }
1151 }

1153     if (s == NULL)
1154         prt_dec(pri, 0, val);
1155     else
1156         outstring(pri, s);
1157 }

1159 void
1160 prt_acl(private_t *pri, int raw, long val) /* print acl() code */
1161 {
1162     const char *s = NULL;

1164     if (!raw) {
1165         switch (val) {
1166             case GETACL:          s = "GETACL";          break;
1167             case SETACL:         s = "SETACL";         break;
1168             case GETACLCNT:      s = "GETACLCNT";      break;
1169             case ACE_GETACL:     s = "ACE_GETACL";     break;
1170             case ACE_SETACL:     s = "ACE_SETACL";     break;
1171             case ACE_GETACLCNT:  s = "ACE_GETACLCNT";  break;
1172         }
1173     }

1175     if (s == NULL)
1176         prt_dec(pri, 0, val);
1177     else
1178         outstring(pri, s);
1179 }

```

```

1181 void
1182 prt_aio(private_t *pri, int raw, long val)    /* print kaio() code */
1183 {
1184     const char *s = NULL;
1185     char buf[32];
1186
1187     if (!raw) {
1188         switch (val & ~AIO_POLL_BIT) {
1189             case AIOREAD:      s = "AIOREAD";      break;
1190             case AIOWRITE:     s = "AIOWRITE";     break;
1191             case AIOWAIT:      s = "AIOWAIT";      break;
1192             case AIOCANCEL:    s = "AIOCANCEL";    break;
1193             case AIONOTIFY:    s = "AIONOTIFY";    break;
1194             case AIOINIT:      s = "AIOINIT";      break;
1195             case AIOSTART:     s = "AIOSTART";     break;
1196             case AIOLIO:       s = "AIOLIO";       break;
1197             case AIOSUSPEND:   s = "AIOSUSPEND";   break;
1198             case AIOERROR:     s = "AIOERROR";     break;
1199             case AIOLIOWAIT:   s = "AIOLIOWAIT";   break;
1200             case AIOAREAD:     s = "AIOAREAD";     break;
1201             case AIOAWRITE:    s = "AIOAWRITE";    break;
1202             /*
1203              * We have to hardcode the values for the 64-bit versions of
1204              * these calls, because <sys/aio.h> defines them to be identical
1205              * when compiled 64-bit.  If our target is 32-bit, we still need
1206              * to decode them correctly.
1207              */
1208             case 13:           s = "AIOLIO64";      break;
1209             case 14:           s = "AIOSUSPEND64";  break;
1210             case 15:           s = "AIOERROR64";   break;
1211             case 16:           s = "AIOLIOWAIT64";  break;
1212             case 17:           s = "AIOAREAD64";   break;
1213             case 18:           s = "AIOAWRITE64";   break;
1214             case 19:           s = "AIOCANCEL64";   break;
1215
1216             /*
1217              * AIOFSYNC doesn't correspond to a syscall.
1218              */
1219             case AIOWAITN:     s = "AIOWAITN";     break;
1220         }
1221         if (s != NULL && (val & AIO_POLL_BIT)) {
1222             (void) strcpy(buf, s, sizeof(buf));
1223             (void) strcat(buf, "|AIO_POLL_BIT", sizeof(buf));
1224             s = (const char *)buf;
1225         }
1226     }
1227
1228     if (s == NULL)
1229         prt_dec(pri, 0, val);
1230     else
1231         outstring(pri, s);
1232 }
1233
1234 void
1235 prt_aud(private_t *pri, int raw, long val)    /* print auditsys() code */
1236 {
1237     const char *s = NULL;
1238
1239     if (!raw) {
1240         switch (val) {
1241             case BSM_GETAUD:   s = "BSM_GETAUD";   break;
1242             case BSM_SETAUD:   s = "BSM_SETAUD";   break;
1243             case BSM_GETAUDIT: s = "BSM_GETAUDIT"; break;
1244             case BSM_SETAUDIT: s = "BSM_SETAUDIT"; break;
1245             case BSM_AUDIT:    s = "BSM_AUDIT";    break;
1246             case BSM_AUDITCTL: s = "BSM_AUDITCTL"; break;

```

```

1247         case BSM_GETAUDIT_ADDR: s = "BSM_GETAUDIT_ADDR"; break;
1248         case BSM_SETAUDIT_ADDR: s = "BSM_SETAUDIT_ADDR"; break;
1249     }
1250 }
1251
1252     if (s == NULL)
1253         prt_dec(pri, 0, val);
1254     else
1255         outstring(pri, s);
1256 }
1257
1258 void
1259 prt_cor(private_t *pri, int raw, long val)    /* print corectl() subcode */
1260 {
1261     const char *s = NULL;
1262
1263     if (!raw) {
1264         switch (val) {
1265             case CC_SET_OPTIONS:
1266                 s = "CC_SET_OPTIONS";      break;
1267             case CC_GET_OPTIONS:
1268                 s = "CC_GET_OPTIONS";      break;
1269             case CC_SET_GLOBAL_PATH:
1270                 s = "CC_SET_GLOBAL_PATH";  break;
1271             case CC_GET_GLOBAL_PATH:
1272                 s = "CC_GET_GLOBAL_PATH";  break;
1273             case CC_SET_PROCESS_PATH:
1274                 s = "CC_SET_PROCESS_PATH"; break;
1275             case CC_GET_PROCESS_PATH:
1276                 s = "CC_GET_PROCESS_PATH"; break;
1277             case CC_SET_GLOBAL_CONTENT:
1278                 s = "CC_SET_GLOBAL_CONTENT"; break;
1279             case CC_GET_GLOBAL_CONTENT:
1280                 s = "CC_GET_GLOBAL_CONTENT"; break;
1281             case CC_SET_PROCESS_CONTENT:
1282                 s = "CC_SET_PROCESS_CONTENT"; break;
1283             case CC_GET_PROCESS_CONTENT:
1284                 s = "CC_GET_PROCESS_CONTENT"; break;
1285             case CC_SET_DEFAULT_PATH:
1286                 s = "CC_SET_DEFAULT_PATH";  break;
1287             case CC_GET_DEFAULT_PATH:
1288                 s = "CC_GET_DEFAULT_PATH";  break;
1289             case CC_SET_DEFAULT_CONTENT:
1290                 s = "CC_SET_DEFAULT_CONTENT"; break;
1291             case CC_GET_DEFAULT_CONTENT:
1292                 s = "CC_GET_DEFAULT_CONTENT"; break;
1293         }
1294     }
1295
1296     if (s == NULL)
1297         prt_dec(pri, 0, val);
1298     else
1299         outstring(pri, s);
1300 }
1301
1302 void
1303 prt_cco(private_t *pri, int raw, long val)    /* print corectl() options */
1304 {
1305     char *s;
1306
1307     if (val == 0)
1308         prt_dec(pri, 0, val);
1309     else if (raw || (val & ~CC_OPTIONS))
1310         prt_hhx(pri, 0, val);
1311     else {
1312 #define CBSIZE    sizeof(pri->code_buf)

```



```

1313     s = pri->code_buf;
1314     *s = '\0';
1315     if (val & CC_GLOBAL_PATH)
1316         (void) strlcat(s, "|CC_GLOBAL_PATH", CBSIZE);
1317     if (val & CC_PROCESS_PATH)
1318         (void) strlcat(s, "|CC_PROCESS_PATH", CBSIZE);
1319     if (val & CC_GLOBAL_SETID)
1320         (void) strlcat(s, "|CC_GLOBAL_SETID", CBSIZE);
1321     if (val & CC_PROCESS_SETID)
1322         (void) strlcat(s, "|CC_PROCESS_SETID", CBSIZE);
1323     if (val & CC_GLOBAL_LOG)
1324         (void) strlcat(s, "|CC_GLOBAL_LOG", CBSIZE);
1325     if (*s == '\0')
1326         prt_hhx(pri, 0, val);
1327     else
1328         outstring(pri, ++s);
1329 #undef CBSIZE
1330 }
1331 }
1332
1333 void
1334 prt_ccc(private_t *pri, int raw, long val) /* print corectl() content */
1335 {
1336     core_content_t ccc;
1337
1338     if (Pread(Proc, &ccc, sizeof(ccc), val) != sizeof(ccc))
1339         prt_hex(pri, 0, val);
1340     else if (!raw && proc_content2str(ccc, pri->code_buf,
1341         sizeof(pri->code_buf)) >= 0)
1342         outstring(pri, pri->code_buf);
1343     else
1344         prt_hhx(pri, 0, (long)ccc);
1345 }
1346
1347 void
1348 prt_rcc(private_t *pri, int raw, long val) /* print corectl() ret. cont. */
1349 {
1350     core_content_t ccc;
1351
1352     if (pri->Errno || Pread(Proc, &ccc, sizeof(ccc), val) != sizeof(ccc))
1353         prt_hex(pri, 0, val);
1354     else if (!raw && proc_content2str(ccc, pri->code_buf,
1355         sizeof(pri->code_buf)) >= 0)
1356         outstring(pri, pri->code_buf);
1357     else
1358         prt_hhx(pri, 0, (long)ccc);
1359 }
1360
1361 void
1362 prt_cpc(private_t *pri, int raw, long val) /* print cpc() subcode */
1363 {
1364     const char *s = NULL;
1365
1366     if (!raw) {
1367         switch (val) {
1368             case CPC_BIND:          s = "CPC_BIND";          break;
1369             case CPC_SAMPLE:        s = "CPC_SAMPLE";        break;
1370             case CPC_INVALIDATE:    s = "CPC_INVALIDATE";    break;
1371             case CPC_RELE:          s = "CPC_RELE";          break;
1372             case CPC_EVLIST_SIZE:   s = "CPC_EVLIST_SIZE";   break;
1373             case CPC_LIST_EVENTS:   s = "CPC_LIST_EVENTS";   break;
1374             case CPC_ATTRLIST_SIZE: s = "CPC_ATTRLIST_SIZE"; break;
1375             case CPC_LIST_ATTRS:    s = "CPC_LIST_ATTRS";    break;
1376             case CPC_IMPL_NAME:     s = "CPC_IMPL_NAME";     break;
1377             case CPC_CPUREF:        s = "CPC_CPUREF";        break;
1378             case CPC_USR_EVENTS:    s = "CPC_USR_EVENTS";    break;

```

```

1379             case CPC_SYS_EVENTS:   s = "CPC_SYS_EVENTS";   break;
1380             case CPC_NPIC:         s = "CPC_NPIC";         break;
1381             case CPC_CAPS:         s = "CPC_CAPS";         break;
1382             case CPC_ENABLE:       s = "CPC_ENABLE";       break;
1383             case CPC_DISABLE:      s = "CPC_DISABLE";      break;
1384         }
1385     }
1386
1387     if (s == NULL)
1388         prt_dec(pri, 0, val);
1389     else
1390         outstring(pri, s);
1391 }
1392
1393 void
1394 outstring(private_t *pri, const char *s)
1395 {
1396     int len = strlen(s);
1397
1398     GROW(len);
1399     (void) strcpy(pri->sys_string + pri->sys_leng, s);
1400     pri->sys_leng += len;
1401 }
1402
1403 void
1404 grow(private_t *pri, int nbyte) /* reallocate format buffer if necessary */
1405 {
1406     while (pri->sys_leng + nbyte >= pri->sys_ssize)
1407         pri->sys_string = my_realloc(pri->sys_string,
1408             pri->sys_ssize * 2, "format buffer");
1409 }
1410
1411 void
1412 prt_clc(private_t *pri, int raw, long val)
1413 {
1414     const char *s = NULL;
1415
1416     if (!raw) {
1417         switch (val) {
1418             case CL_INITIALIZE:    s = "CL_INITIALIZE";    break;
1419             case CL_CONFIG:       s = "CL_CONFIG";       break;
1420         }
1421     }
1422
1423     if (s == NULL)
1424         prt_dec(pri, 0, val);
1425     else
1426         outstring(pri, s);
1427 }
1428
1429 void
1430 prt_clf(private_t *pri, int raw, long val)
1431 {
1432     const char *s = NULL;
1433
1434     if (!raw) {
1435         switch (pri->sys_args[0]) {
1436             case CL_CONFIG:
1437                 switch (pri->sys_args[1]) {
1438                     case CL_NODEID:
1439                         s = "CL_NODEID";
1440                     case CL_HIGHEST_NODEID:
1441                         s = "CL_HIGHEST_NODEID";
1442                     }
1443                 break;
1444             case CL_INITIALIZE:

```

```

1445         switch (pri->sys_args[1]) {
1446             case CL_GET_BOOTFLAG:
1447                 s = "CL_GET_BOOTFLAG";          break;
1448             }
1449         break;
1450     }
1451 }
1453 if (s == NULL)
1454     prt_dec(pri, 0, val);
1455 else
1456     outstring(pri, s);
1457 }
1459 void
1460 prt_sqc(private_t *pri, int raw, long val)      /* print sigqueue() si_code */
1461 {
1462     const char *s = NULL;
1464     if (!raw) {
1465         switch ((int)val) {
1466             case SI_QUEUE:          s = "SI_QUEUE";          break;
1467             case SI_TIMER:         s = "SI_TIMER";          break;
1468             case SI_ASYNCIO:      s = "SI_ASYNCIO";        break;
1469             case SI_MESGQ:       s = "SI_MESGQ";          break;
1470         }
1471     }
1473     if (s == NULL)
1474         prt_dec(pri, 0, val);
1475     else
1476         outstring(pri, s);
1477 }
1479 /*
1480 * print pricntlsys() (key, value) pair key.
1481 */
1482 void
1483 print_pck(private_t *pri, int raw, long val)
1484 {
1485     const char *s = NULL;
1486     char        cname[PC_CLNMSZ];
1488     if ((pri->sys_args[2] != PC_GETXPARGS &&
1489         pri->sys_args[2] != PC_SETXPARGS) || val == 0 || raw) {
1490         prt_dec(pri, 0, val);
1491         return;
1492     }
1494     if (pri->sys_args[3] == NULL) {
1495         if (val == PC_KY_CLNAME) {
1496             s = "PC_KY_CLNAME";
1497             outstring(pri, s);
1498         } else
1499             prt_dec(pri, 0, val);
1500         return;
1501     }
1503     if (Pread(Proc, &cname, PC_CLNMSZ, pri->sys_args[3]) != PC_CLNMSZ) {
1504         prt_dec(pri, 0, val);
1505         return;
1506     }
1508     if (strcmp(cname, "TS") == 0) {
1509         switch (val) {
1510             case TS_KY_UPRILIM:    s = "TS_KY_UPRILIM";    break;

```

```

1511             case TS_KY_UPRI:      s = "TS_KY_UPRI";        break;
1512             default:
1513             }
1514         } else if (strcmp(cname, "IA") == 0) {
1515             switch (val) {
1516                 case IA_KY_UPRILIM: s = "IA_KY_UPRILIM";  break;
1517                 case IA_KY_UPRI:   s = "IA_KY_UPRI";     break;
1518                 case IA_KY_MODE:   s = "IA_KY_MODE";     break;
1519                 default:
1520             }
1521         } else if (strcmp(cname, "RT") == 0) {
1522             switch (val) {
1523                 case RT_KY_PRI:     s = "RT_KY_PRI";      break;
1524                 case RT_KY_TQSECS: s = "RT_KY_TQSECS";   break;
1525                 case RT_KY_TQNSECS: s = "RT_KY_TQNSECS"; break;
1526                 case RT_KY_TQSIG:  s = "RT_KY_TQSIG";    break;
1527                 default:
1528             }
1529         } else if (strcmp(cname, "FSS") == 0) {
1530             switch (val) {
1531                 case FSS_KY_UPRILIM: s = "FSS_KY_UPRILIM"; break;
1532                 case FSS_KY_UPRI:   s = "FSS_KY_UPRI";   break;
1533                 default:
1534             }
1535         } else if (strcmp(cname, "FX") == 0) {
1536             switch (val) {
1537                 case FX_KY_UPRILIM: s = "FX_KY_UPRILIM";  break;
1538                 case FX_KY_UPRI:   s = "FX_KY_UPRI";     break;
1539                 case FX_KY_TQSECS: s = "FX_KY_TQSECS";   break;
1540                 case FX_KY_TQNSECS: s = "FX_KY_TQNSECS"; break;
1541                 default:
1542             }
1543         }
1545         if (s == NULL)
1546             prt_dec(pri, 0, val);
1547         else
1548             outstring(pri, s);
1549     }
1551 /*
1552 * print pricntlsys() fourth argument.
1553 */
1554 /*ARGSUSED*/
1555 void
1556 prt_pc4(private_t *pri, int raw, long val)
1557 {
1558     /* look at pricntlsys function */
1559     if ((pri->sys_args[2] != PC_GETXPARGS &&
1560         pri->sys_args[2] != PC_SETXPARGS))
1561         prt_hex(pri, 0, val);
1562     else if (val)
1563         prt_stg(pri, 0, val);
1564     else
1565         prt_dec(pri, 0, val);
1566 }
1568 /*
1569 * print pricntlsys() (key, value) pairs (5th argument).
1570 */
1571 /*ARGSUSED*/
1572 void
1573 prt_pc5(private_t *pri, int raw, long val)
1574 {
1575     pc_vaparms_t prms;
1576     pc_vaparm_t  *vpp = &prms.pc_parms[0];

```

```

1577     uint_t         cnt;

1580     /* look at pricntlsys function */
1581     if ((pri->sys_args[2] != PC_GETXPARMS &&
1582         pri->sys_args[2] != PC_SETXPARMS) || val == 0) {
1583         prt_dec(pri, 0, 0);
1584         return;
1585     }

1587     if (Pread(Proc, &prms, sizeof (prms), val) != sizeof (prms)) {
1588         prt_hex(pri, 0, val);
1589         return;
1590     }

1592     if ((cnt = prms.pc_vaparmscnt) > PC_VAPARMCNT)
1593         return;

1595     for (; cnt--; vpp++) {
1596         print_pck(pri, 0, vpp->pc_key);
1597         outstring(pri, " ", "");
1598         prt_hex(pri, 0, (long)vpp->pc_parm);
1599         outstring(pri, " ", "");
1600     }

1602     prt_dec(pri, 0, PC_KY_NULL);
1603 }

1606 void
1607 prt_psflags(private_t *pri, secflagset_t val)
1608 {
1609     char str[1024];

1611     if (val == 0) {
1612         outstring(pri, "0x0");
1613         return;
1614     }

1616     *str = '\0';
1617     if (secflag_isset(val, PROC_SEC_ASLR)) {
1618         (void) strlcat(str, "|PROC_SEC_ASLR", sizeof (str));
1619         secflag_clear(&val, PROC_SEC_ASLR);
1620     }
1621     if (secflag_isset(val, PROC_SEC_FORBIDNULLMAP)) {
1622         (void) strlcat(str, "|PROC_SEC_FORBIDNULLMAP",
1623             sizeof (str));
1624         secflag_clear(&val, PROC_SEC_FORBIDNULLMAP);
1625     }
1626     if (secflag_isset(val, PROC_SEC_NOEXECSTACK)) {
1627         (void) strlcat(str, "|PROC_SEC_NOEXECSTACK",
1628             sizeof (str));
1629         secflag_clear(&val, PROC_SEC_NOEXECSTACK);
1630     }

1632     if (val != 0)
1633         (void) snprintf(str, sizeof (str), "%s|0x%x", str, val);

1635     outstring(pri, str + 1);
1636 }

1638 /*
1639  * Print a psecflags(2) delta
1640  */
1641 void
1642 prt_psdelta(private_t *pri, int raw, long value)

```

```

1643 {
1644     secflagdelta_t psd;

1646     if ((raw != 0) ||
1647         (Pread(Proc, &psd, sizeof (psd), value) != sizeof (psd))) {
1648         prt_hex(pri, 0, value);
1649         return;
1650     }
1651     outstring(pri, "{ ");
1652     prt_psflags(pri, psd.psd_add);
1653     outstring(pri, ", ");
1654     prt_psflags(pri, psd.psd_rem);
1655     outstring(pri, ", ");
1656     prt_psflags(pri, psd.psd_assign);
1657     outstring(pri, ", ");
1658     outstring(pri, psd.psd_ass_active ? "B_TRUE" : "B_FALSE");
1659     outstring(pri, " }");
1660 }

1662 /*
1663  * Print a psecflagswhich_t
1664  */
1665 void
1666 prt_psfw(private_t *pri, int raw, long value)
1667 {
1668     psecflagwhich_t which = (psecflagwhich_t)value;
1669     char *s;

1671     if (raw != 0) {
1672         prt_dec(pri, 0, value);
1673         return;
1674     }

1676     switch (which) {
1677     case PSF_EFFECTIVE:
1678         s = "PSF_EFFECTIVE";
1679         break;
1680     case PSF_INHERIT:
1681         s = "PSF_INHERIT";
1682         break;
1683     case PSF_LOWER:
1684         s = "PSF_LOWER";
1685         break;
1686     case PSF_UPPER:
1687         s = "PSF_UPPER";
1688         break;
1689     }

1691     if (s == NULL)
1692         prt_dec(pri, 0, value);
1693     else
1694         outstring(pri, s);
1695 }

1697 #endif /* ! codereview */
1698 /*
1699  * Print processor set id, including logical expansion of "special" ids.
1700  */
1701 void
1702 prt_pst(private_t *pri, int raw, long val)
1703 {
1704     const char *s = NULL;

1706     if (!raw) {
1707         switch ((psetid_t)val) {
1708         case PS_NONE:
1709             s = "PS_NONE";
1710             break;

```

```

1709         case PS_QUERY:      s = "PS_QUERY";      break;
1710         case PS_MYID:       s = "PS_MYID";       break;
1711     }
1712 }

1714     if (s == NULL)
1715         prt_dec(pri, 0, val);
1716     else
1717         outstring(pri, s);
1718 }

1720 /*
1721  * Print meminfo() argument.
1722  */
1723 /*ARGSUSED*/
1724 void
1725 prt_mif(private_t *pri, int raw, long val)
1726 {
1727     struct meminfo minfo;

1729 #ifdef _LP64
1730     if (data_model == PR_MODEL_ILP32) {
1731         struct meminfo32 minfo32;

1733         if (Pread(Proc, &minfo32, sizeof (struct meminfo32), val) !=
1734             sizeof (struct meminfo32)) {
1735             prt_dec(pri, 0, pri->sys_args[1]);    /* addr_count */
1736             outstring(pri, ", ");
1737             prt_hex(pri, 0, val);
1738             return;
1739         }
1740         /*
1741          * arrange the arguments in the order that user calls with
1742          */
1743         prt_hex(pri, 0, minfo32.mi_inaddr);
1744         outstring(pri, ", ");
1745         prt_dec(pri, 0, pri->sys_args[1]);    /* addr_count */
1746         outstring(pri, ", ");
1747         prt_hex(pri, 0, minfo32.mi_info_req);
1748         outstring(pri, ", ");
1749         prt_dec(pri, 0, minfo32.mi_info_count);
1750         outstring(pri, ", ");
1751         prt_hex(pri, 0, minfo32.mi_outdata);
1752         outstring(pri, ", ");
1753         prt_hex(pri, 0, minfo32.mi_validity);
1754         return;
1755     }
1756 #endif
1757     if (Pread(Proc, &minfo, sizeof (struct meminfo), val) !=
1758         sizeof (struct meminfo)) {
1759         prt_dec(pri, 0, pri->sys_args[1]);    /* addr_count */
1760         outstring(pri, ", ");
1761         prt_hex(pri, 0, val);
1762         return;
1763     }
1764     /*
1765      * arrange the arguments in the order that user calls with
1766      */
1767     prt_hex(pri, 0, (long)minfo.mi_inaddr);
1768     outstring(pri, ", ");
1769     prt_dec(pri, 0, pri->sys_args[1]);    /* addr_count */
1770     outstring(pri, ", ");
1771     prt_hex(pri, 0, (long)minfo.mi_info_req);
1772     outstring(pri, ", ");
1773     prt_dec(pri, 0, minfo.mi_info_count);
1774     outstring(pri, ", ");

```

```

1775     prt_hex(pri, 0, (long)minfo.mi_outdata);
1776     outstring(pri, ", ");
1777     prt_hex(pri, 0, (long)minfo.mi_validity);
1778 }

1781 /*
1782  * Print so_socket() 1st argument.
1783  */
1784 /*ARGSUSED*/
1785 void
1786 prt_pfm(private_t *pri, int raw, long val)
1787 {
1788     /* Protocol Families have same names as Address Families */
1789     if ((ulong_t)val < MAX_AF_CODES) {
1790         outstring(pri, "PF_");
1791         outstring(pri, afcodes[val]);
1792     } else {
1793         prt_dec(pri, 0, val);
1794     }
1795 }

1797 /*
1798  * Print sockconfig() subcode.
1799  */
1800 /*ARGSUSED*/
1801 void
1802 prt_skc(private_t *pri, int raw, long val)
1803 {
1804     const char *s = NULL;

1806     if (!raw) {
1807         switch (val) {
1808             case SOCKCONFIG_ADD_SOCKET:
1809                 s = "SOCKCONFIG_ADD_SOCKET"; break;
1810             case SOCKCONFIG_REMOVE_SOCKET:
1811                 s = "SOCKCONFIG_REMOVE_SOCKET"; break;
1812             case SOCKCONFIG_ADD_FILTER:
1813                 s = "SOCKCONFIG_ADD_FILTER"; break;
1814             case SOCKCONFIG_REMOVE_FILTER:
1815                 s = "SOCKCONFIG_REMOVE_FILTER"; break;
1816         }
1817     }
1818     if (s == NULL)
1819         prt_dec(pri, 0, val);
1820     else
1821         outstring(pri, s);
1822 }
1823 /*
1824  * Print so_socket() 2nd argument.
1825  */
1826 /*ARGSUSED*/
1827 void
1828 prt_skt(private_t *pri, int raw, long val)
1829 {
1830     const char *s;
1831     long type = val & SOCK_TYPE_MASK;

1833     if ((ulong_t)type <= MAX_SOCKETYPES &&
1834         (s = socktype_codes[type]) != NULL) {
1835         outstring(pri, s);
1836         if ((val & SOCK_CLOEXEC) != 0) {
1837             outstring(pri, "|SOCK_CLOEXEC");
1838         }
1839     } else {
1840         prt_dec(pri, 0, val);

```

```

1841     }
1842 }

1845 /*
1846  * Print so_socket() 3rd argument.
1847  */
1848 /*ARGSUSED*/
1849 void
1850 prt_skp(private_t *pri, int raw, long val)
1851 {
1852     const char *s;

1854     /* cheating -- look at the protocol-family */
1855     switch (pri->sys_args[0]) {
1856     case PF_INET6:
1857     case PF_INET:
1858     case PF_NCA:    if ((s = ipprotos((int)val)) != NULL) {
1859                     outstring(pri, s);
1860                     break;
1861                 }
1862                 /* FALLTHROUGH */
1863     default:       prt_dec(pri, 0, val);
1864                 break;
1865     }
1866 }

1869 /*
1870  * Print so_socket() 5th argument.
1871  */
1872 /*ARGSUSED*/
1873 void
1874 prt_skv(private_t *pri, int raw, long val)
1875 {
1876     switch (val) {
1877     case SOV_STREAM:    outstring(pri, "SOV_STREAM");    break;
1878     case SOV_DEFAULT:  outstring(pri, "SOV_DEFAULT");    break;
1879     case SOV_SOCKSTREAM: outstring(pri, "SOV_SOCKSTREAM"); break;
1880     case SOV_SOCKBSD:  outstring(pri, "SOV_SOCKBSD");    break;
1881     case SOV_XPG4_2:   outstring(pri, "SOV_XPG4_2");    break;
1882     default:          prt_dec(pri, 0, val);             break;
1883     }
1884 }

1886 /*
1887  * Print accept4() flags argument.
1888  */
1889 void
1890 prt_acf(private_t *pri, int raw, long val)
1891 {
1892     int first = 1;
1893     if (raw || !val ||
1894         (val & ~(SOCK_CLOEXEC|SOCK_NDELAY|SOCK_NONBLOCK))) {
1895         prt_dex(pri, 0, val);
1896         return;
1897     }

1899     if (val & SOCK_CLOEXEC) {
1900         outstring(pri, "|SOCK_CLOEXEC" + first);
1901         first = 0;
1902     }
1903     if (val & SOCK_NDELAY) {
1904         outstring(pri, "|SOCK_NDELAY" + first);
1905         first = 0;
1906     }

```

```

1907         if (val & SOCK_NONBLOCK) {
1908             outstring(pri, "|SOCK_NONBLOCK" + first);
1909         }
1910     }

1913 /*
1914  * Print setsockopt()/getsockopt() 2nd argument.
1915  */
1916 /*ARGSUSED*/
1917 void
1918 prt_sol(private_t *pri, int raw, long val)
1919 {
1920     if (val == SOL_SOCKET) {
1921         outstring(pri, "SOL_SOCKET");
1922     } else if (val == SOL_ROUTE) {
1923         outstring(pri, "SOL_ROUTE");
1924     } else {
1925         const struct protoent *p;
1926         struct protoent res;
1927         char buf[NSS_BUFLEN_PROTOCOLS];

1929         if ((p = getprotobyname_r(val, &res,
1930             (char *)buf, sizeof (buf))) != NULL)
1931             outstring(pri, p->p_name);
1932     } else
1933         prt_dec(pri, 0, val);
1934     }
1935 }

1938 const char *
1939 sol_optname(private_t *pri, long val)
1940 {
1941     #define CBSIZE    sizeof (pri->code_buf)
1942     if (val >= SO_SNDBUF) {
1943         switch (val) {
1944         case SO_SNDBUF:    return ("SO_SNDBUF");
1945         case SO_RCVBUF:    return ("SO_RCVBUF");
1946         case SO_SNDBUF:    return ("SO_SNDBUF");
1947         case SO_RCVLOWAT: return ("SO_RCVLOWAT");
1948         case SO_SNDTIMEO: return ("SO_SNDTIMEO");
1949         case SO_RCVTIMEO: return ("SO_RCVTIMEO");
1950         case SO_ERROR:    return ("SO_ERROR");
1951         case SO_TYPE:     return ("SO_TYPE");
1952         case SO_PROTOCOL: return ("SO_PROTOCOL");
1953         case SO_ANON_MLP: return ("SO_ANON_MLP");
1954         case SO_MAC_EXEMPT: return ("SO_MAC_EXEMPT");
1955         case SO_ALLZONES: return ("SO_ALLZONES");
1956         case SO_MAC_IMPLICIT: return ("SO_MAC_IMPLICIT");
1957         case SO_VRRP:     return ("SO_VRRP");
1958         case SO_EXCLBIND: return ("SO_EXCLBIND");
1959         case SO_DOMAIN:   return ("SO_DOMAIN");

1961         default:          (void) snprintf(pri->code_buf, CBSIZE,
1962             "0x%lx", val);
1963         return (pri->code_buf);
1964     }
1965     } else {
1966         char *s = pri->code_buf;
1967         size_t used = 1;
1968         long val2;

1970         *s = '\0';
1971         val2 = val & ~(SO_DEBUG|SO_ACCEPTCONN|SO_REUSEADDR|SO_KEEPAIVE|
1972             SO_DONTROUTE|SO_BROADCAST|SO_USELOOPBACK|SO_LINGER|

```

```

1973     SO_OOBINLINE|SO_DGRAM_ERRIND|SO_RECVUCRED);
1974     if (val2)
1975         used = snprintf(s, CBSIZE, "|0x%lx", val2);
1976     if (val & SO_DEBUG)
1977         used = strlcat(s, "|SO_DEBUG", CBSIZE);
1978     if (val & SO_ACCEPTCONN)
1979         used = strlcat(s, "|SO_ACCEPTCONN", CBSIZE);
1980     if (val & SO_REUSEADDR)
1981         used = strlcat(s, "|SO_REUSEADDR", CBSIZE);
1982     if (val & SO_KEEPAVAIL)
1983         used = strlcat(s, "|SO_KEEPAVAIL", CBSIZE);
1984     if (val & SO_DONTROUTE)
1985         used = strlcat(s, "|SO_DONTROUTE", CBSIZE);
1986     if (val & SO_BROADCAST)
1987         used = strlcat(s, "|SO_BROADCAST", CBSIZE);
1988     if (val & SO_USELOOPBACK)
1989         used = strlcat(s, "|SO_USELOOPBACK", CBSIZE);
1990     if (val & SO_LINGER)
1991         used = strlcat(s, "|SO_LINGER", CBSIZE);
1992     if (val & SO_OOBINLINE)
1993         used = strlcat(s, "|SO_OOBINLINE", CBSIZE);
1994     if (val & SO_DGRAM_ERRIND)
1995         used = strlcat(s, "|SO_DGRAM_ERRIND", CBSIZE);
1996     if (val & SO_RECVUCRED)
1997         used = strlcat(s, "|SO_RECVUCRED", CBSIZE);
1998     if (used >= CBSIZE || val == 0)
1999         (void) snprintf(s + 1, CBSIZE - 1, "0x%lx", val);
2000     return ((const char *)(s + 1));
2001 }
2002 #undef CBSIZE
2003 }

2005 const char *
2006 route_optname(private_t *pri, long val)
2007 {
2008     switch (val) {
2009     case RT_AWARE:
2010         return ("RT_AWARE");
2011     default:
2012         (void) snprintf(pri->code_buf, sizeof (pri->code_buf),
2013             "0x%lx", val);
2014         return (pri->code_buf);
2015     }
2016 }

2018 const char *
2019 tcp_optname(private_t *pri, long val)
2020 {
2021     switch (val) {
2022     case TCP_NODELAY:         return ("TCP_NODELAY");
2023     case TCP_MAXSEG:         return ("TCP_MAXSEG");
2024     case TCP_KEEPAVAIL:      return ("TCP_KEEPAVAIL");
2025     case TCP_NOTIFY_THRESHOLD: return ("TCP_NOTIFY_THRESHOLD");
2026     case TCP_ABORT_THRESHOLD: return ("TCP_ABORT_THRESHOLD");
2027     case TCP_CONN_NOTIFY_THRESHOLD: return ("TCP_CONN_NOTIFY_THRESHOLD");
2028     case TCP_CONN_ABORT_THRESHOLD: return ("TCP_CONN_ABORT_THRESHOLD");
2029     case TCP_RECVDSTADDR:    return ("TCP_RECVDSTADDR");
2030     case TCP_ANONPRIVBIND:   return ("TCP_ANONPRIVBIND");
2031     case TCP_EXCLBIND:       return ("TCP_EXCLBIND");
2032     case TCP_INIT_CWND:      return ("TCP_INIT_CWND");
2033     case TCP_KEEPAVAIL_THRESHOLD: return ("TCP_KEEPAVAIL_THRESHOLD");
2034     case TCP_KEEPAVAIL_ABORT_THRESHOLD: return ("TCP_KEEPAVAIL_ABORT_THRESHOLD");
2035     case TCP_CORK:           return ("TCP_CORK");
2036     case TCP_RTO_INITIAL:    return ("TCP_RTO_INITIAL");
2037     case TCP_RTO_MIN:        return ("TCP_RTO_MIN");
2038

```

```

2039     case TCP_RTO_MAX:         return ("TCP_RTO_MAX");
2040     case TCP_LINGER2:        return ("TCP_LINGER2");
2041     case TCP_KEEPIRL:        return ("TCP_KEEPIRL");
2042     case TCP_KEEPCNT:        return ("TCP_KEEPCNT");
2043     case TCP_KEEPIRLV:       return ("TCP_KEEPIRLV");
2044
2045     default:
2046         (void) snprintf(pri->code_buf,
2047             sizeof (pri->code_buf),
2048             "0x%lx", val);
2049         return (pri->code_buf);
2050 }

2053 const char *
2054 sctp_optname(private_t *pri, long val)
2055 {
2056     switch (val) {
2057     case Sctp_RTOINFO:        return ("Sctp_RTOINFO");
2058     case Sctp_AssocINFO:     return ("Sctp_AssocINFO");
2059     case Sctp_InitMSG:       return ("Sctp_InitMSG");
2060     case Sctp_NodeLAY:       return ("Sctp_NodeLAY");
2061     case Sctp_AutoClose:     return ("Sctp_AutoClose");
2062     case Sctp_Set_Peer_Primary_Addr:
2063         return ("Sctp_Set_Peer_Primary_Addr");
2064     case Sctp_Primary_Addr:   return ("Sctp_Primary_Addr");
2065     case Sctp_Adaptation_Layer: return ("Sctp_Adaptation_Layer");
2066     case Sctp_Disable_Fragments: return ("Sctp_Disable_Fragments");
2067     case Sctp_Peer_Addr_Params: return ("Sctp_Peer_Addr_Params");
2068     case Sctp_Default_Send_Param: return ("Sctp_Default_Send_Param");
2069     case Sctp_Events:        return ("Sctp_Events");
2070     case Sctp_I_Want_Mapped_V4_Addr:
2071         return ("Sctp_I_Want_Mapped_V4_Addr");
2072     case Sctp_MaxSeg:         return ("Sctp_MaxSeg");
2073     case Sctp_Status:         return ("Sctp_Status");
2074     case Sctp_Get_Peer_Addr_Info: return ("Sctp_Get_Peer_Addr_Info");
2075
2076     case Sctp_Add_Addr:       return ("Sctp_Add_Addr");
2077     case Sctp_Rem_Addr:       return ("Sctp_Rem_Addr");
2078
2079     default:
2080         (void) snprintf(pri->code_buf,
2081             sizeof (pri->code_buf),
2082             "0x%lx", val);
2083         return (pri->code_buf);
2084 }

2087 const char *
2088 udp_optname(private_t *pri, long val)
2089 {
2090     switch (val) {
2091     case UDP_CHECKSUM:        return ("UDP_CHECKSUM");
2092     case UDP_ANONPRIVBIND:    return ("UDP_ANONPRIVBIND");
2093     case UDP_EXCLBIND:        return ("UDP_EXCLBIND");
2094     case UDP_RCVHDR:          return ("UDP_RCVHDR");
2095     case UDP_NAT_T_ENDPOINT: return ("UDP_NAT_T_ENDPOINT");
2096
2097     default:
2098         (void) snprintf(pri->code_buf,
2099             sizeof (pri->code_buf), "0x%lx",
2100             val);
2101         return (pri->code_buf);
2102 }

```

```

2105 /*
2106  * Print setsockopt()/getsockopt() 3rd argument.
2107  */
2108 /*ARGSUSED*/
2109 void
2110 prt_son(private_t *pri, int raw, long val)
2111 {
2112     /* cheating -- look at the level */
2113     switch (pri->sys_args[1]) {
2114     case SOL_SOCKET:      outstring(pri, sol_optname(pri, val));
2115                          break;
2116     case SOL_ROUTE:      outstring(pri, route_optname(pri, val));
2117                          break;
2118     case IPPROTO_TCP:    outstring(pri, tcp_optname(pri, val));
2119                          break;
2120     case IPPROTO_UDP:    outstring(pri, udp_optname(pri, val));
2121                          break;
2122     case IPPROTO_SCTP:   outstring(pri, sctp_optname(pri, val));
2123                          break;
2124     default:             prt_dec(pri, 0, val);
2125                          break;
2126     }
2127 }

2130 /*
2131  * Print utrap type
2132  */
2133 /*ARGSUSED*/
2134 void
2135 prt_utt(private_t *pri, int raw, long val)
2136 {
2137     const char *s = NULL;

2139 #ifdef __sparc
2140     if (!raw) {
2141         switch (val) {
2142         case UT_INSTRUCTION_DISABLED:
2143             s = "UT_INSTRUCTION_DISABLED"; break;
2144         case UT_INSTRUCTION_ERROR:
2145             s = "UT_INSTRUCTION_ERROR"; break;
2146         case UT_INSTRUCTION_PROTECTION:
2147             s = "UT_INSTRUCTION_PROTECTION"; break;
2148         case UT_ILLTRAP_INSTRUCTION:
2149             s = "UT_ILLTRAP_INSTRUCTION"; break;
2150         case UT_ILLEGAL_INSTRUCTION:
2151             s = "UT_ILLEGAL_INSTRUCTION"; break;
2152         case UT_PRIVILEGED_OPCODE:
2153             s = "UT_PRIVILEGED_OPCODE"; break;
2154         case UT_FP_DISABLED:
2155             s = "UT_FP_DISABLED"; break;
2156         case UT_FP_EXCEPTION_IEEE_754:
2157             s = "UT_FP_EXCEPTION_IEEE_754"; break;
2158         case UT_FP_EXCEPTION_OTHER:
2159             s = "UT_FP_EXCEPTION_OTHER"; break;
2160         case UT_TAG_OVERFLOW:
2161             s = "UT_TAG_OVERFLOW"; break;
2162         case UT_DIVISION_BY_ZERO:
2163             s = "UT_DIVISION_BY_ZERO"; break;
2164         case UT_DATA_EXCEPTION:
2165             s = "UT_DATA_EXCEPTION"; break;
2166         case UT_DATA_ERROR:
2167             s = "UT_DATA_ERROR"; break;
2168         case UT_DATA_PROTECTION:
2169             s = "UT_DATA_PROTECTION"; break;
2170         case UT_MEM_ADDRESS_NOT_ALIGNED:

```

```

2171             s = "UT_MEM_ADDRESS_NOT_ALIGNED"; break;
2172         case UT_PRIVILEGED_ACTION:
2173             s = "UT_PRIVILEGED_ACTION"; break;
2174         case UT_ASYNC_DATA_ERROR:
2175             s = "UT_ASYNC_DATA_ERROR"; break;
2176         case UT_TRAP_INSTRUCTION_16:
2177             s = "UT_TRAP_INSTRUCTION_16"; break;
2178         case UT_TRAP_INSTRUCTION_17:
2179             s = "UT_TRAP_INSTRUCTION_17"; break;
2180         case UT_TRAP_INSTRUCTION_18:
2181             s = "UT_TRAP_INSTRUCTION_18"; break;
2182         case UT_TRAP_INSTRUCTION_19:
2183             s = "UT_TRAP_INSTRUCTION_19"; break;
2184         case UT_TRAP_INSTRUCTION_20:
2185             s = "UT_TRAP_INSTRUCTION_20"; break;
2186         case UT_TRAP_INSTRUCTION_21:
2187             s = "UT_TRAP_INSTRUCTION_21"; break;
2188         case UT_TRAP_INSTRUCTION_22:
2189             s = "UT_TRAP_INSTRUCTION_22"; break;
2190         case UT_TRAP_INSTRUCTION_23:
2191             s = "UT_TRAP_INSTRUCTION_23"; break;
2192         case UT_TRAP_INSTRUCTION_24:
2193             s = "UT_TRAP_INSTRUCTION_24"; break;
2194         case UT_TRAP_INSTRUCTION_25:
2195             s = "UT_TRAP_INSTRUCTION_25"; break;
2196         case UT_TRAP_INSTRUCTION_26:
2197             s = "UT_TRAP_INSTRUCTION_26"; break;
2198         case UT_TRAP_INSTRUCTION_27:
2199             s = "UT_TRAP_INSTRUCTION_27"; break;
2200         case UT_TRAP_INSTRUCTION_28:
2201             s = "UT_TRAP_INSTRUCTION_28"; break;
2202         case UT_TRAP_INSTRUCTION_29:
2203             s = "UT_TRAP_INSTRUCTION_29"; break;
2204         case UT_TRAP_INSTRUCTION_30:
2205             s = "UT_TRAP_INSTRUCTION_30"; break;
2206         case UT_TRAP_INSTRUCTION_31:
2207             s = "UT_TRAP_INSTRUCTION_31"; break;
2208     }
2209 }
2210 #endif /* __sparc */

2212     if (s == NULL)
2213         prt_dec(pri, 0, val);
2214     else
2215         outstring(pri, s);
2216 }

2219 /*
2220  * Print utrap handler
2221  */
2222 void
2223 prt_uth(private_t *pri, int raw, long val)
2224 {
2225     const char *s = NULL;

2227     if (!raw) {
2228         switch (val) {
2229         case (long)UTH_NOCHANGE:      s = "UTH_NOCHANGE"; break;
2230         }
2231     }

2233     if (s == NULL)
2234         prt_hex(pri, 0, val);
2235     else
2236         outstring(pri, s);

```

```

2237 }
2239 const char *
2240 access_flags(private_t *pri, long arg)
2241 {
2242 #define E_OK 010
2243     char *str = pri->code_buf;
2245     if (arg & ~(R_OK|W_OK|X_OK|E_OK))
2246         return (NULL);
2248     /* NB: F_OK == 0 */
2249     if (arg == F_OK)
2250         return ("F_OK");
2251     if (arg == E_OK)
2252         return ("F_OK|E_OK");
2254     *str = '\0';
2255     if (arg & R_OK)
2256         (void) strlcat(str, "|R_OK", sizeof (pri->code_buf));
2257     if (arg & W_OK)
2258         (void) strlcat(str, "|W_OK", sizeof (pri->code_buf));
2259     if (arg & X_OK)
2260         (void) strlcat(str, "|X_OK", sizeof (pri->code_buf));
2261     if (arg & E_OK)
2262         (void) strlcat(str, "|E_OK", sizeof (pri->code_buf));
2263     return ((const char *) (str + 1));
2264 #undef E_OK
2265 }
2267 /*
2268  * Print access() flags.
2269  */
2270 void
2271 prt_acc(private_t *pri, int raw, long val)
2272 {
2273     const char *s = raw? NULL : access_flags(pri, val);
2275     if (s == NULL)
2276         prt_dex(pri, 0, val);
2277     else
2278         outstring(pri, s);
2279 }
2281 /*
2282  * Print shutdown() "how" (2nd) argument
2283  */
2284 void
2285 prt_sht(private_t *pri, int raw, long val)
2286 {
2287     if (raw) {
2288         prt_dex(pri, 0, val);
2289         return;
2290     }
2291     switch (val) {
2292     case SHUT_RD:    outstring(pri, "SHUT_RD");    break;
2293     case SHUT_WR:    outstring(pri, "SHUT_WR");    break;
2294     case SHUT_RDWR: outstring(pri, "SHUT_RDWR"); break;
2295     default:         prt_dec(pri, 0, val);         break;
2296     }
2297 }
2299 /*
2300  * Print fcntl() F_SETFL flags (3rd) argument or fdsync flag (2nd arg)
2301  */
2302 static struct fcntl_flags {

```

```

2303     long        val;
2304     const char  *name;
2305 } fcntl_flags[] = {
2306 #define FC_FL(flag)    { (long)flag, "|" # flag }
2307     FC_FL(FREVKED),
2308     FC_FL(FREAD),
2309     FC_FL(FWRITE),
2310     FC_FL(FNDELAY),
2311     FC_FL(FAPPEND),
2312     FC_FL(FSYNC),
2313     FC_FL(FDSYNC),
2314     FC_FL(FRSYNC),
2315     FC_FL(FOFFMAX),
2316     FC_FL(FNONBLOCK),
2317     FC_FL(FCREAT),
2318     FC_FL(FTRUNC),
2319     FC_FL(FEXCL),
2320     FC_FL(FNOCTTY),
2321     FC_FL(FXATTR),
2322     FC_FL(FASYNC),
2323     FC_FL(FNODSYNC)
2324 #undef FC_FL
2325 };
2327 void
2328 prt_ffg(private_t *pri, int raw, long val)
2329 {
2330 #define CBSIZE  sizeof (pri->code_buf)
2331     char *s = pri->code_buf;
2332     size_t used = 1;
2333     struct fcntl_flags *fp;
2335     if (raw) {
2336         (void) snprintf(s, CBSIZE, "0x%lx", val);
2337         outstring(pri, s);
2338         return;
2339     }
2340     if (val == 0) {
2341         outstring(pri, "(no flags)");
2342         return;
2343     }
2345     *s = '\0';
2346     for (fp = fcntl_flags;
2347          fp < &fcntl_flags[sizeof (fcntl_flags) / sizeof (*fp)]; fp++) {
2348         if (val & fp->val) {
2349             used = strlcat(s, fp->name, CBSIZE);
2350             val &= ~fp->val;
2351         }
2352     }
2354     if (val != 0 && used <= CBSIZE)
2355         used += snprintf(s + used, CBSIZE - used, "|0x%lx", val);
2357     if (used >= CBSIZE)
2358         (void) snprintf(s + 1, CBSIZE-1, "0x%lx", val);
2359     outstring(pri, s + 1);
2360 #undef CBSIZE
2361 }
2363 void
2364 prt_prs(private_t *pri, int raw, long val)
2365 {
2366     static size_t setsize;
2367     priv_set_t *set = priv_allocset();

```



```

2369     if (setsize == 0) {
2370         const priv_impl_info_t *info = getprivimplinfo();
2371         if (info != NULL)
2372             setsize = info->priv_setsize * sizeof (priv_chunk_t);
2373     }

2375     if (setsize != 0 && !raw && set != NULL &&
2376         Pread(Proc, set, setsize, val) == setsize) {
2377         int i;

2379         outstring(pri, "{");
2380         for (i = 0; i < setsize / sizeof (priv_chunk_t); i++) {
2381             char buf[9]; /* 8 hex digits + '\0' */
2382             (void) snprintf(buf, sizeof (buf), "%08x",
2383                 ((priv_chunk_t *)set)[i]);
2384             outstring(pri, buf);
2385         }

2387         outstring(pri, "}");
2388     } else {
2389         prt_hex(pri, 0, val);
2390     }

2392     if (set != NULL)
2393         priv_freerset(set);
2394 }

2396 /*
2397  * Print privilege set operation.
2398  */
2399 void
2400 prt_pro(private_t *pri, int raw, long val)
2401 {
2402     const char *s = NULL;

2404     if (!raw) {
2405         switch ((priv_op_t)val) {
2406             case PRIV_ON:      s = "PRIV_ON";      break;
2407             case PRIV_OFF:     s = "PRIV_OFF";     break;
2408             case PRIV_SET:     s = "PRIV_SET";     break;
2409         }
2410     }

2412     if (s == NULL)
2413         prt_dec(pri, 0, val);
2414     else
2415         outstring(pri, s);
2416 }

2418 /*
2419  * Print privilege set name
2420  */
2421 void
2422 prt_prn(private_t *pri, int raw, long val)
2423 {
2424     const char *s = NULL;

2426     if (!raw)
2427         s = priv_getsetbynum((int)val);

2429     if (s == NULL)
2430         prt_dec(pri, 0, val);
2431     else {
2432         char *dup = strdup(s);
2433         char *q;

```

```

2435         /* Do the best we can in this case */
2436         if (dup == NULL) {
2437             outstring(pri, s);
2438             return;
2439         }

2441         outstring(pri, "PRIV_");

2443         q = dup;

2445         while (*q != '\0') {
2446             *q = toupper(*q);
2447             q++;
2448         }
2449         outstring(pri, dup);
2450         free(dup);
2451     }
2452 }

2454 /*
2455  * Print process flag names.
2456  */
2457 void
2458 prt_pfl(private_t *pri, int raw, long val)
2459 {
2460     const char *s = NULL;

2462     if (!raw) {
2463         switch ((int)val) {
2464             case PRIV_DEBUG:   s = "PRIV_DEBUG";   break;
2465             case PRIV_AWARE:   s = "PRIV_AWARE";   break;
2466             case PRIV_XPOLICY: s = "PRIV_XPOLICY";  break;
2467             case PRIV_AWARE_RESET: s = "PRIV_AWARE_RESET"; break;
2468             case PRIV_PFEEXEC: s = "PRIV_PFEEXEC";  break;
2469             case NET_MAC_AWARE: s = "NET_MAC_AWARE"; break;
2470             case NET_MAC_AWARE_INHERIT:
2471                 s = "NET_MAC_AWARE_INHERIT";
2472             break;
2473         }
2474     }

2476     if (s == NULL)
2477         prt_dec(pri, 0, val);
2478     else
2479         outstring(pri, s);
2480 }

2482 /*
2483  * Print lgrp_affinity_{get,set}() arguments.
2484  */
2485 /*ARGSUSED*/
2486 void
2487 prt_laf(private_t *pri, int raw, long val)
2488 {
2489     lgrp_affinity_args_t laff;

2491     if (Pread(Proc, &laff, sizeof (lgrp_affinity_args_t), val) !=
2492         sizeof (lgrp_affinity_args_t)) {
2493         prt_hex(pri, 0, val);
2494         return;
2495     }
2496     /*
2497      * arrange the arguments in the order that user calls with
2498      */
2499     prt_dec(pri, 0, laff.idtype);
2500     outstring(pri, ", ");

```

```

2501     prt_dec(pri, 0, laff.id);
2502     outstring(pri, ", ");
2503     prt_dec(pri, 0, laff.lgrp);
2504     outstring(pri, ", ");
2505     if (pri->sys_args[0] == LGRP_SYS_AFFINITY_SET)
2506         prt_dec(pri, 0, laff.aff);
2507 }

2509 /*
2510  * Print a key_t as IPC_PRIVATE if it is 0.
2511  */
2512 void
2513 prt_key(private_t *pri, int raw, long val)
2514 {
2515     if (!raw && val == 0)
2516         outstring(pri, "IPC_PRIVATE");
2517     else
2518         prt_dec(pri, 0, val);
2519 }

2522 /*
2523  * Print zone_getattr() attribute types.
2524  */
2525 void
2526 prt_zga(private_t *pri, int raw, long val)
2527 {
2528     const char *s = NULL;

2530     if (!raw) {
2531         switch ((int)val) {
2532             case ZONE_ATTR_NAME:      s = "ZONE_ATTR_NAME";   break;
2533             case ZONE_ATTR_ROOT:     s = "ZONE_ATTR_ROOT";    break;
2534             case ZONE_ATTR_STATUS:   s = "ZONE_ATTR_STATUS";  break;
2535             case ZONE_ATTR_PRIVSET:  s = "ZONE_ATTR_PRIVSET"; break;
2536             case ZONE_ATTR_UNIQID:   s = "ZONE_ATTR_UNIQID";  break;
2537             case ZONE_ATTR_POOLID:   s = "ZONE_ATTR_POOLID";  break;
2538             case ZONE_ATTR_INITPID:  s = "ZONE_ATTR_INITPID"; break;
2539             case ZONE_ATTR_SLBL:     s = "ZONE_ATTR_SLBL";    break;
2540             case ZONE_ATTR_INITNAME: s = "ZONE_ATTR_INITNAME"; break;
2541             case ZONE_ATTR_BOOTARGS: s = "ZONE_ATTR_BOOTARGS"; break;
2542             case ZONE_ATTR_BRAND:    s = "ZONE_ATTR_BRAND";   break;
2543             case ZONE_ATTR_FLAGS:    s = "ZONE_ATTR_FLAGS";   break;
2544             case ZONE_ATTR_PHYS_MCAP: s = "ZONE_ATTR_PHYS_MCAP"; break;
2545         }
2546     }

2548     if (s == NULL)
2549         prt_dec(pri, 0, val);
2550     else
2551         outstring(pri, s);
2552 }

2554 /*
2555  * Print a file descriptor as AT_FDCWD if necessary
2556  */
2557 void
2558 prt_atc(private_t *pri, int raw, long val)
2559 {
2560     if ((int)val == AT_FDCWD) {
2561         if (raw)
2562             prt_hex(pri, 0, (uint_t)AT_FDCWD);
2563         else
2564             outstring(pri, "AT_FDCWD");
2565     } else {
2566         prt_dec(pri, 0, val);

```

```

2567     }
2568 }

2570 /*
2571  * Print Trusted Networking database operation codes (labelsys; tn*)
2572  */
2573 static void
2574 prt_tnd(private_t *pri, int raw, long val)
2575 {
2576     const char *s = NULL;

2578     if (!raw) {
2579         switch ((tsol_dbops_t)val) {
2580             case TNDB_NOOP:          s = "TNDB_NOOP";          break;
2581             case TNDB_LOAD:         s = "TNDB_LOAD";          break;
2582             case TNDB_DELETE:       s = "TNDB_DELETE";        break;
2583             case TNDB_FLUSH:        s = "TNDB_FLUSH";         break;
2584             case TNDB_GET:          s = "TNDB_GET";           break;
2585         }
2586     }

2588     if (s == NULL)
2589         prt_dec(pri, 0, val);
2590     else
2591         outstring(pri, s);
2592 }

2594 /*
2595  * Print LIO_XX flags
2596  */
2597 void
2598 prt_lio(private_t *pri, int raw, long val)
2599 {
2600     if (raw)
2601         prt_dec(pri, 0, val);
2602     else if (val == LIO_WAIT)
2603         outstring(pri, "LIO_WAIT");
2604     else if (val == LIO_NOWAIT)
2605         outstring(pri, "LIO_NOWAIT");
2606     else
2607         prt_dec(pri, 0, val);
2608 }

2610 const char *
2611 door_flags(private_t *pri, long val)
2612 {
2613     door_attr_t attr = (door_attr_t)val;
2614     char *str = pri->code_buf;

2616     *str = '\0';
2617     #define PROCESS_FLAG(flg) \
2618     if (attr & flg) { \
2619         (void) strlcat(str, "|" #flg, sizeof (pri->code_buf)); \
2620         attr &= ~flg; \
2621     }

2623     PROCESS_FLAG(DOOR_UNREF);
2624     PROCESS_FLAG(DOOR_UNREF_MULTI);
2625     PROCESS_FLAG(DOOR_PRIVATE);
2626     PROCESS_FLAG(DOOR_REFUSE_DESC);
2627     PROCESS_FLAG(DOOR_NO_CANCEL);
2628     PROCESS_FLAG(DOOR_LOCAL);
2629     PROCESS_FLAG(DOOR_REVOKED);
2630     PROCESS_FLAG(DOOR_IS_UNREF);
2631     #undef PROCESS_FLAG

```

```

2633     if (attr != 0 || *str == '\0') {
2634         size_t len = strlen(str);
2635         (void) snprintf(str + len, sizeof (pri->code_buf) - len,
2636             "|0x%X", attr);
2637     }
2639     return (str + 1);
2640 }
2642 /*
2643  * Print door_create() flags
2644  */
2645 void
2646 prt_dfl(private_t *pri, int raw, long val)
2647 {
2648     if (raw)
2649         prt_hex(pri, 0, val);
2650     else
2651         outstring(pri, door_flags(pri, val));
2652 }
2654 /*
2655  * Print door_*param() param argument
2656  */
2657 void
2658 prt_dpm(private_t *pri, int raw, long val)
2659 {
2660     if (raw)
2661         prt_hex(pri, 0, val);
2662     else if (val == DOOR_PARAM_DESC_MAX)
2663         outstring(pri, "DOOR_PARAM_DESC_MAX");
2664     else if (val == DOOR_PARAM_DATA_MIN)
2665         outstring(pri, "DOOR_PARAM_DATA_MIN");
2666     else if (val == DOOR_PARAM_DATA_MAX)
2667         outstring(pri, "DOOR_PARAM_DATA_MAX");
2668     else
2669         prt_hex(pri, 0, val);
2670 }
2672 /*
2673  * Print rctlsys subcodes
2674  */
2675 void
2676 prt_rsc(private_t *pri, int raw, long val) /* print utssys code */
2677 {
2678     const char *s = raw? NULL : rctlsyscode(val);
2680     if (s == NULL)
2681         prt_dec(pri, 0, val);
2682     else
2683         outstring(pri, s);
2684 }
2686 /*
2687  * Print getrctl flags
2688  */
2689 void
2690 prt_rgf(private_t *pri, int raw, long val)
2691 {
2692     long action = val & (~RCTLSYS_ACTION_MASK);
2694     if (raw)
2695         prt_hex(pri, 0, val);
2696     else if (action == RCTL_FIRST)
2697         outstring(pri, "RCTL_FIRST");
2698     else if (action == RCTL_NEXT)

```

```

2699         outstring(pri, "RCTL_NEXT");
2700     else if (action == RCTL_USAGE)
2701         outstring(pri, "RCTL_USAGE");
2702     else
2703         prt_hex(pri, 0, val);
2704 }
2706 /*
2707  * Print setrctl flags
2708  */
2709 void
2710 prt_rsf(private_t *pri, int raw, long val)
2711 {
2712     long action = val & (~RCTLSYS_ACTION_MASK);
2713     long pval = val & RCTL_LOCAL_ACTION_MASK;
2714     char *s = pri->code_buf;
2716     if (raw) {
2717         prt_hex(pri, 0, val);
2718         return;
2719     } else if (action == RCTL_INSERT)
2720         (void) strcpy(s, "RCTL_INSERT");
2721     else if (action == RCTL_DELETE)
2722         (void) strcpy(s, "RCTL_DELETE");
2723     else if (action == RCTL_REPLACE)
2724         (void) strcpy(s, "RCTL_REPLACE");
2725     else {
2726         prt_hex(pri, 0, val);
2727         return;
2728     }
2730     if (pval & RCTL_USE_RECIPIENT_PID) {
2731         pval ^= RCTL_USE_RECIPIENT_PID;
2732         (void) strcat(s, "|RCTL_USE_RECIPIENT_PID",
2733             sizeof (pri->code_buf));
2734     }
2736     if ((pval & RCTLSYS_ACTION_MASK) != 0)
2737         prt_hex(pri, 0, val);
2738     else if (*s != '\0')
2739         outstring(pri, s);
2740     else
2741         prt_hex(pri, 0, val);
2742 }
2744 /*
2745  * Print rctlctl flags
2746  */
2747 void
2748 prt_rcf(private_t *pri, int raw, long val)
2749 {
2750     long action = val & (~RCTLSYS_ACTION_MASK);
2752     if (raw)
2753         prt_hex(pri, 0, val);
2754     else if (action == RCTLCTL_GET)
2755         outstring(pri, "RCTLCTL_GET");
2756     else if (action == RCTLCTL_SET)
2757         outstring(pri, "RCTLCTL_SET");
2758     else
2759         prt_hex(pri, 0, val);
2760 }
2762 /*
2763  * Print setprojctl flags
2764  */

```

```

2765 void
2766 prt_spf(private_t *pri, int raw, long val)
2767 {
2768     long action = val & TASK_PROJ_MASK;
2770     if (!raw && (action == TASK_PROJ_PURGE))
2771         outstring(pri, "TASK_PROJ_PURGE");
2772     else
2773         prt_hex(pri, 0, val);
2774 }
2776 /*
2777  * Print forkx() flags
2778  */
2779 void
2780 prt_fxf(private_t *pri, int raw, long val)
2781 {
2782     char *str;
2784     if (val == 0)
2785         outstring(pri, "0");
2786     else if (raw || (val & ~(FORK_NOSIGCHLD | FORK_WAITPID)))
2787         prt_hhx(pri, 0, val);
2788     else {
2789         str = pri->code_buf;
2790         *str = '\0';
2791         if (val & FORK_NOSIGCHLD)
2792             (void) strlcat(str, "|FORK_NOSIGCHLD",
2793                 sizeof (pri->code_buf));
2794         if (val & FORK_WAITPID)
2795             (void) strlcat(str, "|FORK_WAITPID",
2796                 sizeof (pri->code_buf));
2797         outstring(pri, str + 1);
2798     }
2799 }
2801 /*
2802  * Print faccessat() flag
2803  */
2804 void
2805 prt_fat(private_t *pri, int raw, long val)
2806 {
2807     if (val == 0)
2808         outstring(pri, "0");
2809     else if (!raw && val == AT_EACCESS)
2810         outstring(pri, "AT_EACCESS");
2811     else
2812         prt_hex(pri, 0, val);
2813 }
2815 /*
2816  * Print unlinkat() flag
2817  */
2818 void
2819 prt_uat(private_t *pri, int raw, long val)
2820 {
2821     if (val == 0)
2822         outstring(pri, "0");
2823     else if (!raw && val == AT_REMOVEDIR)
2824         outstring(pri, "AT_REMOVEDIR");
2825     else
2826         prt_hex(pri, 0, val);
2827 }
2829 /*
2830  * Print AT_SYMLINK_NOFOLLOW / AT_SYMLINK_FOLLOW flag

```

```

2831  */
2832 void
2833 prt_snf(private_t *pri, int raw, long val)
2834 {
2835     if (val == 0)
2836         outstring(pri, "0");
2837     else if (!raw && val == AT_SYMLINK_NOFOLLOW)
2838         outstring(pri, "AT_SYMLINK_NOFOLLOW");
2839     else if (!raw && val == AT_SYMLINK_FOLLOW)
2840         outstring(pri, "AT_SYMLINK_FOLLOW");
2841     else
2842         prt_hex(pri, 0, val);
2843 }
2845 void
2846 prt_grf(private_t *pri, int raw, long val)
2847 {
2848     int first = 1;
2850     if (raw != 0 || val == 0 ||
2851         (val & ~(GRND_NONBLOCK | GRND_RANDOM)) != 0) {
2852         outstring(pri, "0");
2853         return;
2854     }
2856     if (val & GRND_NONBLOCK) {
2857         outstring(pri, "|GRND_NONBLOCK" + first);
2858         first = 0;
2859     }
2860     if (val & GRND_RANDOM) {
2861         outstring(pri, "|GRND_RANDOM" + first);
2862         first = 0;
2863     }
2864 }
2866 /*
2867  * Array of pointers to print functions, one for each format.
2868  */
2869 void (* const Print[])() = {
2870     prt_nov, /* NOV -- no value */
2871     prt_dec, /* DEC -- print value in decimal */
2872     prt_oct, /* OCT -- print value in octal */
2873     prt_hex, /* HEX -- print value in hexadecimal */
2874     prt_dex, /* DEX -- print value in hexadecimal if big enough */
2875     prt_stg, /* STG -- print value as string */
2876     prt_ioc, /* IOC -- print ioctl code */
2877     prt_fcn, /* FCN -- print fcntl code */
2878     prt_s86, /* S86 -- print sysi86 code */
2879     prt_uts, /* UTS -- print utssys code */
2880     prt_opn, /* OPN -- print open code */
2881     prt_sig, /* SIG -- print signal name plus flags */
2882     prt_uat, /* UAT -- print unlinkat() flag */
2883     prt_msc, /* MSC -- print msgsys command */
2884     prt_msf, /* MSF -- print msgsys flags */
2885     prt_smc, /* SMC -- print semsys command */
2886     prt_sef, /* SEF -- print semsys flags */
2887     prt_shc, /* SHC -- print shmsys command */
2888     prt_shf, /* SHF -- print shmsys flags */
2889     prt_fat, /* FAT -- print faccessat( flag */
2890     prt_sfs, /* SFS -- print sysfs code */
2891     prt_rst, /* RST -- print string returned by syscall */
2892     prt_smf, /* SMF -- print streams message flags */
2893     prt_ioa, /* IOA -- print ioctl argument */
2894     prt_pip, /* PIP -- print pipe flags */
2895     prt_mtf, /* MTF -- print mount flags */
2896     prt_mft, /* MFT -- print mount file system type */

```

```

2897 prt_iob, /* IOB -- print contents of I/O buffer */
2898 prt_hhx, /* HHX -- print value in hexadecimal (half size) */
2899 prt_wop, /* WOP -- print waitsys() options */
2900 prt_spm, /* SPM -- print sigprocmask argument */
2901 prt_rlk, /* RLK -- print readlink buffer */
2902 prt_mpr, /* MPR -- print mmap()/mprotect() flags */
2903 prt_mty, /* MTY -- print mmap() mapping type flags */
2904 prt_mcf, /* MCF -- print memcntl() function */
2905 prt_mc4, /* MC4 -- print memcntl() (fourth) argument */
2906 prt_mc5, /* MC5 -- print memcntl() (fifth) argument */
2907 prt_mad, /* MAD -- print madvise() argument */
2908 prt_ulm, /* ULM -- print ulimit() argument */
2909 prt_rlm, /* RLM -- print get/setrlimit() argument */
2910 prt_cnf, /* CNF -- print sysconf() argument */
2911 prt_inf, /* INF -- print sysinfo() argument */
2912 prt_ptc, /* PTC -- print pathconf/fpathconf() argument */
2913 prt_fui, /* FUI -- print fusers() input argument */
2914 prt_idt, /* IDT -- print idtype_t, waitid() argument */
2915 prt_lwf, /* LWF -- print lwp_create() flags */
2916 prt_itm, /* ITM -- print [get|set]itimer() arg */
2917 prt_llo, /* LLO -- print long long offset arg */
2918 prt_mod, /* MOD -- print modctl() subcode */
2919 prt_whn, /* WHN -- print lseek() whence argument */
2920 prt_acl, /* ACL -- print acl() code */
2921 prt_aio, /* AIO -- print kaio() code */
2922 prt_aud, /* AUD -- print auditsys() code */
2923 prt_uns, /* DEC -- print value in unsigned decimal */
2924 prt_clc, /* CLC -- print cladm command argument */
2925 prt_clf, /* CLF -- print cladm flag argument */
2926 prt_cor, /* COR -- print corectl() subcode */
2927 prt_cco, /* CCO -- print corectl() options */
2928 prt_ccc, /* CCC -- print corectl() content */
2929 prt_rcc, /* RCC -- print corectl() returned content */
2930 prt_cpc, /* CPC -- print cpc() subcode */
2931 prt_sqc, /* SQC -- print sigqueue() si_code argument */
2932 prt_pc4, /* PC4 -- print priocntlsys() (fourth) argument */
2933 prt_pc5, /* PC5 -- print priocntlsys() (key, value) pairs */
2934 prt_pst, /* PST -- print processor set id */
2935 prt_mif, /* MIF -- print meminfo() arguments */
2936 prt_pfm, /* PFM -- print so_socket() proto-family (1st) arg */
2937 prt_skt, /* SKT -- print so_socket() socket-type (2nd) arg */
2938 prt_skp, /* SKP -- print so_socket() protocol (3rd) arg */
2939 prt_skv, /* SKV -- print socket version arg */
2940 prt_sol, /* SOL -- print [sg]setsockopt() level (2nd) arg */
2941 prt_son, /* SON -- print [sg]setsockopt() opt-name (3rd) arg */
2942 prt_utt, /* UTT -- print utrap type */
2943 prt_uth, /* UTH -- print utrap handler */
2944 prt_acc, /* ACC -- print access() flags */
2945 prt_sht, /* SHT -- print shutdown() how (2nd) argument */
2946 prt_ffg, /* FFG -- print fcntl() flags (3rd) argument */
2947 prt_prs, /* PRS -- print privilege set */
2948 prt_pro, /* PRO -- print privilege set operation */
2949 prt_prn, /* PRN -- print privilege set name */
2950 prt_pfl, /* PFL -- print privilege/process flag name */
2951 prt_laf, /* LAF -- print lgrp_affinity arguments */
2952 prt_key, /* KEY -- print key_t 0 as IPC_PRIVATE */
2953 prt_zga, /* ZGA -- print zone_getattr attribute types */
2954 prt_atc, /* ATC -- print AT_FDCWD or file descriptor */
2955 prt_lio, /* LIO -- print LIO_XX flags */
2956 prt_dfl, /* DFL -- print door_create() flags */
2957 prt_dpm, /* DPM -- print DOOR_PARAM_XX flags */
2958 prt_tnd, /* TND -- print trusted network data base opcode */
2959 prt_rsc, /* RSC -- print rctlsys() subcodes */
2960 prt_rgf, /* RGF -- print getrctl() flags */
2961 prt_rsf, /* RSF -- print setrctl() flags */
2962 prt_rcf, /* RCF -- print rctlsys_ctl() flags */

```

```

2963 prt_fxf, /* FXF -- print forkx() flags */
2964 prt_spf, /* SPF -- print rctlsys_projset() flags */
2965 prt_unl, /* UNL -- as prt_uns except for -1 */
2966 prt_mob, /* MOB -- print mmapobj() flags */
2967 prt_snf, /* SNF -- print AT_SYMLINK_NOFOLLOW flag */
2968 prt_skc, /* SKC -- print sockconfig() subcode */
2969 prt_acf, /* ACF -- print accept4 flags */
2970 prt_pfd, /* PFD -- print pipe fds */
2971 prt_grf, /* GRF -- print getrandom flags */
2972 prt_psdelta, /* PSDLT -- print psecflags(2) delta */
2973 prt_psfw, /* PSFW -- print psecflags(2) set */
2974 #endif /* ! codereview */
2975 prt_dec, /* HID -- hidden argument, make this the last one */
2976 };

```

new/usr/src/cmd/truss/print.h

1

```
*****
6424 Thu Jun 30 22:01:46 2016
new/usr/src/cmd/truss/print.h
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */

22 /*
23  * Copyright (c) 1989, 2010, Oracle and/or its affiliates. All rights reserved.
24  * Copyright (c) 2015, Joyent, Inc.
25 */

27 /*      Copyright (c) 1984, 1986, 1987, 1988, 1989 AT&T */
28 /*      All Rights Reserved      */

30 /* Copyright (c) 2013, OmniTI Computer Consulting, Inc. All right reserved. */

32 #ifndef _TRUSS_PRINT_H
33 #define _TRUSS_PRINT_H

35 #ifdef __cplusplus
36 extern "C" {
37 #endif

39 /*
40  * Argument & return value print codes.
41  */
42 #define NOV      0      /* no value */
43 #define DEC      1      /* print value in decimal */
44 #define OCT      2      /* print value in octal */
45 #define HEX      3      /* print value in hexadecimal */
46 #define DEX      4      /* print value in hexadecimal if big enough */
47 #define STG      5      /* print value as string */
48 #define IOC      6      /* print ioctl code */
49 #define FCNTL    7      /* print fcntl code */
50 #define S86      8      /* print sysi86 code */
51 #define UTS      9      /* print utssys code */
52 #define OPN     10      /* print open code */
53 #define SIG     11      /* print signal name plus flags */
54 #define UAT     12      /* print unlinkat() flag */
55 #define MSC     13      /* print msgsys command */
56 #define MSF     14      /* print msgsys flags */
57 #define SMC     15      /* print semsys command */
58 #define SEF     16      /* print semsys flags */
```

new/usr/src/cmd/truss/print.h

2

```
59 #define SHC      17      /* print shmsys command */
60 #define SHF      18      /* print shmsys flags */
61 #define FAT      19      /* print faccessat() flag */
62 #define SFS      20      /* print sysfs code */
63 #define RST      21      /* print string returned by sys call */
64 #define SMF      22      /* print streams message flags */
65 #define IOA      23      /* print ioctl argument */
66 #define PIP      24      /* print pipe flags */
67 #define MTF      25      /* print mount flags */
68 #define MFT      26      /* print mount file system type */
69 #define IOB      27      /* print contents of I/O buffer */
70 #define HHX      28      /* print value in hexadecimal (half size) */
71 #define WOP      29      /* print waitsys() options */
72 #define SPM      30      /* print sigprocmask argument */
73 #define RLK      31      /* print readlink buffer */
74 #define MPR      32      /* print mmap()/mprotect() flags */
75 #define MTY      33      /* print mmap() mapping type flags */
76 #define MCF      34      /* print memcntl() function */
77 #define MC4      35      /* print memcntl() (fourth) argument */
78 #define MC5      36      /* print memcntl() (fifth) argument */
79 #define MAD      37      /* print madvise() argument */
80 #define ULM      38      /* print ulimit() argument */
81 #define RLM      39      /* print get/setrlimit() argument */
82 #define CNF      40      /* print sysconfig() argument */
83 #define INF      41      /* print sysinfo() argument */
84 #define PTC      42      /* print pathconf/fpathconf() argument */
85 #define FUI      43      /* print fusers() input argument */
86 #define IDT      44      /* print idtype_t, waitid() argument */
87 #define LWF      45      /* print lwp_create() flags */
88 #define ITM      46      /* print [get|set]littimer() arg */
89 #define LLO      47      /* print long long offset */
90 #define MOD      48      /* print modctl() code */
91 #define WHN      49      /* print lseek() whence argument */
92 #define ACL      50      /* print acl() code */
93 #define AIO      51      /* print kaio() code */
94 #define AUD      52      /* print auditsys() code */
95 #define UNS      53      /* print value in unsigned decimal */
96 #define CLC      54      /* print cladm() command argument */
97 #define CLF      55      /* print cladm() flag argument */
98 #define COR      56      /* print corectl() subcode */
99 #define CCO      57      /* print corectl() options */
100 #define CCC      58      /* print corectl() content */
101 #define RCC      59      /* print corectl() content */
102 #define CPC      60      /* print cpc() subcode */
103 #define SQC      61      /* print sigqueue() si_code argument */
104 #define PC4      62      /* print priocntlsys() (fourth) argument */
105 #define PC5      63      /* print priocntlsys() (key-value) pairs */
106 #define PST      64      /* print processor set id */
107 #define MIF      65      /* print meminfo() argument */
108 #define PFM      66      /* print so_socket() proto-family (1st) arg */
109 #define SKT      67      /* print so_socket() socket type (2nd) arg */
110 #define SKP      68      /* print so_socket() protocol (3rd) arg */
111 #define SKV      69      /* print so_socket() version (5th) arg */
112 #define SOL      70      /* print [sg]setsockopt() level (2nd) arg */
113 #define SON      71      /* print [sg]setsockopt() name (3rd) arg */
114 #define UTT      72      /* print utrap type */
115 #define UTH      73      /* print utrap handler */
116 #define ACC      74      /* print access flags */
117 #define SHT      75      /* print shutdown() "how" (2nd) arg */
118 #define FFG      76      /* print fcntl() flags (3rd) arg */
119 #define PRS      77      /* privilege set */
120 #define PRO      78      /* privilege set operation */
121 #define PRN      79      /* privilege set name */
122 #define PFL      80      /* privilege/process flag name */
123 #define LAF      81      /* print lgrp_affinity arguments */
124 #define KEY      82      /* print key_t 0 as IPC_PRIVATE */
```

```
125 #define ZGA      83      /* print zone_getattr attribute types */
126 #define ATC      84      /* print AT_FDCWD or file descriptor */
127 #define LIO      85      /* print LIO_XX flags */
128 #define DFL      86      /* print door_create() flags */
129 #define DPM      87      /* print DOOR_PARAM_XX flags */
130 #define TND      88      /* print trusted network data base opcode */
131 #define RSC      89      /* print rctlsys subcode */
132 #define RGF      90      /* print rctlsys_get flags */
133 #define RGF      91      /* print rctlsys_set flags */
134 #define RCF      92      /* print rctlsys_ctl flags */
135 #define FXF      93      /* print forkx flags */
136 #define SPF      94      /* print rctlsys_projset flags */
137 #define UN1      95      /* unsigned except for -1 */
138 #define MOB      96      /* print mmapobj() flags */
139 #define SNF      97      /* print AT_SYMLINK_[NO]FOLLOW flag */
140 #define SKC      98      /* print sockconfig subcode */
141 #define ACF      99      /* accept4 flags */
142 #define PFD     100      /* pipe fds[2] */
143 #define GRF     101      /* getrandom flags */
144 #define PSDLT   102      /* secflagsdelta_t */
145 #define PSFW    103      /* psecflagswhich_t */
146 #define HID     104      /* hidden argument, don't print */
144 #define HID     102      /* hidden argument, don't print */
147                          /* make sure HID is always the last member */

149 /*
150 * Print routines, indexed by print codes.
151 */
152 extern void (* const Print[])();

154 #ifdef __cplusplus
155 }
_____unchanged_portion_omitted_
```

```

*****
58033 Thu Jun 30 22:01:47 2016
new/usr/src/cmd/truss/systable.c
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
    unchanged_portion_omitted

221 const struct systable systable[] = {
222 { NULL, 8, HEX, HEX, HEX, HEX, HEX, HEX, HEX, HEX, HEX, HEX},
223 { "_exit", 1, DEC, NOV, DEC},
224 { "psecflags", 3, DEC, NOV, HEX, PSFW, PSDLT},
224 { NULL, 8, HEX, HEX, HEX, HEX, HEX, HEX, HEX, HEX, HEX, HEX},
225 { "read", 3, DEC, NOV, DEC, IOB, UNS},
226 { "write", 3, DEC, NOV, DEC, IOB, UNS},
227 { "open", 3, DEC, NOV, STG, OPN, OCT},
228 { "close", 1, DEC, NOV, DEC},
229 { "linkat", 5, DEC, NOV, ATC, STG, ATC, STG, SNF},
230 { NULL, 8, HEX, HEX, HEX, HEX, HEX, HEX, HEX, HEX, HEX, HEX},
231 { "link", 2, DEC, NOV, STG, STG},
232 { "unlink", 1, DEC, NOV, STG},
233 { "symlinkat", 3, DEC, NOV, STG, ATC, STG},
234 { "chdir", 1, DEC, NOV, STG},
235 { "time", 0, DEC, NOV},
236 { "mknod", 3, DEC, NOV, STG, OCT, HEX},
237 { "chmod", 2, DEC, NOV, STG, OCT},
238 { "chown", 3, DEC, NOV, STG, DEC, DEC},
239 { "brk", 1, DEC, NOV, HEX},
240 { "stat", 2, DEC, NOV, STG, HEX},
241 { "lseek", 3, DEC, NOV, DEC, DEX, WHN},
242 { "getpid", 0, DEC, DEC},
243 { "mount", 8, DEC, NOV, STG, STG, MTF, MFT, HEX, DEC, HEX, DEC},
244 { "readlinkat", 4, DEC, NOV, ATC, STG, RLK, UNS},
245 { "setuid", 1, DEC, NOV, UNS},
246 { "getuid", 0, UNS, UNS},
247 { "stime", 1, DEC, NOV, DEC},
248 { "pcsample", 2, DEC, NOV, HEX, DEC},
249 { "alarm", 1, DEC, NOV, UNS},
250 { "fstat", 2, DEC, NOV, DEC, HEX},
251 { "pause", 0, DEC, NOV},
252 { NULL, 8, HEX, HEX, HEX, HEX, HEX, HEX, HEX, HEX, HEX, HEX},
253 { "stty", 2, DEC, NOV, DEC, DEC},
254 { "gtty", 2, DEC, NOV, DEC, DEC},
255 { "access", 2, DEC, NOV, STG, ACC},
256 { "nice", 1, DEC, NOV, DEC},
257 { "statfs", 4, DEC, NOV, STG, HEX, DEC, DEC},
258 { "sync", 0, DEC, NOV},
259 { "kill", 2, DEC, NOV, DEC, SIG},
260 { "fstatfs", 4, DEC, NOV, DEC, HEX, DEC, DEC},
261 { "pgrp", 3, DEC, NOV, DEC, DEC, DEC},
262 { "uucopystr", 3, DEC, NOV, STG, RST, UNS},
263 { NULL, 8, HEX, HEX, HEX, HEX, HEX, HEX, HEX, HEX, HEX, HEX},
264 { "pipe", 2, DEC, NOV, PFD, PIP},
265 { "times", 1, DEC, NOV, HEX},
266 { "profil", 4, DEC, NOV, HEX, UNS, HEX, OCT},
267 { "faccessat", 4, DEC, NOV, ATC, STG, ACC, FAT},
268 { "setgid", 1, DEC, NOV, UNS},
269 { "getgid", 0, UNS, UNS},
270 { "mknodat", 4, DEC, NOV, ATC, STG, OCT, HEX},
271 { "msgsys", 6, DEC, NOV, DEC, DEC, DEC, DEC, DEC},
272 { "sysi86", 4, HEX, NOV, S86, HEX, HEX, HEX, DEC, DEC},
273 { "acct", 1, DEC, NOV, STG},
274 { "shmsys", 4, DEC, NOV, DEC, HEX, HEX, HEX},

```

```

275 { "semsys", 5, DEC, NOV, DEC, HEX, HEX, HEX, HEX},
276 { "ioctl", 3, DEC, NOV, DEC, IOC, IOA},
277 { "uadmin", 3, DEC, NOV, DEC, DEC, DEC},
278 { "fchownat", 5, DEC, NOV, ATC, STG, DEC, DEC, SNF},
279 { "utssys", 4, DEC, NOV, HEX, DEC, UTS, HEX},
280 { "fdsync", 2, DEC, NOV, DEC, FFG},
281 { "execve", 3, DEC, NOV, STG, HEX, HEX},
282 { "umask", 1, OCT, NOV, OCT},
283 { "chroot", 1, DEC, NOV, STG},
284 { "fcntl", 3, DEC, NOV, DEC, FCN, HEX},
285 { "ulimit", 2, DEC, NOV, ULM, DEC},
286 { "renameat", 4, DEC, NOV, ATC, STG, ATC, STG},
287 { "unlinkat", 3, DEC, NOV, ATC, STG, UAT},
288 { "fstatat", 4, DEC, NOV, ATC, STG, HEX, SNF},
289 { "fstatat64", 4, DEC, NOV, ATC, STG, HEX, SNF},
290 { "openat", 4, DEC, NOV, ATC, STG, OPN, OCT},
291 { "openat64", 4, DEC, NOV, ATC, STG, OPN, OCT},
292 { "tasksys", 5, DEC, NOV, DEC, DEC, DEC, HEX, DEC},
293 { "acctctl", 3, DEC, NOV, HEX, HEX, UNS},
294 { "exacctsys", 6, DEC, NOV, DEC, IDT, DEC, HEX, DEC, HEX},
295 { "getpagesizes", 2, DEC, NOV, HEX, DEC},
296 { "rctl", 6, DEC, NOV, RSC, STG, HEX, HEX, DEC, DEC},
297 { "sidsys", 4, UNS, UNS, DEC, DEC, DEC, DEC},
298 { NULL, 8, HEX, HEX, HEX, HEX, HEX, HEX, HEX, HEX, HEX, HEX},
299 { "lwp_park", 3, DEC, NOV, DEC, HEX, DEC},
300 { "sendfile", 5, DEC, NOV, DEC, DEC, HEX, DEC, HEX},
301 { "rmdir", 1, DEC, NOV, STG},
302 { "mkdir", 2, DEC, NOV, STG, OCT},
303 { "getdents", 3, DEC, NOV, DEC, HEX, UNS},
304 { "privsys", 5, HEX, NOV, DEC, DEC, DEC, HEX, DEC},
305 { "ucredsys", 3, DEC, NOV, DEC, DEC, HEX},
306 { "sysfs", 3, DEC, NOV, SFS, DEX, DEX},
307 { "getmsg", 4, DEC, NOV, DEC, HEX, HEX, HEX},
308 { "putmsg", 4, DEC, NOV, DEC, HEX, HEX, SMF},
309 { NULL, 8, HEX, HEX, HEX, HEX, HEX, HEX, HEX, HEX, HEX, HEX},
310 { "lstat", 2, DEC, NOV, STG, HEX},
311 { "symlink", 2, DEC, NOV, STG, STG},
312 { "readlink", 3, DEC, NOV, STG, RLK, UNS},
313 { "setgroups", 2, DEC, NOV, DEC, HEX},
314 { "getgroups", 2, DEC, NOV, DEC, HEX},
315 { "fchmod", 2, DEC, NOV, DEC, OCT},
316 { "fchown", 3, DEC, NOV, DEC, DEC, DEC},
317 { "sigprocmask", 3, DEC, NOV, SPM, HEX, HEX},
318 { "sigsuspend", 1, DEC, NOV, HEX},
319 { "sigaltstack", 2, DEC, NOV, HEX, HEX},
320 { "sigaction", 3, DEC, NOV, SIG, HEX, HEX},
321 { "sigpendsys", 2, DEC, NOV, DEC, HEX},
322 { "context", 2, DEC, NOV, DEC, HEX},
323 { "fchmodat", 4, DEC, NOV, ATC, STG, OCT, SNF},
324 { "mkdirat", 3, DEC, NOV, ATC, STG, OCT},
325 { "statvfs", 2, DEC, NOV, STG, HEX},
326 { "fstatvfs", 2, DEC, NOV, DEC, HEX},
327 { "getloadavg", 2, DEC, NOV, HEX, DEC},
328 { "nffsys", 2, DEC, NOV, DEC, HEX},
329 { "waitid", 4, DEC, NOV, IDT, DEC, HEX, WOP},
330 { "sigsendsys", 2, DEC, NOV, HEX, SIG},
331 { "hrtsys", 5, DEC, NOV, DEC, HEX, HEX, HEX, HEX},
332 { "utimesys", 5, DEC, NOV, DEC, HEX, HEX, HEX, HEX},
333 { "sigresend", 3, DEC, NOV, SIG, HEX, HEX},
334 { "prioctlsys", 5, DEC, NOV, DEC, HEX, DEC, PC4, PC5},
335 { "pathconf", 2, DEC, NOV, STG, PTC},
336 { "mincore", 3, DEC, NOV, HEX, UNS, HEX},
337 { "mmap", 6, HEX, NOV, HEX, UNS, MPR, MTY, DEC, DEC},
338 { "mprotect", 3, DEC, NOV, HEX, UNS, MPR},
339 { "munmap", 2, DEC, NOV, HEX, UNS},
340 { "fpathconf", 2, DEC, NOV, DEC, PTC},

```



```

341 {"vfork",          0, DEC, NOV},          /* 119 */
342 {"fchdir",        1, DEC, NOV, DEC},       /* 120 */
343 {"readv",         3, DEC, NOV, DEC, HEX, DEC}, /* 121 */
344 {"writev",        3, DEC, NOV, DEC, HEX, DEC}, /* 122 */
345 {"preadv",        4, DEC, NOV, DEC, HEX, DEC, DEC}, /* 123 */
346 {"pwritev",       4, DEC, NOV, DEC, HEX, DEC, DEC}, /* 124 */
347 {"NULL",          8, HEX, HEX, HEX, HEX, HEX, HEX, HEX, HEX}, /* 125 */
348 {"getrandom",     3, DEC, NOV, IOB, UNS, GRF}, /* 126 */
349 {"mmapobj",       5, DEC, NOV, DEC, MOB, HEX, HEX, HEX}, /* 127 */
350 {"setrlimit",     2, DEC, NOV, RLM, HEX}, /* 128 */
351 {"getrlimit",     2, DEC, NOV, RLM, HEX}, /* 129 */
352 {"lchown",        3, DEC, NOV, STG, DEC, DEC}, /* 130 */
353 {"memcntl",       6, DEC, NOV, HEX, UNS, MCF, MC4, MC5, DEC}, /* 131 */
354 {"getpmsg",       5, DEC, NOV, DEC, HEX, HEX, HEX, HEX}, /* 132 */
355 {"putpmsg",       5, DEC, NOV, DEC, HEX, HEX, DEC, HXH}, /* 133 */
356 {"rename",        2, DEC, NOV, STG, STG}, /* 134 */
357 {"uname",         1, DEC, NOV, HEX}, /* 135 */
358 {"setegid",       1, DEC, NOV, UNS}, /* 136 */
359 {"sysconfig",     1, DEC, NOV, CNF}, /* 137 */
360 {"adjtime",       2, DEC, NOV, HEX, HEX}, /* 138 */
361 {"sysinfo",       3, DEC, NOV, INF, RST, DEC}, /* 139 */
362 {"sharefs",      3, DEC, NOV, DEC, HEX, DEC}, /* 140 */
363 {"seteuid",       1, DEC, NOV, UNS}, /* 141 */
364 {"forksys",      2, DEC, NOV, DEC, HXH}, /* 142 */
365 {"NULL",          8, HEX, HEX, HEX, HEX, HEX, HEX, HEX, HEX}, /* 143 */
366 {"sigtimedwait", 3, DEC, NOV, HEX, HEX, HEX}, /* 144 */
367 {"lwp_info",      1, DEC, NOV, HEX}, /* 145 */
368 {"yield",         0, DEC, NOV}, /* 146 */
369 {"NULL",          8, HEX, HEX, HEX, HEX, HEX, HEX, HEX, HEX}, /* 147 */
370 {"lwp_sema_post", 1, DEC, NOV, HEX}, /* 148 */
371 {"lwp_sema_trywait", 1, DEC, NOV, HEX}, /* 149 */
372 {"lwp_detach",   1, DEC, NOV, DEC}, /* 150 */
373 {"corectl",      4, DEC, NOV, DEC, HEX, HEX, HEX}, /* 151 */
374 {"modctl",       5, DEC, NOV, MOD, HEX, HEX, HEX, HEX}, /* 152 */
375 {"fchroot",      1, DEC, NOV, DEC}, /* 153 */
376 {"NULL",          8, HEX, HEX, HEX, HEX, HEX, HEX, HEX, HEX}, /* 154 */
377 {"vhangup",      0, DEC, NOV}, /* 155 */
378 {"gettimeofday", 1, DEC, NOV, HEX}, /* 156 */
379 {"getitimer",    2, DEC, NOV, ITM, HEX}, /* 157 */
380 {"setitimer",    3, DEC, NOV, ITM, HEX, HEX}, /* 158 */
381 {"lwp_create",   3, DEC, NOV, HEX, LWF, HEX}, /* 159 */
382 {"lwp_exit",     0, DEC, NOV}, /* 160 */
383 {"lwp_suspend",  1, DEC, NOV, DEC}, /* 161 */
384 {"lwp_continue", 1, DEC, NOV, DEC}, /* 162 */
385 {"lwp_kill",    2, DEC, NOV, DEC, SIG}, /* 163 */
386 {"lwp_self",    0, DEC, NOV}, /* 164 */
387 {"lwp_sigmask",  5, HEX, HEX, SPM, HEX, HEX, HEX, HEX}, /* 165 */
388 {"lwp_private",  3, HEX, NOV, DEC, DEC, HEX}, /* 166 */
389 {"lwp_wait",    2, DEC, NOV, DEC, HEX}, /* 167 */
390 {"lwp_mutex_wakeup", 2, DEC, NOV, HEX, DEC}, /* 168 */
391 {"NULL",          8, HEX, HEX, HEX, HEX, HEX, HEX, HEX, HEX}, /* 169 */
392 {"lwp_cond_wait", 4, DEC, NOV, HEX, HEX, HEX, DEC}, /* 170 */
393 {"lwp_cond_signal", 1, DEC, NOV, HEX}, /* 171 */
394 {"lwp_cond_broadcast", 1, DEC, NOV, HEX}, /* 172 */
395 {"pread",        4, DEC, NOV, DEC, IOB, UNS, DEX}, /* 173 */
396 {"pwrite",       4, DEC, NOV, DEC, IOB, UNS, DEX}, /* 174 */
397 {"llseek",      4, LLO, NOV, DEC, LLO, HID, WHN}, /* 175 */
398 {"inst_sync",   2, DEC, NOV, STG, DEC}, /* 176 */
399 {"brand",       6, DEC, NOV, DEC, HEX, HEX, HEX, HEX}, /* 177 */
400 {"kaio",        7, DEC, NOV, AIO, HEX, HEX, HEX, HEX, HEX}, /* 178 */
401 {"cpc",         5, DEC, NOV, CPC, DEC, HEX, HEX, HEX}, /* 179 */
402 {"lgrpsys",     3, DEC, NOV, DEC, DEC, HEX}, /* 180 */
403 {"rusagesys",   5, DEC, NOV, DEC, HEX, DEC, HEX, HEX}, /* 181 */
404 {"portfs",      6, HEX, HEX, DEC, HEX, HEX, HEX, HEX}, /* 182 */
405 {"pollsys",     4, DEC, NOV, HEX, DEC, HEX, HEX}, /* 183 */
406 {"labelsys",    2, DEC, NOV, DEC, HEX}, /* 184 */

```

```

407 {"acl",          4, DEC, NOV, STG, ACL, DEC, HEX}, /* 185 */
408 {"auditsys",     4, DEC, NOV, AUD, HEX, HEX, HEX}, /* 186 */
409 {"processor_bind", 4, DEC, NOV, IDT, DEC, DEC, HEX}, /* 187 */
410 {"processor_info", 2, DEC, NOV, DEC, HEX}, /* 188 */
411 {"p_online",     2, DEC, NOV, DEC, DEC}, /* 189 */
412 {"sigqueue",     5, DEC, NOV, DEC, SIG, HEX, SQC, DEC}, /* 190 */
413 {"clock_gettime", 2, DEC, NOV, DEC, HEX}, /* 191 */
414 {"clock_settime", 2, DEC, NOV, DEC, HEX}, /* 192 */
415 {"clock_getres", 2, DEC, NOV, DEC, HEX}, /* 193 */
416 {"timer_create", 3, DEC, NOV, DEC, HEX, HEX}, /* 194 */
417 {"timer_delete", 1, DEC, NOV, DEC}, /* 195 */
418 {"timer_settime", 4, DEC, NOV, DEC, DEC, HEX, HEX}, /* 196 */
419 {"timer_gettime", 2, DEC, NOV, DEC, HEX}, /* 197 */
420 {"timer_getoverrun", 1, DEC, NOV, DEC}, /* 198 */
421 {"nanosleep",    2, DEC, NOV, HEX, HEX}, /* 199 */
422 {"facl",        4, DEC, NOV, DEC, ACL, DEC, HEX}, /* 200 */
423 {"door",        6, DEC, NOV, DEC, HEX, HEX, HEX, DEC}, /* 201 */
424 {"setreuid",    2, DEC, NOV, UNL, UNL}, /* 202 */
425 {"setregid",    2, DEC, NOV, UNL, UNL}, /* 203 */
426 {"install_utrap", 3, DEC, NOV, DEC, HEX, HEX}, /* 204 */
427 {"signotify",   3, DEC, NOV, DEC, HEX, HEX}, /* 205 */
428 {"schedctl",    0, HEX, NOV}, /* 206 */
429 {"pset",        5, DEC, NOV, DEC, HEX, HEX, HEX, HEX}, /* 207 */
430 {"sparc_utrap_install", 5, DEC, NOV, UTT, UTH, UTH, HEX, HEX}, /* 208 */
431 {"resolvepath", 3, DEC, NOV, STG, RLK, DEC}, /* 209 */
432 {"lwp_mutex_timedlock", 3, DEC, NOV, HEX, HEX, HEX}, /* 210 */
433 {"lwp_sema_timedwait", 3, DEC, NOV, HEX, HEX, DEC}, /* 211 */
434 {"lwp_rwlock_sys", 3, DEC, NOV, DEC, HEX, HEX}, /* 212 */
435 {"getdents64",  3, DEC, NOV, DEC, HEX, UNS}, /* 213 */
436 {"mmap64",      7, HEX, NOV, HEX, UNS, MPR, MTY, DEC, LLO, HID}, /* 214 */
437 {"stat64",     2, DEC, NOV, STG, HEX}, /* 215 */
438 {"lstat64",    2, DEC, NOV, STG, HEX}, /* 216 */
439 {"fstat64",    2, DEC, NOV, DEC, HEX}, /* 217 */
440 {"statvfs64",  2, DEC, NOV, STG, HEX}, /* 218 */
441 {"fstatvfs64", 2, DEC, NOV, DEC, HEX}, /* 219 */
442 {"setrlimit64", 2, DEC, NOV, RLM, HEX}, /* 220 */
443 {"getrlimit64", 2, DEC, NOV, RLM, HEX}, /* 221 */
444 {"pread64",    5, DEC, NOV, DEC, IOB, UNS, LLO, HID}, /* 222 */
445 {"pwrite64",   5, DEC, NOV, DEC, IOB, UNS, LLO, HID}, /* 223 */
446 {"NULL",       8, HEX, HEX, HEX, HEX, HEX, HEX, HEX, HEX}, /* 224 */
447 {"open64",     3, DEC, NOV, STG, OPN, OCT}, /* 225 */
448 {"rpcmod",     3, DEC, NOV, DEC, HEX}, /* 226 */
449 {"zone",       5, DEC, NOV, DEC, HEX, HEX, HEX, HEX}, /* 227 */
450 {"autofs",     2, DEC, NOV, DEC, HEX}, /* 228 */
451 {"getcwd",     3, DEC, NOV, RST, DEC}, /* 229 */
452 {"so_socket",  5, DEC, NOV, PFM, SKT, SKP, STG, SKV}, /* 230 */
453 {"so_socketpair", 1, DEC, NOV, HEX}, /* 231 */
454 {"bind",       4, DEC, NOV, DEC, HEX, DEC, SKV}, /* 232 */
455 {"listen",     3, DEC, NOV, DEC, DEC, SKV}, /* 233 */
456 {"accept",     5, DEC, NOV, DEC, HEX, HEX, SKV, ACF}, /* 234 */
457 {"connect",    4, DEC, NOV, DEC, HEX, DEC, SKV}, /* 235 */
458 {"shutdown",   3, DEC, NOV, DEC, SHT, SKV}, /* 236 */
459 {"recv",       4, DEC, NOV, DEC, IOB, DEC, DEC}, /* 237 */
460 {"recvfrom",   6, DEC, NOV, DEC, IOB, DEC, DEC, HEX, HEX}, /* 238 */
461 {"recvmmsg",   3, DEC, NOV, DEC, HEX, DEC}, /* 239 */
462 {"send",       4, DEC, NOV, DEC, IOB, DEC, DEC}, /* 240 */
463 {"sendmsg",    3, DEC, NOV, DEC, HEX, DEC}, /* 241 */
464 {"sendto",     6, DEC, NOV, DEC, IOB, DEC, DEC, HEX, DEC}, /* 242 */
465 {"getpeername", 4, DEC, NOV, DEC, HEX, HEX, SKV}, /* 243 */
466 {"getsockname", 4, DEC, NOV, DEC, HEX, HEX, SKV}, /* 244 */
467 {"getsockopt",  6, DEC, NOV, DEC, SOL, SON, HEX, HEX, SKV}, /* 245 */
468 {"setsockopt",  6, DEC, NOV, DEC, SOL, SON, HEX, DEC, SKV}, /* 246 */
469 {"sockconfig",  5, DEC, NOV, DEC, HEX, HEX, HEX, HEX}, /* 247 */
470 {"ntp_gettime", 1, DEC, NOV, HEX}, /* 248 */
471 {"ntp_adjtime", 1, DEC, NOV, HEX}, /* 249 */
472 {"lwp_mutex_unlock", 1, DEC, NOV, HEX}, /* 250 */

```

new/usr/src/cmd/truss/systable.c

5

```
473 {"lwp_mutex_trylock", 2, DEC, NOV, HEX, HEX}, /* 251 */
474 {"lwp_mutex_register", 2, DEC, NOV, HEX, HEX}, /* 252 */
475 {"cladm", 3, DEC, NOV, CLC, CLF, HEX}, /* 253 */
476 {"uucopy", 3, DEC, NOV, HEX, HEX, UNS}, /* 254 */
477 {"umount2", 2, DEC, NOV, STG, MTF}, /* 255 */
478 { NULL, -1, DEC, NOV},
479 };
unchanged_portion_omitted
```

new/usr/src/cmd/zoneadmd/vplat.c

1

```
*****
147442 Thu Jun 30 22:01:48 2016
new/usr/src/cmd/zoneadmd/vplat.c
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
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 /*
23  * Copyright (c) 2003, 2010, Oracle and/or its affiliates. All rights reserved.
24  * Copyright (c) 2013, Joyent Inc. All rights reserved.
25  * Copyright (c) 2015 by Delphix. All rights reserved.
26 */
27
28 /*
29  * Copyright 2011 Nexenta Systems, Inc. All rights reserved.
30 */
31
32 /*
33  * This module contains functions used to bring up and tear down the
34  * Virtual Platform: [un]mounting file-systems, [un]plumbing network
35  * interfaces, [un]configuring devices, establishing resource controls,
36  * and creating/destroying the zone in the kernel. These actions, on
37  * the way up, ready the zone; on the way down, they halt the zone.
38  * See the much longer block comment at the beginning of zoneadmd.c
39  * for a bigger picture of how the whole program functions.
40  *
41  * This module also has primary responsibility for the layout of "scratch
42  * zones." These are mounted, but inactive, zones that are used during
43  * operating system upgrade and potentially other administrative action. The
44  * scratch zone environment is similar to the miniroot environment. The zone's
45  * actual root is mounted read-write on /a, and the standard paths (/usr,
46  * /sbin, /lib) all lead to read-only copies of the running system's binaries.
47  * This allows the administrative tools to manipulate the zone using "-R /a"
48  * without relying on any binaries in the zone itself.
49  *
50  * If the scratch zone is on an alternate root (Live Upgrade [LU] boot
51  * environment), then we must resolve the lofs mounts used there to uncover
52  * writable (unshared) resources. Shared resources, though, are always
53  * read-only. In addition, if the "same" zone with a different root path is
54  * currently running, then "/b" inside the zone points to the running zone's
55  * root. This allows LU to synchronize configuration files during the upgrade
56  * process.
57  *
58  * To construct this environment, this module creates a tmpfs mount on
```

new/usr/src/cmd/zoneadmd/vplat.c

2

```
59  * $ZONEPATH/lu. Inside this scratch area, the miniroot-like environment as
60  * described above is constructed on the fly. The zone is then created using
61  * $ZONEPATH/lu as the root.
62  *
63  * Note that scratch zones are inactive. The zone's bits are not running and
64  * likely cannot be run correctly until upgrade is done. Init is not running
65  * there, nor is SMF. Because of this, the "mounted" state of a scratch zone
66  * is not a part of the usual halt/ready/boot state machine.
67  */
68
69 #include <sys/param.h>
70 #include <sys/mount.h>
71 #include <sys/mntent.h>
72 #include <sys/socket.h>
73 #include <sys/utsname.h>
74 #include <sys/types.h>
75 #include <sys/stat.h>
76 #include <sys/sockio.h>
77 #include <sys/stropts.h>
78 #include <sys/conf.h>
79 #include <sys/systeminfo.h>
80 #include <sys/secflags.h>
81 #endif /* !codereview */
82
83 #include <libldpi.h>
84 #include <libdllink.h>
85 #include <libdlvlan.h>
86
87 #include <inet/tcp.h>
88 #include <arpa/inet.h>
89 #include <netinet/in.h>
90 #include <net/route.h>
91
92 #include <stdio.h>
93 #include <errno.h>
94 #include <fcntl.h>
95 #include <unistd.h>
96 #include <rctl.h>
97 #include <stdlib.h>
98 #include <string.h>
99 #include <strings.h>
100 #include <wait.h>
101 #include <limits.h>
102 #include <libgen.h>
103 #include <libzfs.h>
104 #include <libdevinfo.h>
105 #include <zone.h>
106 #include <assert.h>
107 #include <libcontract.h>
108 #include <libcontract_priv.h>
109 #include <uuid/uuid.h>
110
111 #include <sys/mntio.h>
112 #include <sys/mnttab.h>
113 #include <sys/fs/autofs.h> /* for _autofssys() */
114 #include <sys/fs/lofs_info.h>
115 #include <sys/fs/zfs.h>
116
117 #include <pool.h>
118 #include <sys/pool.h>
119 #include <sys/priocntl.h>
120
121 #include <libbrand.h>
122 #include <sys/brand.h>
123 #include <libzonecfg.h>
124 #include <synch.h>
```

```

126 #include "zoneadmd.h"
127 #include <tsol/label.h>
128 #include <libtsnet.h>
129 #include <sys/priv.h>
130 #include <libinetutil.h>

132 #define V4_ADDR_LEN    32
133 #define V6_ADDR_LEN    128

135 #define RESOURCE_DEFAULT_OPTS \
136     MNTOPT_RO ", " MNTOPT_LOFS_NOSUB ", " MNTOPT_NODEVICES

138 #define DFSTYPES        "/etc/dfs/fstypes"
139 #define MAXTNZLEN        2048

141 #define ALT_MOUNT(mount_cmd)    ((mount_cmd) != Z_MNT_BOOT)

143 /* a reasonable estimate for the number of lwps per process */
144 #define LWPS_PER_PROCESS    10

146 /* for routing socket */
147 static int rts_seqno = 0;

149 /* mangled zone name when mounting in an alternate root environment */
150 static char kernzone[ZONENAME_MAX];

152 /* array of cached mount entries for resolve_lofs */
153 static struct mnttab *resolve_lofs_mnts, *resolve_lofs_mnt_max;

155 /* for Trusted Extensions */
156 static tsol_zcent_t *get_zone_label(zlog_t *, priv_set_t *);
157 static int tsol_mounts(zlog_t *, char *, char *);
158 static void tsol_unmounts(zlog_t *, char *);

160 static m_label_t *zlabel = NULL;
161 static m_label_t *zid_label = NULL;
162 static priv_set_t *zprivs = NULL;

164 static const char *DFLT_FS_ALLOWED = "hsfs,smbfs,nfs,nfs3,nfs4,nfsdyn";

166 /* from libsocket, not in any header file */
167 extern int getnetmaskbyaddr(struct in_addr, struct in_addr *);

169 /* from zoneadmd */
170 extern char query_hook[];

172 /*
173  * For each "net" resource configured in zonecfg, we track a zone_addr_list_t
174  * node in a linked list that is sorted by linkid. The list is constructed as
175  * the xml configuration file is parsed, and the information
176  * contained in each node is added to the kernel before the zone is
177  * booted, to be retrieved and applied from within the exclusive-IP NGZ
178  * on boot.
179  */
180 typedef struct zone_addr_list {
181     struct zone_addr_list *za_next;
182     datalink_id_t za_linkid; /* datalink_id_t of interface */
183     struct zone_nwifftab za_nwifftab; /* address, defrouter properties */
184 } zone_addr_list_t;

186 /*
187  * An optimization for build_mnttable: reallocate (and potentially copy the
188  * data) only once every N times through the loop.
189  */
190 #define MNTTAB_HUNK    32

```

```

192 /* some handy macros */
193 #define SIN(s)    ((struct sockaddr_in *)s)
194 #define SIN6(s)  ((struct sockaddr_in6 *)s)

196 /*
197  * Private autofss system call
198  */
199 extern int _autofssys(int, void *);

201 static int
202 autofss_cleanup(zoneid_t zoneid)
203 {
204     /*
205      * Ask autofss to unmount all trigger nodes in the given zone.
206      */
207     return (_autofssys(AUTOFS_UNMOUNTALL, (void *)zoneid));
208 }

210 static void
211 free_mnttable(struct mnttab *mnt_array, uint_t nelemp)
212 {
213     uint_t i;

215     if (mnt_array == NULL)
216         return;
217     for (i = 0; i < nelemp; i++) {
218         free(mnt_array[i].mnt_mountp);
219         free(mnt_array[i].mnt_fstype);
220         free(mnt_array[i].mnt_special);
221         free(mnt_array[i].mnt_mntopts);
222         assert(mnt_array[i].mnt_time == NULL);
223     }
224     free(mnt_array);
225 }

227 /*
228  * Build the mount table for the zone rooted at "zroot", storing the resulting
229  * array of struct mnttab in "mnt_array" and the number of elements in the
230  * array in "nelemp".
231  */
232 static int
233 build_mnttable(zlog_t *zlogp, const char *zroot, size_t zrootlen, FILE *mnttab,
234               struct mnttab **mnt_arrayp, uint_t *nelemp)
235 {
236     struct mnttab mnt;
237     struct mnttab *mnts;
238     struct mnttab *mnp;
239     uint_t nmnt;

241     rewind(mnttab);
242     resetmnttab(mnttab);
243     nmnt = 0;
244     mnts = NULL;
245     while (getmntent(mnttab, &mnt) == 0) {
246         struct mnttab *tmp_array;

248         if (strcmp(mnt.mnt_mountp, zroot, zrootlen) != 0)
249             continue;
250         if (nmnt % MNTTAB_HUNK == 0) {
251             tmp_array = realloc(mnts,
252                               (nmnt + MNTTAB_HUNK) * sizeof (*mnts));
253             if (tmp_array == NULL) {
254                 free_mnttable(mnts, nmnt);
255                 return (-1);
256             }

```

```

257     mnts = tmp_array;
258 }
259 mnp = &mnts[nmnt++];

261 /*
262  * Zero out any fields we're not using.
263  */
264 (void) memset(mnp, 0, sizeof (*mnp));

266 if (mnt.mnt_special != NULL)
267     mnp->mnt_special = strdup(mnt.mnt_special);
268 if (mnt.mnt_mntopts != NULL)
269     mnp->mnt_mntopts = strdup(mnt.mnt_mntopts);
270 mnp->mnt_mountp = strdup(mnt.mnt_mountp);
271 mnp->mnt_fstype = strdup(mnt.mnt_fstype);
272 if ((mnt.mnt_special != NULL && mnp->mnt_special == NULL) ||
273     (mnt.mnt_mntopts != NULL && mnp->mnt_mntopts == NULL) ||
274     (mnt.mnt_mountp == NULL || mnp->mnt_mountp == NULL) ||
275     (mnt.mnt_fstype == NULL || mnp->mnt_fstype == NULL)) {
276     zerror(zlogp, B_TRUE, "memory allocation failed");
277     free_mnttable(mnts, nmnt);
278     return (-1);
279 }
280 *mnt_arrayp = mnts;
281 *nelemp = nmnt;
282 return (0);
283 }

285 /*
286  * This is an optimization. The resolve_lofs function is used quite frequently
287  * to manipulate file paths, and on a machine with a large number of zones,
288  * there will be a huge number of mounted file systems. Thus, we trigger a
289  * reread of the list of mount points
290  */
291 static void
292 lofs_discard_mnttab(void)
293 {
294     free_mnttable(resolve_lofs_mnts,
295                 resolve_lofs_mnt_max - resolve_lofs_mnts);
296     resolve_lofs_mnts = resolve_lofs_mnt_max = NULL;
297 }

299 static int
300 lofs_read_mnttab(zlog_t *zlogp)
301 {
302     FILE *mnttab;
303     uint_t nmnts;

305     if ((mnttab = fopen(MNTTAB, "r")) == NULL)
306         return (-1);
307     if (build_mnttable(zlogp, "", 0, mnttab, &resolve_lofs_mnts,
308                     &nmnts) == -1) {
309         (void) fclose(mnttab);
310         return (-1);
311     }
312     (void) fclose(mnttab);
313     resolve_lofs_mnt_max = resolve_lofs_mnts + nmnts;
314     return (0);
315 }

317 /*
318  * This function loops over potential loopback mounts and symlinks in a given
319  * path and resolves them all down to an absolute path.
320  */
321 void
322 resolve_lofs(zlog_t *zlogp, char *path, size_t pathlen)

```

```

323 {
324     int len, arlen;
325     const char *alroot;
326     char tmppath[MAXPATHLEN];
327     boolean_t outside_alroot;

329     if ((len = resolvepath(path, tmppath, sizeof (tmppath))) == -1)
330         return;
331     tmppath[len] = '\0';
332     (void) strcpy(path, tmppath, sizeof (tmppath));

334     /* This happens once per zoneadmd operation. */
335     if (resolve_lofs_mnts == NULL && lofs_read_mnttab(zlogp) == -1)
336         return;

338     alroot = zonecfg_get_root();
339     arlen = strlen(alroot);
340     outside_alroot = B_FALSE;
341     for (;;) {
342         struct mnttab *mnp;

344         /* Search in reverse order to find longest match */
345         for (mnp = resolve_lofs_mnt_max - 1; mnp >= resolve_lofs_mnts;
346             mnp--) {
347             if (mnp->mnt_fstype == NULL ||
348                 mnp->mnt_mountp == NULL ||
349                 mnp->mnt_special == NULL)
350                 continue;
351             len = strlen(mnp->mnt_mountp);
352             if (strncmp(mnp->mnt_mountp, path, len) == 0 &&
353                 (path[len] == '/' || path[len] == '\0'))
354                 break;
355         }
356         if (mnp < resolve_lofs_mnts)
357             break;
358         /* If it's not a lofs then we're done */
359         if (strcmp(mnp->mnt_fstype, MNTTYPE_LOFS) != 0)
360             break;
361         if (outside_alroot) {
362             char *cp;
363             int olen = sizeof (MNTOPT_RO) - 1;

365             /*
366              * If we run into a read-only mount outside of the
367              * alternate root environment, then the user doesn't
368              * want this path to be made read-write.
369              */
370             if (mnp->mnt_mntopts != NULL &&
371                 (cp = strstr(mnp->mnt_mntopts, MNTOPT_RO)) !=
372                 NULL &&
373                 (cp == mnp->mnt_mntopts || cp[-1] == ',') &&
374                 (cp[olen] == '\0' || cp[olen] == ',')) {
375                 break;
376             }
377         } else if (arlen > 0 &&
378                 (strncmp(mnp->mnt_special, alroot, arlen) != 0 ||
379                  (mnp->mnt_special[arlen] != '\0' &&
380                   mnp->mnt_special[arlen] != '/'))) {
381             outside_alroot = B_TRUE;
382         }
383     }
384     /* use temporary buffer because new path might be longer */
385     (void) snprintf(tmppath, sizeof (tmppath), "%s%s",
386                  mnp->mnt_special, path + len);
387     if ((len = resolvepath(tmppath, path, pathlen)) == -1)
388         break;
389     path[len] = '\0';

```

```

389     }
390 }

392 /*
393  * For a regular mount, check if a replacement lofs mount is needed because the
394  * referenced device is already mounted somewhere.
395  */
396 static int
397 check_lofs_needed(zlog_t *zlogp, struct zone_fstab *fsptr)
398 {
399     struct mnttab *mnp;
400     zone_fsopt_t *optptr, *onext;

402     /* This happens once per zoneadmd operation. */
403     if (resolve_lofs_mnts == NULL && lofs_read_mnttab(zlogp) == -1)
404         return (-1);

406     /*
407      * If this special node isn't already in use, then it's ours alone;
408      * no need to worry about conflicting mounts.
409      */
410     for (mnp = resolve_lofs_mnts; mnp < resolve_lofs_mnt_max;
411          mnp++) {
412         if (strcmp(mnp->mnt_special, fsptr->zone_fs_special) == 0)
413             break;
414     }
415     if (mnp >= resolve_lofs_mnt_max)
416         return (0);

418     /*
419      * Convert this duplicate mount into a lofs mount.
420      */
421     (void) strcpy(fsptr->zone_fs_special, mnp->mnt_mountp,
422                 sizeof (fsptr->zone_fs_special));
423     (void) strcpy(fsptr->zone_fs_type, MNTTYPE_LOFS,
424                 sizeof (fsptr->zone_fs_type));
425     fsptr->zone_fs_raw[0] = '\0';

427     /*
428      * Discard all but one of the original options and set that to our
429      * default set of options used for resources.
430      */
431     optptr = fsptr->zone_fs_options;
432     if (optptr == NULL) {
433         optptr = malloc(sizeof (*optptr));
434         if (optptr == NULL) {
435             zerror(zlogp, B_TRUE, "cannot mount %s",
436                  fsptr->zone_fs_dir);
437             return (-1);
438         }
439     } else {
440         while ((onext = optptr->zone_fsopt_next) != NULL) {
441             optptr->zone_fsopt_next = onext->zone_fsopt_next;
442             free(onext);
443         }
444     }
445     (void) strcpy(optptr->zone_fsopt_opt, RESOURCE_DEFAULT_OPTS);
446     optptr->zone_fsopt_next = NULL;
447     fsptr->zone_fs_options = optptr;
448     return (0);
449 }

451 int
452 make_one_dir(zlog_t *zlogp, const char *prefix, const char *subdir, mode_t mode,
453             uid_t userid, gid_t groupid)
454 {

```

```

455     char path[MAXPATHLEN];
456     struct stat st;

458     if (sprintf(path, sizeof (path), "%s%s", prefix, subdir) >
459         sizeof (path)) {
460         zerror(zlogp, B_FALSE, "pathname %s%s is too long", prefix,
461              subdir);
462         return (-1);
463     }

465     if (lstat(path, &st) == 0) {
466         /*
467          * We don't check the file mode since presumably the zone
468          * administrator may have had good reason to change the mode,
469          * and we don't need to second guess him.
470          */
471         if (!S_ISDIR(st.st_mode)) {
472             if (S_ISREG(st.st_mode)) {
473                 /*
474                  * Allow readonly mounts of /etc/ files; this
475                  * is needed most by Trusted Extensions.
476                  */
477                 if (strcmp(subdir, "/etc/",
478                          strlen("/etc/") != 0) {
479                     zerror(zlogp, B_FALSE,
480                          "%s is not in /etc", path);
481                     return (-1);
482                 }
483             } else {
484                 zerror(zlogp, B_FALSE,
485                      "%s is not a directory", path);
486                 return (-1);
487             }
488         }
489         return (0);
490     }

492     if (mkdirp(path, mode) != 0) {
493         if (errno == EROFS)
494             zerror(zlogp, B_FALSE, "Could not mkdir %s.\nIt is on a
495                  "read-only file system in this local zone.\nMake
496                  "sure %s exists in the global zone.", path, subdir);
497         else
498             zerror(zlogp, B_TRUE, "mkdirp of %s failed", path);
499         return (-1);
500     }

502     (void) chown(path, userid, groupid);
503     return (0);
504 }

506 static void
507 free_remote_fstypes(char **types)
508 {
509     uint_t i;

511     if (types == NULL)
512         return;
513     for (i = 0; types[i] != NULL; i++)
514         free(types[i]);
515     free(types);
516 }

518 static char **
519 get_remote_fstypes(zlog_t *zlogp)
520 {

```

```

521 char **types = NULL;
522 FILE *fp;
523 char buf[MAXPATHLEN];
524 char fstype[MAXPATHLEN];
525 uint_t lines = 0;
526 uint_t i;

528 if ((fp = fopen(DFSTYPES, "r")) == NULL) {
529     zerror(zlogp, B_TRUE, "failed to open %s", DFSTYPES);
530     return (NULL);
531 }
532 /*
533  * Count the number of lines
534  */
535 while (fgets(buf, sizeof (buf), fp) != NULL)
536     lines++;
537 if (lines == 0) /* didn't read anything; empty file */
538     goto out;
539 rewind(fp);
540 /*
541  * Allocate enough space for a NULL-terminated array.
542  */
543 types = calloc(lines + 1, sizeof (char *));
544 if (types == NULL) {
545     zerror(zlogp, B_TRUE, "memory allocation failed");
546     goto out;
547 }
548 i = 0;
549 while (fgets(buf, sizeof (buf), fp) != NULL) {
550     /* LINTED - fstype is big enough to hold buf */
551     if (sscanf(buf, "%s", fstype) == 0) {
552         zerror(zlogp, B_FALSE, "unable to parse %s", DFSTYPES);
553         free_remote_fstypes(types);
554         types = NULL;
555         goto out;
556     }
557     types[i] = strdup(fstype);
558     if (types[i] == NULL) {
559         zerror(zlogp, B_TRUE, "memory allocation failed");
560         free_remote_fstypes(types);
561         types = NULL;
562         goto out;
563     }
564     i++;
565 }
566 out:
567 (void) fclose(fp);
568 return (types);
569 }

571 static boolean_t
572 is_remote_fstype(const char *fstype, char *const *remote_fstypes)
573 {
574     uint_t i;

576     if (remote_fstypes == NULL)
577         return (B_FALSE);
578     for (i = 0; remote_fstypes[i] != NULL; i++) {
579         if (strcmp(remote_fstypes[i], fstype) == 0)
580             return (B_TRUE);
581     }
582     return (B_FALSE);
583 }

585 /*
586  * This converts a zone root path (normally of the form ../root) to a Live

```

```

587 * Upgrade scratch zone root (of the form ../lu).
588 */
589 static void
590 root_to_lu(zlog_t *zlogp, char *zroot, size_t zrootlen, boolean_t isresolved)
591 {
592     if (!isresolved && zonecfg_in_alt_root())
593         resolve_lofs(zlogp, zroot, zrootlen);
594     (void) strcpy(strrchr(zroot, '/') + 1, "lu");
595 }

597 /*
598  * The general strategy for unmounting filesystems is as follows:
599  *
600  * - Remote filesystems may be dead, and attempting to contact them as
601  * part of a regular unmount may hang forever; we want to always try to
602  * forcibly unmount such filesystems and only fall back to regular
603  * unmounts if the filesystem doesn't support forced unmounts.
604  *
605  * - We don't want to unnecessarily corrupt metadata on local
606  * filesystems (ie UFS), so we want to start off with graceful unmounts,
607  * and only escalate to doing forced unmounts if we get stuck.
608  *
609  * We start off walking backwards through the mount table. This doesn't
610  * give us strict ordering but ensures that we try to unmount submounts
611  * first. We thus limit the number of failed unmount2(2) calls.
612  *
613  * The mechanism for determining if we're stuck is to count the number
614  * of failed unmounts each iteration through the mount table. This
615  * gives us an upper bound on the number of filesystems which remain
616  * mounted (autofs trigger nodes are dealt with separately). If at the
617  * end of one unmount+autofs_cleanup cycle we still have the same number
618  * of mounts that we started out with, we're stuck and try a forced
619  * unmount. If that fails (filesystem doesn't support forced unmounts)
620  * then we bail and are unable to teardown the zone. If it succeeds,
621  * we're no longer stuck so we continue with our policy of trying
622  * graceful mounts first.
623  *
624  * Zone must be down (ie, no processes or threads active).
625  */
626 static int
627 unmount_filesystems(zlog_t *zlogp, zoneid_t zoneid, boolean_t unmount_cmd)
628 {
629     int error = 0;
630     FILE *mnttab;
631     struct mnttab *mnts;
632     uint_t nmnt;
633     char zroot[MAXPATHLEN + 1];
634     size_t zrootlen;
635     uint_t oldcount = UINT_MAX;
636     boolean_t stuck = B_FALSE;
637     char **remote_fstypes = NULL;

639     if (zone_get_rootpath(zone_name, zroot, sizeof (zroot)) != Z_OK) {
640         zerror(zlogp, B_FALSE, "unable to determine zone root");
641         return (-1);
642     }
643     if (unmount_cmd)
644         root_to_lu(zlogp, zroot, sizeof (zroot), B_FALSE);

646     (void) strcat(zroot, "/");
647     zrootlen = strlen(zroot);

649     /*
650      * For Trusted Extensions unmount each higher level zone's mount
651      * of our zone's /export/home
652      */

```

```

653     if (!unmount_cmd)
654         tsol_unmounts(zlogp, zone_name);

656     if ((mnttab = fopen(MNTTAB, "r")) == NULL) {
657         zerror(zlogp, B_TRUE, "failed to open %s", MNTTAB);
658         return (-1);
659     }
660     /*
661     * Use our hacky mntfs ioctl so we see everything, even mounts with
662     * MS_NOMNTTAB.
663     */
664     if (ioctl(fileno(mnttab), MNTIOC_SHOWHIDDEN, NULL) < 0) {
665         zerror(zlogp, B_TRUE, "unable to configure %s", MNTTAB);
666         error++;
667         goto out;
668     }

670     /*
671     * Build the list of remote fstypes so we know which ones we
672     * should forcibly unmount.
673     */
674     remote_fstypes = get_remote_fstypes(zlogp);
675     for (; /* ever */; ) {
676         uint_t newcount = 0;
677         boolean_t unmounted;
678         struct mnttab *mnp;
679         char *path;
680         uint_t i;

682         mnts = NULL;
683         nmnt = 0;
684         /*
685         * MNTTAB gives us a way to walk through mounted
686         * filesystems; we need to be able to walk them in
687         * reverse order, so we build a list of all mounted
688         * filesystems.
689         */
690         if (build_mnttable(zlogp, zroot, zrootlen, mnttab, &mnts,
691             &nmnt) != 0) {
692             error++;
693             goto out;
694         }
695         for (i = 0; i < nmnt; i++) {
696             mnp = &mnts[nmnt - i - 1]; /* access in reverse order */
697             path = mnp->mnt_mountp;
698             unmounted = B_FALSE;
699             /*
700             * Try forced unmount first for remote filesystems.
701             *
702             * Not all remote filesystems support forced unmounts,
703             * so if this fails (ENOTSUP) we'll continue on
704             * and try a regular unmount.
705             */
706             if (is_remote_fstype(mnp->mnt_fstype, remote_fstypes)) {
707                 if (umount2(path, MS_FORCE) == 0)
708                     unmounted = B_TRUE;
709             }
710             /*
711             * Try forced unmount if we're stuck.
712             */
713             if (stuck) {
714                 if (umount2(path, MS_FORCE) == 0) {
715                     unmounted = B_TRUE;
716                     stuck = B_FALSE;
717                 } else {
718                     /*

```

```

719         * The first failure indicates a
720         * mount we won't be able to get
721         * rid of automatically, so we
722         * bail.
723         */
724         error++;
725         zerror(zlogp, B_FALSE,
726             "unable to unmount '%s'", path);
727         free_mnttable(mnts, nmnt);
728         goto out;
729     }
730     }
731     /*
732     * Try regular unmounts for everything else.
733     */
734     if (!unmounted && umount2(path, 0) != 0)
735         newcount++;
736     }
737     free_mnttable(mnts, nmnt);

739     if (newcount == 0)
740         break;
741     if (newcount >= oldcount) {
742         /*
743         * Last round didn't unmount anything; we're stuck and
744         * should start trying forced unmounts.
745         */
746         stuck = B_TRUE;
747     }
748     oldcount = newcount;

750     /*
751     * Autofs doesn't let you unmount its trigger nodes from
752     * userland so we have to tell the kernel to cleanup for us.
753     */
754     if (autofs_cleanup(zoneid) != 0) {
755         zerror(zlogp, B_TRUE, "unable to remove autofs nodes");
756         error++;
757         goto out;
758     }
759     }

761 out:
762     free_remote_fstypes(remote_fstypes);
763     (void) fclose(mnttab);
764     return (error ? -1 : 0);
765 }

767 static int
768 fs_compare(const void *m1, const void *m2)
769 {
770     struct zone_fstab *i = (struct zone_fstab *)m1;
771     struct zone_fstab *j = (struct zone_fstab *)m2;

773     return (strcmp(i->zone_fs_dir, j->zone_fs_dir));
774 }

776 /*
777 * Fork and exec (and wait for) the mentioned binary with the provided
778 * arguments. Returns (-1) if something went wrong with fork(2) or exec(2),
779 * returns the exit status otherwise.
780 *
781 * If we were unable to exec the provided pathname (for whatever
782 * reason), we return the special token ZEXIT_EXEC. The current value
783 * of ZEXIT_EXEC doesn't conflict with legitimate exit codes of the
784 * consumers of this function; any future consumers must make sure this

```



```

785  * remains the case.
786  */
787  static int
788  forkexec(zlog_t *zlogp, const char *path, char *const argv[])
789  {
790      pid_t child_pid;
791      int child_status = 0;
792
793      /*
794       * Do not let another thread localize a message while we are forking.
795       */
796      (void) mutex_lock(&msglock);
797      child_pid = fork();
798      (void) mutex_unlock(&msglock);
799      if (child_pid == -1) {
800          zerror(zlogp, B_TRUE, "could not fork for %s", argv[0]);
801          return (-1);
802      } else if (child_pid == 0) {
803          closefrom(0);
804          /* redirect stdin, stdout & stderr to /dev/null */
805          (void) open("/dev/null", O_RDONLY); /* stdin */
806          (void) open("/dev/null", O_WRONLY); /* stdout */
807          (void) open("/dev/null", O_WRONLY); /* stderr */
808          (void) execv(path, argv);
809          /*
810           * Since we are in the child, there is no point calling zerror()
811           * since there is nobody waiting to consume it. So exit with a
812           * special code that the parent will recognize and call zerror()
813           * accordingly.
814           */
815          _exit(ZEXIT_EXEC);
816      } else {
817          (void) waitpid(child_pid, &child_status, 0);
818      }
819
820      if (WIFSIGNALED(child_status)) {
821          zerror(zlogp, B_FALSE, "%s unexpectedly terminated due to "
822              "signal %d", path, WTERMSIG(child_status));
823          return (-1);
824      }
825      assert(WIFEXITED(child_status));
826      if (WEXITSTATUS(child_status) == ZEXIT_EXEC) {
827          zerror(zlogp, B_FALSE, "failed to exec %s", path);
828          return (-1);
829      }
830      return (WEXITSTATUS(child_status));
831  }
832
833
834  static int
835  isregfile(const char *path)
836  {
837      struct stat64 st;
838
839      if (stat64(path, &st) == -1)
840          return (-1);
841
842      return (S_ISREG(st.st_mode));
843  }
844
845  static int
846  dofscck(zlog_t *zlogp, const char *fstype, const char *rawdev)
847  {
848      char cmdbuf[MAXPATHLEN];
849      char *argv[5];
850      int status;

```

```

852      /*
853       * We could alternatively have called /usr/sbin/fsck -F <fstype>, but
854       * that would cost us an extra fork/exec without buying us anything.
855       */
856      if (snprintf(cmdbuf, sizeof (cmdbuf), "/usr/lib/fs/%s/fsck", fstype)
857          >= sizeof (cmdbuf)) {
858          zerror(zlogp, B_FALSE, "file-system type %s too long", fstype);
859          return (-1);
860      }
861
862      /*
863       * If it doesn't exist, that's OK: we verified this previously
864       * in zoneadm.
865       */
866      if (isregfile(cmdbuf) == -1)
867          return (0);
868
869      argv[0] = "fsck";
870      argv[1] = "-o";
871      argv[2] = "p";
872      argv[3] = (char *)rawdev;
873      argv[4] = NULL;
874
875      status = forkexec(zlogp, cmdbuf, argv);
876      if (status == 0 || status == -1)
877          return (status);
878      zerror(zlogp, B_FALSE, "fsck of '%s' failed with exit status %d; "
879          "run fsck manually", rawdev, status);
880      return (-1);
881  }
882
883  static int
884  domount(zlog_t *zlogp, const char *fstype, const char *opts,
885          const char *special, const char *directory)
886  {
887      char cmdbuf[MAXPATHLEN];
888      char *argv[6];
889      int status;
890
891      /*
892       * We could alternatively have called /usr/sbin/mount -F <fstype>, but
893       * that would cost us an extra fork/exec without buying us anything.
894       */
895      if (snprintf(cmdbuf, sizeof (cmdbuf), "/usr/lib/fs/%s/mount", fstype)
896          >= sizeof (cmdbuf)) {
897          zerror(zlogp, B_FALSE, "file-system type %s too long", fstype);
898          return (-1);
899      }
900      argv[0] = "mount";
901      if (opts[0] == '\0') {
902          argv[1] = (char *)special;
903          argv[2] = (char *)directory;
904          argv[3] = NULL;
905      } else {
906          argv[1] = "-o";
907          argv[2] = (char *)opts;
908          argv[3] = (char *)special;
909          argv[4] = (char *)directory;
910          argv[5] = NULL;
911      }
912
913      status = forkexec(zlogp, cmdbuf, argv);
914      if (status == 0 || status == -1)
915          return (status);
916      if (opts[0] == '\0')

```

```

917         zerror(zlogp, B_FALSE, "\\%s %s %s\\ "
918               "failed with exit code %d",
919               cmdbuf, special, directory, status);
920     else
921         zerror(zlogp, B_FALSE, "\\%s -o %s %s %s\\ "
922               "failed with exit code %d",
923               cmdbuf, opts, special, directory, status);
924     return (-1);
925 }

927 /*
928 * Check if a given mount point path exists.
929 * If it does, make sure it doesn't contain any symlinks.
930 * Note that if "leaf" is false we're checking an intermediate
931 * component of the mount point path, so it must be a directory.
932 * If "leaf" is true, then we're checking the entire mount point
933 * path, so the mount point itself can be anything aside from a
934 * symbolic link.
935 *
936 * If the path is invalid then a negative value is returned. If the
937 * path exists and is a valid mount point path then 0 is returned.
938 * If the path doesn't exist return a positive value.
939 */
940 static int
941 valid_mount_point(zlog_t *zlogp, const char *path, const boolean_t leaf)
942 {
943     struct stat statbuf;
944     char respath[MAXPATHLEN];
945     int res;

947     if (lstat(path, &statbuf) != 0) {
948         if (errno == ENOENT)
949             return (1);
950         zerror(zlogp, B_TRUE, "can't stat %s", path);
951         return (-1);
952     }
953     if (S_ISLNK(statbuf.st_mode)) {
954         zerror(zlogp, B_FALSE, "%s is a symlink", path);
955         return (-1);
956     }
957     if (!leaf && !S_ISDIR(statbuf.st_mode)) {
958         zerror(zlogp, B_FALSE, "%s is not a directory", path);
959         return (-1);
960     }
961     if ((res = resolvepath(path, respath, sizeof(respath))) == -1) {
962         zerror(zlogp, B_TRUE, "unable to resolve path %s", path);
963         return (-1);
964     }
965     respath[res] = '\0';
966     if (strcmp(path, respath) != 0) {
967         /*
968          * We don't like ".."s, "."s, or "/"s throwing us off
969          */
970         zerror(zlogp, B_FALSE, "%s is not a canonical path", path);
971         return (-1);
972     }
973     return (0);
974 }

976 /*
977 * Validate a mount point path. A valid mount point path is an
978 * absolute path that either doesn't exist, or, if it does exist it
979 * must be an absolute canonical path that doesn't have any symbolic
980 * links in it. The target of a mount point path can be any filesystem
981 * object. (Different filesystems can support different mount points,
982 * for example "lofs" and "mntfs" both support files and directories

```

```

983 * while "ufs" just supports directories.)
984 *
985 * If the path is invalid then a negative value is returned. If the
986 * path exists and is a valid mount point path then 0 is returned.
987 * If the path doesn't exist return a positive value.
988 */
989 int
990 valid_mount_path(zlog_t *zlogp, const char *rootpath, const char *spec,
991                const char *dir, const char *fstype)
992 {
993     char abspath[MAXPATHLEN], *slashp, *slashp_next;
994     int rv;

996     /*
997      * Sanity check the target mount point path.
998      * It must be a non-null string that starts with a '/'.
999      */
1000     if (dir[0] != '/') {
1001         /* Something went wrong. */
1002         zerror(zlogp, B_FALSE, "invalid mount directory, "
1003               "type: \"%s\", special: \"%s\", dir: \"%s\"",
1004               fstype, spec, dir);
1005         return (-1);
1006     }

1008     /*
1009      * Join rootpath and dir. Make sure abspath ends with '/', this
1010      * is added to all paths (even non-directory paths) to allow us
1011      * to detect the end of paths below. If the path already ends
1012      * in a '/', then that's ok too (although we'll fail the
1013      * canonical path check in valid_mount_point()).
1014      */
1015     if (snprintf(abspath, sizeof(abspath),
1016                 "%s%s/", rootpath, dir) >= sizeof(abspath)) {
1017         zerror(zlogp, B_FALSE, "pathname %s%s is too long",
1018               rootpath, dir);
1019         return (-1);
1020     }

1022     /*
1023      * Starting with rootpath, verify the mount path one component
1024      * at a time. Continue until we've evaluated all of abspath.
1025      */
1026     slashp = &abspath[strlen(rootpath)];
1027     assert(*slashp == '/');
1028     do {
1029         slashp_next = strchr(slashp + 1, '/');
1030         *slashp = '\0';
1031         if (slashp_next != NULL) {
1032             /* This is an intermediary mount path component. */
1033             rv = valid_mount_point(zlogp, abspath, B_FALSE);
1034         } else {
1035             /* This is the last component of the mount path. */
1036             rv = valid_mount_point(zlogp, abspath, B_TRUE);
1037         }
1038         if (rv < 0)
1039             return (rv);
1040         *slashp = '/';
1041     } while ((slashp = slashp_next) != NULL);
1042     return (rv);
1043 }

1045 static int
1046 mount_one_dev_device_cb(void *arg, const char *match, const char *name)
1047 {
1048     di_prof_t prof = arg;

```

```

1050     if (name == NULL)
1051         return (di_prof_add_dev(prof, match));
1052     return (di_prof_add_map(prof, match, name));
1053 }

1055 static int
1056 mount_one_dev_symlink_cb(void *arg, const char *source, const char *target)
1057 {
1058     di_prof_t prof = arg;

1060     return (di_prof_add_symlink(prof, source, target));
1061 }

1063 int
1064 vplat_get_iptype(zlog_t *zlogp, zone_iptype_t *iptypep)
1065 {
1066     zone_dochandle_t handle;

1068     if ((handle = zonecfg_init_handle()) == NULL) {
1069         zerror(zlogp, B_TRUE, "getting zone configuration handle");
1070         return (-1);
1071     }
1072     if (zonecfg_get_snapshot_handle(zone_name, handle) != Z_OK) {
1073         zerror(zlogp, B_FALSE, "invalid configuration");
1074         zonecfg_fini_handle(handle);
1075         return (-1);
1076     }
1077     if (zonecfg_get_iptype(handle, iptypep) != Z_OK) {
1078         zerror(zlogp, B_FALSE, "invalid ip-type configuration");
1079         zonecfg_fini_handle(handle);
1080         return (-1);
1081     }
1082     zonecfg_fini_handle(handle);
1083     return (0);
1084 }

1086 /*
1087  * Apply the standard lists of devices/symlinks/mappings and the user-specified
1088  * list of devices (via zonecfg) to the /dev filesystem. The filesystem will
1089  * use these as a profile/filter to determine what exists in /dev.
1090  */
1091 static int
1092 mount_one_dev(zlog_t *zlogp, char *devpath, zone_mnt_t mount_cmd)
1093 {
1094     char                brand[MAXNAMELEN];
1095     zone_dochandle_t    handle = NULL;
1096     brand_handle_t      bh = NULL;
1097     struct zone_devtab  ztab;
1098     di_prof_t           prof = NULL;
1099     int                 err;
1100     int                 retval = -1;
1101     zone_iptype_t       iptype;
1102     const char          *curr_iptype;

1104     if (di_prof_init(devpath, &prof)) {
1105         zerror(zlogp, B_TRUE, "failed to initialize profile");
1106         goto cleanup;
1107     }

1109     /*
1110     * Get a handle to the brand info for this zone.
1111     * If we are mounting the zone, then we must always use the default
1112     * brand device mounts.
1113     */
1114     if (ALT_MOUNT(mount_cmd)) {

```

```

1115         (void) strncpy(brand, default_brand, sizeof (brand));
1116     } else {
1117         (void) strncpy(brand, brand_name, sizeof (brand));
1118     }

1120     if ((bh = brand_open(brand)) == NULL) {
1121         zerror(zlogp, B_FALSE, "unable to determine zone brand");
1122         goto cleanup;
1123     }

1125     if (vplat_get_iptype(zlogp, &iptype) < 0) {
1126         zerror(zlogp, B_TRUE, "unable to determine ip-type");
1127         goto cleanup;
1128     }
1129     switch (iptype) {
1130     case ZS_SHARED:
1131         curr_iptype = "shared";
1132         break;
1133     case ZS_EXCLUSIVE:
1134         curr_iptype = "exclusive";
1135         break;
1136     }

1138     if (brand_platform_iter_devices(bh, zone_name,
1139         mount_one_dev_device_cb, prof, curr_iptype) != 0) {
1140         zerror(zlogp, B_TRUE, "failed to add standard device");
1141         goto cleanup;
1142     }

1144     if (brand_platform_iter_link(bh,
1145         mount_one_dev_symlink_cb, prof) != 0) {
1146         zerror(zlogp, B_TRUE, "failed to add standard symlink");
1147         goto cleanup;
1148     }

1150     /* Add user-specified devices and directories */
1151     if ((handle = zonecfg_init_handle()) == NULL) {
1152         zerror(zlogp, B_FALSE, "can't initialize zone handle");
1153         goto cleanup;
1154     }
1155     if (err = zonecfg_get_handle(zone_name, handle)) {
1156         zerror(zlogp, B_FALSE, "can't get handle for zone "
1157             "%s: %s", zone_name, zonecfg_strerror(err));
1158         goto cleanup;
1159     }
1160     if (err = zonecfg_setdevent(handle)) {
1161         zerror(zlogp, B_FALSE, "%s: %s", zone_name,
1162             zonecfg_strerror(err));
1163         goto cleanup;
1164     }
1165     while (zonecfg_getdevent(handle, &ztab) == Z_OK) {
1166         if (di_prof_add_dev(prof, ztab.zone_dev_match)) {
1167             zerror(zlogp, B_TRUE, "failed to add "
1168                 "user-specified device");
1169             goto cleanup;
1170         }
1171     }
1172     (void) zonecfg_enddevent(handle);

1174     /* Send profile to kernel */
1175     if (di_prof_commit(prof)) {
1176         zerror(zlogp, B_TRUE, "failed to commit profile");
1177         goto cleanup;
1178     }

1180     retval = 0;

```

```

1182 cleanup:
1183     if (bh != NULL)
1184         brand_close(bh);
1185     if (handle != NULL)
1186         zonecfg_fini_handle(handle);
1187     if (prof)
1188         di_prof_fini(prof);
1189     return (retval);
1190 }

1192 static int
1193 mount_one(zlogp_t *zlogp, struct zone_fstab *fsptr, const char *rootpath,
1194          zone_mnt_t mount_cmd)
1195 {
1196     char path[MAXPATHLEN];
1197     char optstr[MAX_MNTOPT_STR];
1198     zone_fsopt_t *optptr;
1199     int rv;

1201     if ((rv = valid_mount_path(zlogp, rootpath, fsptr->zone_fs_special,
1202                               fsptr->zone_fs_dir, fsptr->zone_fs_type)) < 0) {
1203         zerror(zlogp, B_FALSE, "%s%s is not a valid mount point",
1204              rootpath, fsptr->zone_fs_dir);
1205         return (-1);
1206     } else if (rv > 0) {
1207         /* The mount point path doesn't exist, create it now. */
1208         if (make_one_dir(zlogp, rootpath, fsptr->zone_fs_dir,
1209                        DEFAULT_DIR_MODE, DEFAULT_DIR_USER,
1210                        DEFAULT_DIR_GROUP) != 0) {
1211             zerror(zlogp, B_FALSE, "failed to create mount point");
1212             return (-1);
1213         }

1215         /*
1216          * Now this might seem weird, but we need to invoke
1217          * valid_mount_path() again. Why? Because it checks
1218          * to make sure that the mount point path is canonical,
1219          * which it can only do if the path exists, so now that
1220          * we've created the path we have to verify it again.
1221          */
1222         if ((rv = valid_mount_path(zlogp, rootpath,
1223                                   fsptr->zone_fs_special, fsptr->zone_fs_dir,
1224                                   fsptr->zone_fs_type)) < 0) {
1225             zerror(zlogp, B_FALSE,
1226                  "%s%s is not a valid mount point",
1227                   rootpath, fsptr->zone_fs_dir);
1228             return (-1);
1229         }
1230     }

1232     (void) sprintf(path, sizeof (path), "%s%s", rootpath,
1233                  fsptr->zone_fs_dir);

1235     /*
1236      * In general the strategy here is to do just as much verification as
1237      * necessary to avoid crashing or otherwise doing something bad; if the
1238      * administrator initiated the operation via zoneadm(lm), he'll get
1239      * auto-verification which will let him know what's wrong. If he
1240      * modifies the zone configuration of a running zone and doesn't attempt
1241      * to verify that it's OK we won't crash but won't bother trying to be
1242      * too helpful either. zoneadm verify is only a couple keystrokes away.
1243      */
1244     if (!zonecfg_valid_fs_type(fsptr->zone_fs_type)) {
1245         zerror(zlogp, B_FALSE, "cannot mount %s on %s: "
1246              "invalid file-system type %s", fsptr->zone_fs_special,

```

```

1247         fsptr->zone_fs_dir, fsptr->zone_fs_type);
1248         return (-1);
1249     }

1251     /*
1252      * If we're looking at an alternate root environment, then construct
1253      * read-only loopback mounts as necessary. Note that any special
1254      * paths for lofs zone mounts in an alternate root must have
1255      * already been pre-pended with any alternate root path by the
1256      * time we get here.
1257      */
1258     if (zonecfg_in_alt_root()) {
1259         struct stat64 st;

1261         if (stat64(fsptr->zone_fs_special, &st) != -1 &&
1262             S_ISBLK(st.st_mode)) {
1263             /*
1264              * If we're going to mount a block device we need
1265              * to check if that device is already mounted
1266              * somewhere else, and if so, do a lofs mount
1267              * of the device instead of a direct mount
1268              */
1269             if (check_lofs_needed(zlogp, fsptr) == -1)
1270                 return (-1);
1271         } else if (strcmp(fsptr->zone_fs_type, MNTTYPE_LOFS) == 0) {
1272             /*
1273              * For lofs mounts, the special node is inside the
1274              * alternate root. We need lofs resolution for
1275              * this case in order to get at the underlying
1276              * read-write path.
1277              */
1278             resolve_lofs(zlogp, fsptr->zone_fs_special,
1279                          sizeof (fsptr->zone_fs_special));
1280         }
1281     }

1283     /*
1284      * Run 'fsck -m' if there's a device to fsck.
1285      */
1286     if (fsptr->zone_fs_raw[0] != '\0' &&
1287         dofscck(zlogp, fsptr->zone_fs_type, fsptr->zone_fs_raw) != 0) {
1288         return (-1);
1289     } else if (isregfile(fsptr->zone_fs_special) == 1 &&
1290         dofscck(zlogp, fsptr->zone_fs_type, fsptr->zone_fs_special) != 0) {
1291         return (-1);
1292     }

1294     /*
1295      * Build up mount option string.
1296      */
1297     optstr[0] = '\0';
1298     if (fsptr->zone_fs_options != NULL) {
1299         (void) strcpy(optstr, fsptr->zone_fs_options->zone_fsopt_opt,
1300                      sizeof (optstr));
1301         for (optptr = fsptr->zone_fs_options->zone_fsopt_next;
1302              optptr != NULL; optptr = optptr->zone_fsopt_next) {
1303             (void) strcat(optstr, ",", sizeof (optstr));
1304             (void) strcat(optstr, optptr->zone_fsopt_opt,
1305                          sizeof (optstr));
1306         }
1307     }

1309     if ((rv = domount(zlogp, fsptr->zone_fs_type, optstr,
1310                      fsptr->zone_fs_special, path)) != 0)
1311         return (rv);

```

```

1313  /*
1314  * The mount succeeded.  If this was not a mount of /dev then
1315  * we're done.
1316  */
1317  if (strcmp(fsptr->zone_fs_type, MNTTYPE_DEV) != 0)
1318      return (0);

1320  /*
1321  * We just mounted an instance of a /dev filesystem, so now we
1322  * need to configure it.
1323  */
1324  return (mount_one_dev(zlogp, path, mount_cmd));
1325 }

1327 static void
1328 free_fs_data(struct zone_fstab *fsarray, uint_t nelelem)
1329 {
1330     uint_t i;

1332     if (fsarray == NULL)
1333         return;
1334     for (i = 0; i < nelelem; i++)
1335         zonecfg_free_fs_option_list(fsarray[i].zone_fs_options);
1336     free(fsarray);
1337 }

1339 /*
1340 * This function initiates the creation of a small Solaris Environment for
1341 * scratch zone. The Environment creation process is split up into two
1342 * functions(build_mounted_pre_var() and build_mounted_post_var()). It
1343 * is done this way because:
1344 * We need to have both /etc and /var in the root of the scratchzone.
1345 * We loopback mount zone's own /etc and /var into the root of the
1346 * scratch zone. Unlike /etc, /var can be a separate filesystem. So we
1347 * need to delay the mount of /var till the zone's root gets populated.
1348 * So mounting of localdirs[](/etc and /var) have been moved to the
1349 * build_mounted_post_var() which gets called only after the zone
1350 * specific filesystems are mounted.
1351 *
1352 * Note that the scratch zone we set up for updating the zone (Z_MNT_UPDATE)
1353 * does not loopback mount the zone's own /etc and /var into the root of the
1354 * scratch zone.
1355 */
1356 static boolean_t
1357 build_mounted_pre_var(zlog_t *zlogp, char *rootpath,
1358 size_t rootlen, const char *zonepath, char *luroot, size_t lurootlen)
1359 {
1360     char tmp[MAXPATHLEN], fromdir[MAXPATHLEN];
1361     const char **cpp;
1362     static const char *makedirs[] = {
1363         "/system", "/system/contract", "/system/object", "/proc",
1364         "/dev", "/tmp", "/a", NULL
1365     };
1366     char *altstr;
1367     FILE *fp;
1368     uuid_t uuid;

1370     resolve_lofs(zlogp, rootpath, rootlen);
1371     (void) snprintf(luroot, lurootlen, "%s/lu", zonepath);
1372     resolve_lofs(zlogp, luroot, lurootlen);
1373     (void) snprintf(tmp, sizeof (tmp), "%s/bin", luroot);
1374     (void) symlink("./usr/bin", tmp);

1376     /*
1377     * These are mostly special mount points; not handled here. (See
1378     * zone_mount_early.)

```

```

1379     /*
1380     for (cpp = makedirs; *cpp != NULL; cpp++) {
1381         (void) snprintf(tmp, sizeof (tmp), "%s%s", luroot, *cpp);
1382         if (mkdir(tmp, 0755) != 0) {
1383             zerror(zlogp, B_TRUE, "cannot create %s", tmp);
1384             return (B_FALSE);
1385         }
1386     }
1387     /*
1388     * This is here to support lucopy.  If there's an instance of this same
1389     * zone on the current running system, then we mount its root up as
1390     * read-only inside the scratch zone.
1391     */
1392     (void) zonecfg_get_uuid(zone_name, uuid);
1393     altstr = strdup(zonecfg_get_root());
1394     if (altstr == NULL) {
1395         zerror(zlogp, B_TRUE, "memory allocation failed");
1396         return (B_FALSE);
1397     }
1398     zonecfg_set_root("");
1399     (void) strlcpy(tmp, zone_name, sizeof (tmp));
1400     (void) zonecfg_get_name_by_uuid(uuid, tmp, sizeof (tmp));
1401     if (zone_get_rootpath(tmp, fromdir, sizeof (fromdir)) == Z_OK &&
1402         strcmp(fromdir, rootpath) != 0) {
1403         (void) snprintf(tmp, sizeof (tmp), "%s/b", luroot);
1404         if (mkdir(tmp, 0755) != 0) {
1405             zerror(zlogp, B_TRUE, "cannot create %s", tmp);
1406             return (B_FALSE);
1407         }
1408         if (domount(zlogp, MNTTYPE_LOFS, RESOURCE_DEFAULT_OPTS, fromdir,
1409             tmp) != 0) {
1410             zerror(zlogp, B_TRUE, "cannot mount %s on %s", tmp,
1411                 fromdir);
1412             return (B_FALSE);
1413         }
1414     }
1415     zonecfg_set_root(altstr);
1416     free(altstr);

1418     if ((fp = zonecfg_open_scratch(luroot, B_TRUE)) == NULL) {
1419         zerror(zlogp, B_TRUE, "cannot open zone mapfile");
1420         return (B_FALSE);
1421     }
1422     (void) ftruncate(fileno(fp), 0);
1423     if (zonecfg_add_scratch(fp, zone_name, kernzone, "/") == -1) {
1424         zerror(zlogp, B_TRUE, "cannot add zone mapfile entry");
1425     }
1426     zonecfg_close_scratch(fp);
1427     (void) snprintf(tmp, sizeof (tmp), "%s/a", luroot);
1428     if (domount(zlogp, MNTTYPE_LOFS, "", rootpath, tmp) != 0)
1429         return (B_FALSE);
1430     (void) strlcpy(rootpath, tmp, rootlen);
1431     return (B_TRUE);
1432 }

1435 static boolean_t
1436 build_mounted_post_var(zlog_t *zlogp, zone_mnt_t mount_cmd, char *rootpath,
1437     const char *luroot)
1438 {
1439     char tmp[MAXPATHLEN], fromdir[MAXPATHLEN];
1440     const char **cpp;
1441     const char **loopdirs;
1442     const char **tmpdirs;
1443     static const char *localdirs[] = {
1444         "/etc", "/var", NULL

```

```

1445 };
1446 static const char *scr_loopdirs[] = {
1447     "/etc/lib", "/etc/fs", "/lib", "/sbin", "/platform",
1448     "/usr", NULL
1449 };
1450 static const char *upd_loopdirs[] = {
1451     "/etc", "/kernel", "/lib", "/opt", "/platform", "/sbin",
1452     "/usr", "/var", NULL
1453 };
1454 static const char *scr_tmpdirs[] = {
1455     "/tmp", "/var/run", NULL
1456 };
1457 static const char *upd_tmpdirs[] = {
1458     "/tmp", "/var/run", "/var/tmp", NULL
1459 };
1460 struct stat st;
1461
1462 if (mount_cmd == Z_MNT_SCRATCH) {
1463     /*
1464      * These are mounted read-write from the zone undergoing
1465      * upgrade. We must be careful not to 'leak' things from the
1466      * main system into the zone, and this accomplishes that goal.
1467      */
1468     for (cpp = localdirs; *cpp != NULL; cpp++) {
1469         (void) snprintf(tmp, sizeof (tmp), "%s%s", luroot,
1470             *cpp);
1471         (void) snprintf(fromdir, sizeof (fromdir), "%s%s",
1472             rootpath, *cpp);
1473         if (mkdir(tmp, 0755) != 0) {
1474             zerror(zlogp, B_TRUE, "cannot create %s", tmp);
1475             return (B_FALSE);
1476         }
1477         if (domount(zlogp, MNTTYPE_LOFS, "", fromdir, tmp)
1478             != 0) {
1479             zerror(zlogp, B_TRUE, "cannot mount %s on %s",
1480                 tmp, *cpp);
1481             return (B_FALSE);
1482         }
1483     }
1484 }
1485
1486 if (mount_cmd == Z_MNT_UPDATE)
1487     loopdirs = upd_loopdirs;
1488 else
1489     loopdirs = scr_loopdirs;
1490
1491 /*
1492  * These are things mounted read-only from the running system because
1493  * they contain binaries that must match system.
1494  */
1495 for (cpp = loopdirs; *cpp != NULL; cpp++) {
1496     (void) snprintf(tmp, sizeof (tmp), "%s%s", luroot, *cpp);
1497     if (mkdir(tmp, 0755) != 0) {
1498         if (errno != EEXIST) {
1499             zerror(zlogp, B_TRUE, "cannot create %s", tmp);
1500             return (B_FALSE);
1501         }
1502         if (lstat(tmp, &st) != 0) {
1503             zerror(zlogp, B_TRUE, "cannot stat %s", tmp);
1504             return (B_FALSE);
1505         }
1506     }
1507     /*
1508      * Ignore any non-directories encountered. These are
1509      * things that have been converted into symlinks
1510      * (/etc/fs and /etc/lib) and no longer need a lofs
1511      * fixup.

```

```

1511     */
1512     if (!S_ISDIR(st.st_mode))
1513         continue;
1514 }
1515 if (domount(zlogp, MNTTYPE_LOFS, RESOURCE_DEFAULT_OPTS, *cpp,
1516     tmp) != 0) {
1517     zerror(zlogp, B_TRUE, "cannot mount %s on %s", tmp,
1518         *cpp);
1519     return (B_FALSE);
1520 }
1521 }
1522
1523 if (mount_cmd == Z_MNT_UPDATE)
1524     tmpdirs = upd_tmpdirs;
1525 else
1526     tmpdirs = scr_tmpdirs;
1527
1528 /*
1529  * These are things with tmpfs mounted inside.
1530  */
1531 for (cpp = tmpdirs; *cpp != NULL; cpp++) {
1532     (void) snprintf(tmp, sizeof (tmp), "%s%s", luroot, *cpp);
1533     if (mount_cmd == Z_MNT_SCRATCH && mkdir(tmp, 0755) != 0 &&
1534         errno != EEXIST) {
1535         zerror(zlogp, B_TRUE, "cannot create %s", tmp);
1536         return (B_FALSE);
1537     }
1538 }
1539
1540 /*
1541  * We could set the mode for /tmp when we do the mkdir but
1542  * since that can be modified by the umask we will just set
1543  * the correct mode for /tmp now.
1544  */
1545 if (strcmp(*cpp, "/tmp") == 0 && chmod(tmp, 01777) != 0) {
1546     zerror(zlogp, B_TRUE, "cannot chmod %s", tmp);
1547     return (B_FALSE);
1548 }
1549
1550 if (domount(zlogp, MNTTYPE_TMPFS, "", "swap", tmp) != 0) {
1551     zerror(zlogp, B_TRUE, "cannot mount swap on %s", *cpp);
1552     return (B_FALSE);
1553 }
1554 }
1555 }
1556
1557 typedef struct plat_gmount_cb_data {
1558     zlog_t          *pgcd_zlogp;
1559     struct zone_fstab *pgcd_fs_tab;
1560     int             *pgcd_num_fs;
1561 } plat_gmount_cb_data_t;
1562
1563 /*
1564  * plat_gmount_cb() is a callback function invoked by libbrand to iterate
1565  * through all global brand platform mounts.
1566  */
1567 int
1568 plat_gmount_cb(void *data, const char *spec, const char *dir,
1569     const char *fstype, const char *opt)
1570 {
1571     plat_gmount_cb_data_t *cp = data;
1572     zlog_t                 *zlogp = cp->pgcd_zlogp;
1573     struct zone_fstab      *fs_ptr = *cp->pgcd_fs_tab;
1574     int                     num_fs = *cp->pgcd_num_fs;
1575     struct zone_fstab      *fsp, *tmp_ptr;

```

```

1577     num_fs++;
1578     if ((tmp_ptr = realloc(fs_ptr, num_fs * sizeof (*tmp_ptr))) == NULL) {
1579         zerror(zlogp, B_TRUE, "memory allocation failed");
1580         return (-1);
1581     }
1583     fs_ptr = tmp_ptr;
1584     fsp = &fs_ptr[num_fs - 1];
1586     /* update the callback struct passed in */
1587     *cp->pgcd_fs_tab = fs_ptr;
1588     *cp->pgcd_num_fs = num_fs;
1590     fsp->zone_fs_raw[0] = '\0';
1591     (void) strcpy(fsp->zone_fs_special, spec,
1592                 sizeof (fsp->zone_fs_special));
1593     (void) strcpy(fsp->zone_fs_dir, dir, sizeof (fsp->zone_fs_dir));
1594     (void) strcpy(fsp->zone_fs_type, fstype, sizeof (fsp->zone_fs_type));
1595     fsp->zone_fs_options = NULL;
1596     if ((opt != NULL) &&
1597         (zonecfg_add_fs_option(fsp, (char *)opt) != Z_OK)) {
1598         zerror(zlogp, B_FALSE, "error adding property");
1599         return (-1);
1600     }
1602     return (0);
1603 }
1605 static int
1606 mount_filesystems(zone_dochandle_t handle, zlog_t *zlogp,
1607                  struct zone_fstab **fs_tabp, int *num_fsp, zone_mnt_t mount_cmd)
1608 {
1609     struct zone_fstab *tmp_ptr, *fs_ptr, *fsp, fstab;
1610     int num_fs;
1612     num_fs = *num_fsp;
1613     fs_ptr = *fs_tabp;
1615     if (zonecfg_setfsent(handle) != Z_OK) {
1616         zerror(zlogp, B_FALSE, "invalid configuration");
1617         return (-1);
1618     }
1619     while (zonecfg_getfsent(handle, &fstab) == Z_OK) {
1620         /*
1621          * ZFS filesystems will not be accessible under an alternate
1622          * root, since the pool will not be known. Ignore them in this
1623          * case.
1624          */
1625         if (ALT_MOUNT(mount_cmd) &&
1626             strcmp(fstab.zone_fs_type, MNTTYPE_ZFS) == 0)
1627             continue;
1629         num_fs++;
1630         if ((tmp_ptr = realloc(fs_ptr,
1631                               num_fs * sizeof (*tmp_ptr))) == NULL) {
1632             zerror(zlogp, B_TRUE, "memory allocation failed");
1633             (void) zonecfg_endfsent(handle);
1634             return (-1);
1635         }
1636         /* update the pointers passed in */
1637         *fs_tabp = tmp_ptr;
1638         *num_fsp = num_fs;
1640         fs_ptr = tmp_ptr;
1641         fsp = &fs_ptr[num_fs - 1];
1642         (void) strcpy(fsp->zone_fs_dir,

```

```

1643         fstab.zone_fs_dir, sizeof (fsp->zone_fs_dir));
1644         (void) strcpy(fsp->zone_fs_raw, fstab.zone_fs_raw,
1645                     sizeof (fsp->zone_fs_raw));
1646         (void) strcpy(fsp->zone_fs_type, fstab.zone_fs_type,
1647                     sizeof (fsp->zone_fs_type));
1648         fsp->zone_fs_options = fstab.zone_fs_options;
1650         /*
1651          * For all lofs mounts, make sure that the 'special'
1652          * entry points inside the alternate root. The
1653          * source path for a lofs mount in a given zone needs
1654          * to be relative to the root of the boot environment
1655          * that contains the zone. Note that we don't do this
1656          * for non-lofs mounts since they will have a device
1657          * as a backing store and device paths must always be
1658          * specified relative to the current boot environment.
1659          */
1660         fsp->zone_fs_special[0] = '\0';
1661         if (strcmp(fsp->zone_fs_type, MNTTYPE_LOFS) == 0) {
1662             (void) strcat(fsp->zone_fs_special, zonecfg_get_root(),
1663                          sizeof (fsp->zone_fs_special));
1664         }
1665         (void) strcat(fsp->zone_fs_special, fstab.zone_fs_special,
1666                     sizeof (fsp->zone_fs_special));
1667     }
1668     (void) zonecfg_endfsent(handle);
1669     return (0);
1670 }
1672 static int
1673 mount_filesystems(zlog_t *zlogp, zone_mnt_t mount_cmd)
1674 {
1675     char rootpath[MAXPATHLEN];
1676     char zonepath[MAXPATHLEN];
1677     char brand[MAXNAMELEN];
1678     char lurroot[MAXPATHLEN];
1679     int i, num_fs = 0;
1680     struct zone_fstab *fs_ptr = NULL;
1681     zone_dochandle_t handle = NULL;
1682     zone_state_t zstate;
1683     brand_handle_t bh;
1684     plat_gmount_cb_data_t cb;
1686     if (zone_get_state(zone_name, &zstate) != Z_OK ||
1687         (zstate != ZONE_STATE_READY && zstate != ZONE_STATE_MOUNTED)) {
1688         zerror(zlogp, B_FALSE,
1689              "zone must be in '%s' or '%s' state to mount file-systems",
1690              zone_state_str(ZONE_STATE_READY),
1691              zone_state_str(ZONE_STATE_MOUNTED));
1692         goto bad;
1693     }
1695     if (zone_get_zonepath(zone_name, zonepath, sizeof (zonepath)) != Z_OK) {
1696         zerror(zlogp, B_TRUE, "unable to determine zone path");
1697         goto bad;
1698     }
1700     if (zone_get_rootpath(zone_name, rootpath, sizeof (rootpath)) != Z_OK) {
1701         zerror(zlogp, B_TRUE, "unable to determine zone root");
1702         goto bad;
1703     }
1705     if ((handle = zonecfg_init_handle()) == NULL) {
1706         zerror(zlogp, B_TRUE, "getting zone configuration handle");
1707         goto bad;
1708     }

```

```

1709     if (zonecfg_get_snapshot_handle(zone_name, handle) != Z_OK ||
1710         zonecfg_setfsent(handle) != Z_OK) {
1711         zerror(zlogp, B_FALSE, "invalid configuration");
1712         goto bad;
1713     }
1714
1715     /*
1716     * If we are mounting the zone, then we must always use the default
1717     * brand global mounts.
1718     */
1719     if (ALT_MOUNT(mount_cmd)) {
1720         (void) strcpy(brand, default_brand, sizeof (brand));
1721     } else {
1722         (void) strcpy(brand, brand_name, sizeof (brand));
1723     }
1724
1725     /* Get a handle to the brand info for this zone */
1726     if ((bh = brand_open(brand)) == NULL) {
1727         zerror(zlogp, B_FALSE, "unable to determine zone brand");
1728         zonecfg_fini_handle(handle);
1729         return (-1);
1730     }
1731
1732     /*
1733     * Get the list of global filesystems to mount from the brand
1734     * configuration.
1735     */
1736     cb.pgcd_zlogp = zlogp;
1737     cb.pgcd_fs_tab = &fs_ptr;
1738     cb.pgcd_num_fs = &num_fs;
1739     if (brand_platform_iter_gmounts(bh, zonepath,
1740         plat_gmount_cb, &cb) != 0) {
1741         zerror(zlogp, B_FALSE, "unable to mount filesystems");
1742         brand_close(bh);
1743         zonecfg_fini_handle(handle);
1744         return (-1);
1745     }
1746     brand_close(bh);
1747
1748     /*
1749     * Iterate through the rest of the filesystems. Sort them all,
1750     * then mount them in sorted order. This is to make sure the
1751     * higher level directories (e.g., /usr) get mounted before
1752     * any beneath them (e.g., /usr/local).
1753     */
1754     if (mount_filesystems_fsent(handle, zlogp, &fs_ptr, &num_fs,
1755         mount_cmd) != 0)
1756         goto bad;
1757
1758     zonecfg_fini_handle(handle);
1759     handle = NULL;
1760
1761     /*
1762     * Normally when we mount a zone all the zone filesystems
1763     * get mounted relative to rootpath, which is usually
1764     * <zonepath>/root. But when mounting a zone for administration
1765     * purposes via the zone "mount" state, build_mounted_pre_var()
1766     * updates rootpath to be <zonepath>/lu/a so we'll mount all
1767     * the zones filesystems there instead.
1768     *
1769     * build_mounted_pre_var() and build_mounted_post_var() will
1770     * also do some extra work to create directories and lofs mount
1771     * a bunch of global zone file system paths into <zonepath>/lu.
1772     *
1773     * This allows us to be able to enter the zone (now rooted at
1774     * <zonepath>/lu) and run the upgrade/patch tools that are in the

```

```

1775     * global zone and have them upgrade the to-be-modified zone's
1776     * files mounted on /a. (Which mirrors the existing standard
1777     * upgrade environment.)
1778     */
1779     * There is of course one catch. When doing the upgrade
1780     * we need <zoneroot>/lu/dev to be the /dev filesystem
1781     * for the zone and we don't want to have any /dev filesystem
1782     * mounted at <zoneroot>/lu/a/dev. Since /dev is specified
1783     * as a normal zone filesystem by default we'll try to mount
1784     * it at <zoneroot>/lu/a/dev, so we have to detect this
1785     * case and instead mount it at <zoneroot>/lu/dev.
1786     */
1787     * All this work is done in three phases:
1788     * 1) Create and populate lu directory (build_mounted_pre_var()).
1789     * 2) Mount the required filesystems as per the zone configuration.
1790     * 3) Set up the rest of the scratch zone environment
1791     *     (build_mounted_post_var()).
1792     */
1793     if (ALT_MOUNT(mount_cmd) && !build_mounted_pre_var(zlogp,
1794         rootpath, sizeof (rootpath), zonepath, luroot, sizeof (luroot)))
1795         goto bad;
1796
1797     qsort(fs_ptr, num_fs, sizeof (*fs_ptr), fs_compare);
1798
1799     for (i = 0; i < num_fs; i++) {
1800         if (ALT_MOUNT(mount_cmd) &&
1801             strcmp(fs_ptr[i].zone_fs_dir, "/dev") == 0) {
1802             size_t slen = strlen(rootpath) - 2;
1803
1804             /*
1805             * By default we'll try to mount /dev as /a/dev
1806             * but /dev is special and always goes at the top
1807             * so strip the trailing '/a' from the rootpath.
1808             */
1809             assert(strcmp(&rootpath[slen], "/a") == 0);
1810             rootpath[slen] = '\0';
1811             if (mount_one(zlogp, &fs_ptr[i], rootpath, mount_cmd)
1812                 != 0)
1813                 goto bad;
1814             rootpath[slen] = '/';
1815             continue;
1816         }
1817         if (mount_one(zlogp, &fs_ptr[i], rootpath, mount_cmd) != 0)
1818             goto bad;
1819     }
1820     if (ALT_MOUNT(mount_cmd) &&
1821         !build_mounted_post_var(zlogp, mount_cmd, rootpath, luroot))
1822         goto bad;
1823
1824     /*
1825     * For Trusted Extensions cross-mount each lower level /export/home
1826     */
1827     if (mount_cmd == Z_MNT_BOOT &&
1828         tsol_mounts(zlogp, zone_name, rootpath) != 0)
1829         goto bad;
1830
1831     free_fs_data(fs_ptr, num_fs);
1832
1833     /*
1834     * Everything looks fine.
1835     */
1836     return (0);
1837
1838 bad:
1839     if (handle != NULL)
1840         zonecfg_fini_handle(handle);

```



```

1841     free_fs_data(fs_ptr, num_fs);
1842     return (-1);
1843 }

1845 /* caller makes sure neither parameter is NULL */
1846 static int
1847 addr2netmask(char *prefixstr, int maxprefixlen, uchar_t *maskstr)
1848 {
1849     int prefixlen;

1851     prefixlen = atoi(prefixstr);
1852     if (prefixlen < 0 || prefixlen > maxprefixlen)
1853         return (1);
1854     while (prefixlen > 0) {
1855         if (prefixlen >= 8) {
1856             *maskstr++ = 0xFF;
1857             prefixlen -= 8;
1858             continue;
1859         }
1860         *maskstr |= 1 << (8 - prefixlen);
1861         prefixlen--;
1862     }
1863     return (0);
1864 }

1866 /*
1867 * Tear down all interfaces belonging to the given zone. This should
1868 * be called with the zone in a state other than "running", so that
1869 * interfaces can't be assigned to the zone after this returns.
1870 *
1871 * If anything goes wrong, log an error message and return an error.
1872 */
1873 static int
1874 unconfigure_shared_network_interfaces(zlog_t *zlogp, zoneid_t zone_id)
1875 {
1876     struct lifnum lifn;
1877     struct lifconf lifc;
1878     struct lifreq *lifrp, lifrl;
1879     int64_t lifc_flags = LIFC_NOXMIT | LIFC_ALLZONES;
1880     int num_ifs, s, i, ret_code = 0;
1881     uint_t bufsize;
1882     char *buf = NULL;

1884     if ((s = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
1885         zerror(zlogp, B_TRUE, "could not get socket");
1886         ret_code = -1;
1887         goto bad;
1888     }
1889     lifn.lifn_family = AF_UNSPEC;
1890     lifn.lifn_flags = (int)lifc_flags;
1891     if (ioctl(s, SIOCGLIFNUM, (char *)&lifn) < 0) {
1892         zerror(zlogp, B_TRUE,
1893             "could not determine number of network interfaces");
1894         ret_code = -1;
1895         goto bad;
1896     }
1897     num_ifs = lifn.lifn_count;
1898     bufsize = num_ifs * sizeof (struct lifreq);
1899     if ((buf = malloc(bufsize)) == NULL) {
1900         zerror(zlogp, B_TRUE, "memory allocation failed");
1901         ret_code = -1;
1902         goto bad;
1903     }
1904     lifc.lifc_family = AF_UNSPEC;
1905     lifc.lifc_flags = (int)lifc_flags;
1906     lifc.lifc_len = bufsize;

```

```

1907     lifc.lifc_buf = buf;
1908     if (ioctl(s, SIOCGLIFCONF, (char *)&lifc) < 0) {
1909         zerror(zlogp, B_TRUE, "could not get configured network "
1910             "interfaces");
1911         ret_code = -1;
1912         goto bad;
1913     }
1914     lifrp = lifc.lifc_req;
1915     for (i = lifc.lifc_len / sizeof (struct lifreq); i > 0; i--, lifrp++) {
1916         (void) close(s);
1917         if ((s = socket(lifrp->lifr_addr.ss_family, SOCK_DGRAM, 0)) <
1918             0) {
1919             zerror(zlogp, B_TRUE, "%s: could not get socket",
1920                 lifrl.lifr_name);
1921             ret_code = -1;
1922             continue;
1923         }
1924         (void) memset(&lifrl, 0, sizeof (lifrl));
1925         (void) strncpy(lifrl.lifr_name, lifrp->lifr_name,
1926             sizeof (lifrl.lifr_name));
1927         if (ioctl(s, SIOCGLIFZONE, (caddr_t)&lifrl) < 0) {
1928             if (errno == ENXIO)
1929                 /*
1930                  * Interface may have been removed by admin or
1931                  * another zone halting.
1932                  */
1933                 continue;
1934             zerror(zlogp, B_TRUE,
1935                 "%s: could not determine the zone to which this "
1936                 "network interface is bound", lifrl.lifr_name);
1937             ret_code = -1;
1938             continue;
1939         }
1940         if (lifrl.lifr_zoneid == zone_id) {
1941             if (ioctl(s, SIOCGLIFREMOVEIF, (caddr_t)&lifrl) < 0) {
1942                 zerror(zlogp, B_TRUE,
1943                     "%s: could not remove network interface",
1944                         lifrl.lifr_name);
1945                 ret_code = -1;
1946                 continue;
1947             }
1948         }
1949     }
1950 bad:
1951     if (s > 0)
1952         (void) close(s);
1953     if (buf)
1954         free(buf);
1955     return (ret_code);
1956 }

1958 static union sockunion {
1959     struct sockaddr sa;
1960     struct sockaddr_in sin;
1961     struct sockaddr_dl sdl;
1962     struct sockaddr_in6 sin6;
1963 } so_dst, so_ifp;

1965 static struct {
1966     struct rt_msghdr hdr;
1967     char space[512];
1968 } rtmsg;

1970 static int
1971 salen(struct sockaddr *sa)
1972 {

```

```

1973     switch (sa->sa_family) {
1974     case AF_INET:
1975         return (sizeof (struct sockaddr_in));
1976     case AF_LINK:
1977         return (sizeof (struct sockaddr_dl));
1978     case AF_INET6:
1979         return (sizeof (struct sockaddr_in6));
1980     default:
1981         return (sizeof (struct sockaddr));
1982     }
1983 }

1985 #define ROUNDUP_LONG(a) \
1986     ((a) > 0 ? (1 + (((a) - 1) | (sizeof (long) - 1))) : sizeof (long))

1988 /*
1989  * Look up which zone is using a given IP address.  The address in question
1990  * is expected to have been stuffed into the structure to which lifr points
1991  * via a previous SIOCGLIFADDR ioctl().
1992  *
1993  * This is done using black router socket magic.
1994  *
1995  * Return the name of the zone on success or NULL on failure.
1996  *
1997  * This is a lot of code for a simple task; a new ioctl request to take care
1998  * of this might be a useful RFE.
1999  */

2001 static char *
2002 who_is_using(zlog_t *zlogp, struct lifreq *lifr)
2003 {
2004     static char answer[ZONENAME_MAX];
2005     pid_t pid;
2006     int s, rlen, l, i;
2007     char *cp = rtmsg.space;
2008     struct sockaddr_dl *ifp = NULL;
2009     struct sockaddr *sa;
2010     char save_if_name[LIFNAMSIZ];

2012     answer[0] = '\0';

2014     pid = getpid();
2015     if ((s = socket(PF_ROUTE, SOCK_RAW, 0)) < 0) {
2016         zerror(zlogp, B_TRUE, "could not get routing socket");
2017         return (NULL);
2018     }

2020     if (lifr->lifr_addr.ss_family == AF_INET) {
2021         struct sockaddr_in *sin4;

2023         so_dst.sa.sa_family = AF_INET;
2024         sin4 = (struct sockaddr_in *)&lifr->lifr_addr;
2025         so_dst.sin.sin_addr = sin4->sin_addr;
2026     } else {
2027         struct sockaddr_in6 *sin6;

2029         so_dst.sa.sa_family = AF_INET6;
2030         sin6 = (struct sockaddr_in6 *)&lifr->lifr_addr;
2031         so_dst.sin6.sin6_addr = sin6->sin6_addr;
2032     }

2034     so_ifp.sa.sa_family = AF_LINK;

2036     (void) memset(&rtmsg, 0, sizeof (rtmsg));
2037     rtmsg.hdr.rtm_type = RTM_GET;
2038     rtmsg.hdr.rtm_flags = RTF_UP | RTF_HOST;

```

```

2039     rtmsg.hdr.rtm_version = RTM_VERSION;
2040     rtmsg.hdr.rtm_seq = ++rts_seqno;
2041     rtmsg.hdr.rtm_addrs = RTA_IFP | RTA_DST;

2043     l = ROUNDUP_LONG(salen(&so_dst.sa));
2044     (void) memmove(cp, &(so_dst), l);
2045     cp += l;
2046     l = ROUNDUP_LONG(salen(&so_ifp.sa));
2047     (void) memmove(cp, &(so_ifp), l);
2048     cp += l;

2050     rtmsg.hdr.rtm_msglen = l = cp - (char *)&rtmsg;

2052     if ((rlen = write(s, &rtmsg, l)) < 0) {
2053         zerror(zlogp, B_TRUE, "writing to routing socket");
2054         return (NULL);
2055     } else if (rlen < (int)rtmsg.hdr.rtm_msglen) {
2056         zerror(zlogp, B_TRUE,
2057             "write to routing socket got only %d for len\n", rlen);
2058         return (NULL);
2059     }
2060     do {
2061         l = read(s, &rtmsg, sizeof (rtmsg));
2062     } while (l > 0 && (rtmsg.hdr.rtm_seq != rts_seqno ||
2063         rtmsg.hdr.rtm_pid != pid));
2064     if (l < 0) {
2065         zerror(zlogp, B_TRUE, "reading from routing socket");
2066         return (NULL);
2067     }

2069     if (rtmsg.hdr.rtm_version != RTM_VERSION) {
2070         zerror(zlogp, B_FALSE,
2071             "routing message version %d not understood",
2072             rtmsg.hdr.rtm_version);
2073         return (NULL);
2074     }
2075     if (rtmsg.hdr.rtm_msglen != (ushort_t)l) {
2076         zerror(zlogp, B_FALSE, "message length mismatch, "
2077             "expected %d bytes, returned %d bytes",
2078             rtmsg.hdr.rtm_msglen, l);
2079         return (NULL);
2080     }
2081     if (rtmsg.hdr.rtm_errno != 0) {
2082         errno = rtmsg.hdr.rtm_errno;
2083         zerror(zlogp, B_TRUE, "RTM_GET routing socket message");
2084         return (NULL);
2085     }
2086     if ((rtmsg.hdr.rtm_addrs & RTA_IFP) == 0) {
2087         zerror(zlogp, B_FALSE, "network interface not found");
2088         return (NULL);
2089     }
2090     cp = ((char *)&rtmsg.hdr + l);
2091     for (i = 1; i != 0; i <= 1) {
2092         /* LINTED E_BAD_PTR_CAST_ALIGN */
2093         sa = (struct sockaddr *)cp;
2094         if (i != RTA_IFP) {
2095             if ((i & rtmsg.hdr.rtm_addrs) != 0)
2096                 cp += ROUNDUP_LONG(salen(sa));
2097             continue;
2098         }
2099         if (sa->sa_family == AF_LINK &&
2100             ((struct sockaddr_dl *)sa)->sdl_nlen != 0)
2101             ifp = (struct sockaddr_dl *)sa;
2102         break;
2103     }
2104     if (ifp == NULL) {

```

```

2105         zerror(zlogp, B_FALSE, "network interface could not be "
2106                 "determined");
2107         return (NULL);
2108     }
2109
2110     /*
2111     * We need to set the I/F name to what we got above, then do the
2112     * appropriate ioctl to get its zone name.  But lifr->lifr_name is
2113     * used by the calling function to do a REMOVEIF, so if we leave the
2114     * "good" zone's I/F name in place, *that* I/F will be removed instead
2115     * of the bad one.  So we save the old (bad) I/F name before over-
2116     * writing it and doing the ioctl, then restore it after the ioctl.
2117     */
2118     (void) strncpy(save_if_name, lifr->lifr_name, sizeof (save_if_name));
2119     (void) strncpy(lifr->lifr_name, ifp->sdl_data, ifp->sdl_nlen);
2120     lifr->lifr_name[ifp->sdl_nlen] = '\0';
2121     i = ioctl(s, SIOCGLIFZONE, lifr);
2122     (void) strncpy(lifr->lifr_name, save_if_name, sizeof (save_if_name));
2123     if (i < 0) {
2124         zerror(zlogp, B_TRUE,
2125              "%s: could not determine the zone network interface "
2126              "belongs to", lifr->lifr_name);
2127         return (NULL);
2128     }
2129     if (getzonenamebyid(lifr->lifr_zoneid, answer, sizeof (answer)) < 0)
2130         (void) snprintf(answer, sizeof (answer), "%d",
2131                        lifr->lifr_zoneid);
2132
2133     if (strlen(answer) > 0)
2134         return (answer);
2135     return (NULL);
2136 }
2137
2138 /*
2139 * Configures a single interface: a new virtual interface is added, based on
2140 * the physical interface nwifabptr->zone_nwif_physical, with the address
2141 * specified in nwifabptr->zone_nwif_address, for zone zone_id.  Note that
2142 * the "address" can be an IPv6 address (with a /prefixlength required), an
2143 * IPv4 address (with a /prefixlength optional), or a name; for the latter,
2144 * an IPv4 name-to-address resolution will be attempted.
2145 *
2146 * If anything goes wrong, we log an detailed error message, attempt to tear
2147 * down whatever we set up and return an error.
2148 */
2149 static int
2150 configure_one_interface(zlog_t *zlogp, zoneid_t zone_id,
2151                       struct zone_nwifab *nwifabptr)
2152 {
2153     struct lifreq lifr;
2154     struct sockaddr_in netmask4;
2155     struct sockaddr_in6 netmask6;
2156     struct sockaddr_storage laddr;
2157     struct in_addr in4;
2158     sa_family_t af;
2159     char *slash = strchr(nwifabptr->zone_nwif_address, '/');
2160     int s;
2161     boolean_t got_netmask = B_FALSE;
2162     boolean_t is_loopback = B_FALSE;
2163     char addrstr4[INET_ADDRSTRLEN];
2164     int res;
2165
2166     res = zonecfg_valid_net_address(nwifabptr->zone_nwif_address, &lifr);
2167     if (res != Z_OK) {
2168         zerror(zlogp, B_FALSE, "%s: %s", zonecfg_strerror(res),
2169              nwifabptr->zone_nwif_address);
2170         return (-1);

```

```

2171     }
2172     af = lifr.lifr_addr.ss_family;
2173     if (af == AF_INET)
2174         in4 = ((struct sockaddr_in *)&lifr.lifr_addr)->sin_addr;
2175     if ((s = socket(af, SOCK_DGRAM, 0)) < 0) {
2176         zerror(zlogp, B_TRUE, "could not get socket");
2177         return (-1);
2178     }
2179
2180     /*
2181     * This is a similar kind of "hack" like in addif() to get around
2182     * the problem of SIOCLIFADDIF.  The problem is that this ioctl
2183     * does not include the netmask when adding a logical interface.
2184     * To get around this problem, we first add the logical interface
2185     * with a 0 address.  After that, we set the netmask if provided.
2186     * Finally we set the interface address.
2187     */
2188     laddr = lifr.lifr_addr;
2189     (void) strncpy(lifr.lifr_name, nwifabptr->zone_nwif_physical,
2190                  sizeof (lifr.lifr_name));
2191     (void) memset(&lifr.lifr_addr, 0, sizeof (lifr.lifr_addr));
2192
2193     if (ioctl(s, SIOCLIFADDIF, (caddr_t)&lifr) < 0) {
2194         /*
2195         * Here, we know that the interface can't be brought up.
2196         * A similar warning message was already printed out to
2197         * the console by zoneadm(1M) so instead we log the
2198         * message to syslog and continue.
2199         */
2200         zerror(&zlogsys, B_TRUE, "WARNING: skipping network interface "
2201              "'%s' which may not be present/plumbed in the "
2202              "global zone.", lifr.lifr_name);
2203         (void) close(s);
2204         return (Z_OK);
2205     }
2206
2207     /* Preserve literal IPv4 address for later potential printing. */
2208     if (af == AF_INET)
2209         (void) inet_ntop(AF_INET, &in4, addrstr4, INET_ADDRSTRLEN);
2210
2211     lifr.lifr_zoneid = zone_id;
2212     if (ioctl(s, SIOCSLIFZONE, (caddr_t)&lifr) < 0) {
2213         zerror(zlogp, B_TRUE, "%s: could not place network interface "
2214              "into zone", lifr.lifr_name);
2215         goto bad;
2216     }
2217
2218     /*
2219     * Loopback interface will use the default netmask assigned, if no
2220     * netmask is found.
2221     */
2222     if (strcmp(nwifabptr->zone_nwif_physical, "lo0") == 0) {
2223         is_loopback = B_TRUE;
2224     }
2225     if (af == AF_INET) {
2226         /*
2227         * The IPv4 netmask can be determined either
2228         * directly if a prefix length was supplied with
2229         * the address or via the netmasks database.  Not
2230         * being able to determine it is a common failure,
2231         * but it often is not fatal to operation of the
2232         * interface.  In that case, a warning will be
2233         * printed after the rest of the interface's
2234         * parameters have been configured.
2235         */
2236         (void) memset(&netmask4, 0, sizeof (netmask4));

```

```

2237     if (slashp != NULL) {
2238         if (addr2netmask(slashp + 1, V4_ADDR_LEN,
2239             (uchar_t *)&netmask4.sin_addr) != 0) {
2240             *slashp = '/';
2241             zerror(zlogp, B_FALSE,
2242                 "%s: invalid prefix length in %s",
2243                 lifr.lifr_name,
2244                 nwiftabptr->zone_nwif_address);
2245             goto bad;
2246         }
2247         got_netmask = B_TRUE;
2248     } else if (getnetmaskbyaddr(in4,
2249         &netmask4.sin_addr) == 0) {
2250         got_netmask = B_TRUE;
2251     }
2252     if (got_netmask) {
2253         netmask4.sin_family = af;
2254         (void) memcpy(&lifr.lifr_addr, &netmask4,
2255             sizeof (netmask4));
2256     }
2257 } else {
2258     (void) memset(&netmask6, 0, sizeof (netmask6));
2259     if (addr2netmask(slashp + 1, V6_ADDR_LEN,
2260         (uchar_t *)&netmask6.sin6_addr) != 0) {
2261         *slashp = '/';
2262         zerror(zlogp, B_FALSE,
2263             "%s: invalid prefix length in %s",
2264             lifr.lifr_name,
2265             nwiftabptr->zone_nwif_address);
2266         goto bad;
2267     }
2268     got_netmask = B_TRUE;
2269     netmask6.sin6_family = af;
2270     (void) memcpy(&lifr.lifr_addr, &netmask6,
2271         sizeof (netmask6));
2272 }
2273 if (got_netmask &&
2274     ioctl(s, SIOCSLIFNETMASK, (caddr_t)&lifr) < 0) {
2275     zerror(zlogp, B_TRUE, "%s: could not set netmask",
2276         lifr.lifr_name);
2277     goto bad;
2278 }

2280 /* Set the interface address */
2281 lifr.lifr_addr = laddr;
2282 if (ioctl(s, SIOCSLIFADDR, (caddr_t)&lifr) < 0) {
2283     zerror(zlogp, B_TRUE,
2284         "%s: could not set IP address to %s",
2285         lifr.lifr_name, nwiftabptr->zone_nwif_address);
2286     goto bad;
2287 }

2289 if (ioctl(s, SIOCGLIFFLAGS, (caddr_t)&lifr) < 0) {
2290     zerror(zlogp, B_TRUE, "%s: could not get flags",
2291         lifr.lifr_name);
2292     goto bad;
2293 }
2294 lifr.lifr_flags |= IFF_UP;
2295 if (ioctl(s, SIOCSLIFFLAGS, (caddr_t)&lifr) < 0) {
2296     int save_errno = errno;
2297     char *zone_using;

2299     /*
2300     * If we failed with something other than EADDRNOTAVAIL,
2301     * then skip to the end. Otherwise, look up our address,
2302     * then call a function to determine which zone is already

```

```

2303     * using that address.
2304     */
2305     if (errno != EADDRNOTAVAIL) {
2306         zerror(zlogp, B_TRUE,
2307             "%s: could not bring network interface up",
2308             lifr.lifr_name);
2309         goto bad;
2310     }
2311     if (ioctl(s, SIOCGLIFADDR, (caddr_t)&lifr) < 0) {
2312         zerror(zlogp, B_TRUE, "%s: could not get address",
2313             lifr.lifr_name);
2314         goto bad;
2315     }
2316     zone_using = who_is_using(zlogp, &lifr);
2317     errno = save_errno;
2318     if (zone_using == NULL)
2319         zerror(zlogp, B_TRUE,
2320             "%s: could not bring network interface up",
2321             lifr.lifr_name);
2322     else
2323         zerror(zlogp, B_TRUE, "%s: could not bring network "
2324             "interface up: address in use by zone '%s'",
2325             lifr.lifr_name, zone_using);
2326     goto bad;
2327 }

2329 if (!got_netmask && !is_loopback) {
2330     /*
2331     * A common, but often non-fatal problem, is that the system
2332     * cannot find the netmask for an interface address. This is
2333     * often caused by it being only in /etc/inet/netmasks, but
2334     * /etc/nsswitch.conf says to use NIS or NIS+ and it's not
2335     * in that. This doesn't show up at boot because the netmask
2336     * is obtained from /etc/inet/netmasks when no network
2337     * interfaces are up, but isn't consulted when NIS/NIS+ is
2338     * available. We warn the user here that something like this
2339     * has happened and we're just running with a default and
2340     * possible incorrect netmask.
2341     */
2342     char buffer[INET6_ADDRSTRLEN];
2343     void *addr;
2344     const char *nomatch = "no matching subnet found in netmasks(4)";

2346     if (af == AF_INET)
2347         addr = &((struct sockaddr_in *)
2348             (&lifr.lifr_addr))->sin_addr;
2349     else
2350         addr = &((struct sockaddr_in6 *)
2351             (&lifr.lifr_addr))->sin6_addr;

2353     /*
2354     * Find out what netmask the interface is going to be using.
2355     * If we just brought up an IPMP data address on an underlying
2356     * interface above, the address will have already migrated, so
2357     * the SIOCGLIFNETMASK won't be able to find it (but we need
2358     * to bring the address up to get the actual netmask). Just
2359     * omit printing the actual netmask in this corner-case.
2360     */
2361     if (ioctl(s, SIOCGLIFNETMASK, (caddr_t)&lifr) < 0 ||
2362         inet_ntop(af, addr, buffer, sizeof (buffer)) == NULL) {
2363         zerror(zlogp, B_FALSE, "WARNING: %s; using default.",
2364             nomatch);
2365     } else {
2366         zerror(zlogp, B_FALSE,
2367             "WARNING: %s: %s: %s; using default of %s.",
2368             lifr.lifr_name, nomatch, addrstr4, buffer);

```

```

2369     }
2370 }
2372 /*
2373  * If a default router was specified for this interface
2374  * set the route now. Ignore if already set.
2375  */
2376 if (strlen(nwifabptr->zone_nwif_defrouter) > 0) {
2377     int status;
2378     char *argv[7];
2380     argv[0] = "route";
2381     argv[1] = "add";
2382     argv[2] = "-ifp";
2383     argv[3] = nwifabptr->zone_nwif_physical;
2384     argv[4] = "default";
2385     argv[5] = nwifabptr->zone_nwif_defrouter;
2386     argv[6] = NULL;
2388     status = forkexec(zlogp, "/usr/sbin/route", argv);
2389     if (status != 0 && status != EEXIST)
2390         zerror(zlogp, B_FALSE, "Unable to set route for "
2391             "interface %s to %s\n",
2392             nwifabptr->zone_nwif_physical,
2393             nwifabptr->zone_nwif_defrouter);
2394 }
2396 (void) close(s);
2397 return (Z_OK);
2398 bad:
2399 (void) ioctl(s, SIOCLIFREMOVEIF, (caddr_t)&lifr);
2400 (void) close(s);
2401 return (-1);
2402 }
2404 /*
2405  * Sets up network interfaces based on information from the zone configuration.
2406  * IPv4 and IPv6 loopback interfaces are set up "for free", modeling the global
2407  * system.
2408  *
2409  * If anything goes wrong, we log a general error message, attempt to tear down
2410  * whatever we set up, and return an error.
2411  */
2412 static int
2413 configure_shared_network_interfaces(zlog_t *zlogp)
2414 {
2415     zone_dochandle_t handle;
2416     struct zone_nwifab nwifab, loopback_iftab;
2417     zoneid_t zoneid;
2419     if ((zoneid = getzoneidbyname(zone_name)) == ZONE_ID_UNDEFINED) {
2420         zerror(zlogp, B_TRUE, "unable to get zoneid");
2421         return (-1);
2422     }
2424     if ((handle = zonecfg_init_handle()) == NULL) {
2425         zerror(zlogp, B_TRUE, "getting zone configuration handle");
2426         return (-1);
2427     }
2428     if (zonecfg_get_snapshot_handle(zone_name, handle) != Z_OK) {
2429         zerror(zlogp, B_FALSE, "invalid configuration");
2430         zonecfg_fini_handle(handle);
2431         return (-1);
2432     }
2433     if (zonecfg_setnwifent(handle) == Z_OK) {
2434         for (;;) {

```

```

2435         if (zonecfg_getnwifent(handle, &nwifab) != Z_OK)
2436             break;
2437         if (configure_one_interface(zlogp, zoneid, &nwifab) !=
2438             Z_OK) {
2439             (void) zonecfg_endnwifent(handle);
2440             zonecfg_fini_handle(handle);
2441             return (-1);
2442         }
2443     }
2444     (void) zonecfg_endnwifent(handle);
2445 }
2446 zonecfg_fini_handle(handle);
2447 if (is_system_labeled()) {
2448     /*
2449     * Labeled zones share the loopback interface
2450     * so it is not plumbed for shared stack instances.
2451     */
2452     return (0);
2453 }
2454 (void) strncpy(loopback_iftab.zone_nwif_physical, "lo0",
2455     sizeof(loopback_iftab.zone_nwif_physical));
2456 (void) strncpy(loopback_iftab.zone_nwif_address, "127.0.0.1",
2457     sizeof(loopback_iftab.zone_nwif_address));
2458 loopback_iftab.zone_nwif_defrouter[0] = '\0';
2459 if (configure_one_interface(zlogp, zoneid, &loopback_iftab) != Z_OK)
2460     return (-1);
2462 /* Always plumb up the IPv6 loopback interface. */
2463 (void) strncpy(loopback_iftab.zone_nwif_address, "::1/128",
2464     sizeof(loopback_iftab.zone_nwif_address));
2465 if (configure_one_interface(zlogp, zoneid, &loopback_iftab) != Z_OK)
2466     return (-1);
2467 return (0);
2468 }
2470 static void
2471 zdllerror(zlog_t *zlogp, dladm_status_t err, const char *dlname, const char *str)
2472 {
2473     char errmsg[DLADM_STRSIZE];
2475     (void) dladm_status2str(err, errmsg);
2476     zerror(zlogp, B_FALSE, "%s '%s': %s", str, dlname, errmsg);
2477 }
2479 static int
2480 add_datalink(zlog_t *zlogp, char *zone_name, datalink_id_t linkid, char *dlname)
2481 {
2482     dladm_status_t err;
2483     boolean_t cpuset, poolset;
2484     char *poolp;
2486     /* First check if it's in use by global zone. */
2487     if (zonecfg_ifname_exists(AF_INET, dlname) ||
2488         zonecfg_ifname_exists(AF_INET6, dlname)) {
2489         zerror(zlogp, B_FALSE, "WARNING: skipping network interface "
2490             "'%s' which is used in the global zone", dlname);
2491         return (-1);
2492     }
2494     /* Set zoneid of this link. */
2495     err = dladm_set_linkprop(dld_handle, linkid, "zone", &zone_name, 1,
2496         DLADM_OPT_ACTIVE);
2497     if (err != DLADM_STATUS_OK) {
2498         zdllerror(zlogp, err, dlname,
2499             "WARNING: unable to add network interface");
2500         return (-1);

```

```

2501     }
2502
2503     /*
2504     * Set the pool of this link if the zone has a pool and
2505     * neither the cpus nor the pool datalink property is
2506     * already set.
2507     */
2508     err = dladm_linkprop_is_set(dld_handle, linkid, DLADM_PROP_VAL_CURRENT,
2509     "cpus", &cpuset);
2510     if (err != DLADM_STATUS_OK) {
2511         zdlerror(zlogp, err, dlname,
2512         "WARNING: unable to check if cpus link property is set");
2513     }
2514     err = dladm_linkprop_is_set(dld_handle, linkid, DLADM_PROP_VAL_CURRENT,
2515     "pool", &poolset);
2516     if (err != DLADM_STATUS_OK) {
2517         zdlerror(zlogp, err, dlname,
2518         "WARNING: unable to check if pool link property is set");
2519     }
2520
2521     if ((strlen(pool_name) != 0) && !cpuset && !poolset) {
2522         poolp = pool_name;
2523         err = dladm_set_linkprop(dld_handle, linkid, "pool",
2524         &poolp, 1, DLADM_OPT_ACTIVE);
2525         if (err != DLADM_STATUS_OK) {
2526             zerror(zlogp, B_FALSE, "WARNING: unable to set "
2527             "pool %s to datalink %s", pool_name, dlname);
2528             bzero(pool_name, sizeof (pool_name));
2529         }
2530     } else {
2531         bzero(pool_name, sizeof (pool_name));
2532     }
2533     return (0);
2534 }
2535
2536 static boolean_t
2537 sockaddr_to_str(sa_family_t af, const struct sockaddr *sockaddr,
2538 char *straddr, size_t len)
2539 {
2540     struct sockaddr_in *sin;
2541     struct sockaddr_in6 *sin6;
2542     const char *str = NULL;
2543
2544     if (af == AF_INET) {
2545         /* LINTED E_BAD_PTR_CAST_ALIGN */
2546         sin = SIN(sockaddr);
2547         str = inet_ntop(AF_INET, (void *)&sin->sin_addr, straddr, len);
2548     } else if (af == AF_INET6) {
2549         /* LINTED E_BAD_PTR_CAST_ALIGN */
2550         sin6 = SIN6(sockaddr);
2551         str = inet_ntop(AF_INET6, (void *)&sin6->sin6_addr, straddr,
2552         len);
2553     }
2554
2555     return (str != NULL);
2556 }
2557
2558 static int
2559 ipv4_prefixlen(struct sockaddr_in *sin)
2560 {
2561     struct sockaddr_in *m;
2562     struct sockaddr_storage mask;
2563
2564     m = SIN(&mask);
2565     m->sin_family = AF_INET;
2566     if (getnetmaskbyaddr(sin->sin_addr, &m->sin_addr) == 0) {

```

```

2567         return (mask2plen((struct sockaddr *)&mask));
2568     } else if (IN_CLASSA(htonl(sin->sin_addr.s_addr))) {
2569         return (8);
2570     } else if (IN_CLASSB(htonl(sin->sin_addr.s_addr))) {
2571         return (16);
2572     } else if (IN_CLASSC(htonl(sin->sin_addr.s_addr))) {
2573         return (24);
2574     }
2575     return (0);
2576 }
2577
2578 static int
2579 zone_setattr_network(int type, zoneid_t zoneid, datalink_id_t linkid,
2580 void *buf, size_t bufsize)
2581 {
2582     zone_net_data_t *zndata;
2583     size_t znsz;
2584     int err;
2585
2586     znsz = sizeof (*zndata) + bufsize;
2587     zndata = calloc(1, znsz);
2588     if (zndata == NULL)
2589         return (ENOMEM);
2590     zndata->zn_type = type;
2591     zndata->zn_len = bufsize;
2592     zndata->zn_linkid = linkid;
2593     bcopy(buf, zndata->zn_val, zndata->zn_len);
2594     err = zone_setattr(zoneid, ZONE_ATTR_NETWORK, zndata, znsz);
2595     free(zndata);
2596     return (err);
2597 }
2598
2599 static int
2600 add_net_for_linkid(zlog_t *zlogp, zoneid_t zoneid, zone_addr_list_t *start)
2601 {
2602     struct lifreq lifr;
2603     char **astr, *address;
2604     dladm_status_t dlstatus;
2605     char *ip_nospoof = "ip-nospoof";
2606     int nnet, naddr, err = 0, j;
2607     size_t zlen, cpleft;
2608     zone_addr_list_t *ptr, *end;
2609     char tmp[INET6_ADDRSTRLEN], *maskstr;
2610     char *zaddr, *cp;
2611     struct in6_addr *routes = NULL;
2612     boolean_t is_set;
2613     datalink_id_t linkid;
2614
2615     assert(start != NULL);
2616     naddr = 0; /* number of addresses */
2617     nnet = 0; /* number of net resources */
2618     linkid = start->za_linkid;
2619     for (ptr = start; ptr != NULL && ptr->za_linkid == linkid;
2620         ptr = ptr->za_next) {
2621         nnet++;
2622     }
2623     end = ptr;
2624     zlen = nnet * (INET6_ADDRSTRLEN + 1);
2625     astr = calloc(1, nnet * sizeof (uintptr_t));
2626     zaddr = calloc(1, zlen);
2627     if (astr == NULL || zaddr == NULL) {
2628         err = ENOMEM;
2629         goto done;
2630     }
2631     cp = zaddr;
2632     cpleft = zlen;

```

```

2633     j = 0;
2634     for (ptr = start; ptr != end; ptr = ptr->za_next) {
2635         address = ptr->za_nwifab.zone_nwif_allowed_address;
2636         if (address[0] == '\0')
2637             continue;
2638         (void) snprintf(tmp, sizeof (tmp), "%s", address);
2639         /*
2640          * Validate the data. zonecfg_valid_net_address() clobbers
2641          * the /<mask> in the address string.
2642          */
2643         if (zonecfg_valid_net_address(address, &lifr) != Z_OK) {
2644             zerror(zlogp, B_FALSE, "invalid address [%s]\n",
2645                 address);
2646             err = EINVAL;
2647             goto done;
2648         }
2649         /*
2650          * convert any hostnames to numeric address strings.
2651          */
2652         if (!sockaddr_to_str(lifr.lifr_addr.ss_family,
2653             (const struct sockaddr *)&lifr.lifr_addr, cp, cpleft)) {
2654             err = EINVAL;
2655             goto done;
2656         }
2657         /*
2658          * make a copy of the numeric string for the data needed
2659          * by the "allowed-ips" datalink property.
2660          */
2661         astr[j] = strdup(cp);
2662         if (astr[j] == NULL) {
2663             err = ENOMEM;
2664             goto done;
2665         }
2666         j++;
2667         /*
2668          * compute the default netmask from the address, if necessary
2669          */
2670         if ((maskstr = strchr(tmp, '/')) == NULL) {
2671             int prefixlen;
2672
2673             if (lifr.lifr_addr.ss_family == AF_INET) {
2674                 prefixlen = ipv4_prefixlen(
2675                     SIN(&lifr.lifr_addr));
2676             } else {
2677                 struct sockaddr_in6 *sin6;
2678
2679                 sin6 = SIN6(&lifr.lifr_addr);
2680                 if (IN6_IS_ADDR_LINKLOCAL(&sin6->sin6_addr))
2681                     prefixlen = 10;
2682                 else
2683                     prefixlen = 64;
2684             }
2685             (void) snprintf(tmp, sizeof (tmp), "%d", prefixlen);
2686             maskstr = tmp;
2687         } else {
2688             maskstr++;
2689         }
2690         /* append the "/<netmask>" */
2691         (void) strlcat(cp, "/", cpleft);
2692         (void) strlcat(cp, maskstr, cpleft);
2693         (void) strlcat(cp, ",", cpleft);
2694         cp += strlen(cp, zlen);
2695         cpleft = &zaddr[INET6_ADDRSTRLEN] - cp;
2696     }
2697     naddr = j; /* the actual number of addresses in the net resource */
2698     assert(naddr <= nnet);

```

```

2700     /*
2701     * zonecfg has already verified that the defrouter property can only
2702     * be set if there is at least one address defined for the net resource.
2703     * If j is 0, there are no addresses defined, and therefore no routers
2704     * to configure, and we are done at that point.
2705     */
2706     if (j == 0)
2707         goto done;
2708
2709     /* over-write last ',' with '\0' */
2710     zaddr[strlen(zaddr, zlen) + 1] = '\0';
2711
2712     /*
2713     * First make sure L3 protection is not already set on the link.
2714     */
2715     dlstatus = dladm_linkprop_is_set(dld_handle, linkid, DLADM_OPT_ACTIVE,
2716         "protection", &is_set);
2717     if (dlstatus != DLADM_STATUS_OK) {
2718         err = EINVAL;
2719         zerror(zlogp, B_FALSE, "unable to check if protection is set");
2720         goto done;
2721     }
2722     if (is_set) {
2723         err = EINVAL;
2724         zerror(zlogp, B_FALSE, "Protection is already set");
2725         goto done;
2726     }
2727     dlstatus = dladm_linkprop_is_set(dld_handle, linkid, DLADM_OPT_ACTIVE,
2728         "allowed-ips", &is_set);
2729     if (dlstatus != DLADM_STATUS_OK) {
2730         err = EINVAL;
2731         zerror(zlogp, B_FALSE, "unable to check if allowed-ips is set");
2732         goto done;
2733     }
2734     if (is_set) {
2735         zerror(zlogp, B_FALSE, "allowed-ips is already set");
2736         err = EINVAL;
2737         goto done;
2738     }
2739
2740     /*
2741     * Enable ip-nospoof for the link, and add address to the allowed-ips
2742     * list.
2743     */
2744     dlstatus = dladm_set_linkprop(dld_handle, linkid, "protection",
2745         &ip_nospoof, 1, DLADM_OPT_ACTIVE);
2746     if (dlstatus != DLADM_STATUS_OK) {
2747         zerror(zlogp, B_FALSE, "could not set protection\n");
2748         err = EINVAL;
2749         goto done;
2750     }
2751     dlstatus = dladm_set_linkprop(dld_handle, linkid, "allowed-ips",
2752         astr, naddr, DLADM_OPT_ACTIVE);
2753     if (dlstatus != DLADM_STATUS_OK) {
2754         zerror(zlogp, B_FALSE, "could not set allowed-ips\n");
2755         err = EINVAL;
2756         goto done;
2757     }
2758
2759     /* now set the address in the data-store */
2760     err = zone_setattr_network(ZONE_NETWORK_ADDRESS, zoneid, linkid,
2761         zaddr, strlen(zaddr, zlen) + 1);
2762     if (err != 0)
2763         goto done;

```

```

2765 /*
2766  * add the default routers
2767  */
2768 routes = calloc(1, nnet * sizeof(*routes));
2769 j = 0;
2770 for (ptr = start; ptr != end; ptr = ptr->za_next) {
2771     address = ptr->za_nwif->zone_nwif_defrouter;
2772     if (address[0] == '\0')
2773         continue;
2774     if (strchr(address, '/') == NULL && strchr(address, ':') != 0) {
2775         /*
2776          * zonecfg_valid_net_address() expects numeric IPv6
2777          * addresses to have a CIDR format netmask.
2778          */
2779         (void) snprintf(tmp, sizeof(tmp), "%d", V6_ADDR_LEN);
2780         (void) strlcat(address, tmp, INET6_ADDRSTRLEN);
2781     }
2782     if (zonecfg_valid_net_address(address, &lifr) != Z_OK) {
2783         zerror(zlogp, B_FALSE,
2784             "invalid router [%s]\n", address);
2785         err = EINVAL;
2786         goto done;
2787     }
2788     if (lifr.lifr_addr.ss_family == AF_INET6) {
2789         routes[j] = SIN6(&lifr.lifr_addr)->sin6_addr;
2790     } else {
2791         IN6_INADDR_TO_V4MAPPED(&SIN(&lifr.lifr_addr)->sin_addr,
2792             &routes[j]);
2793     }
2794     j++;
2795 }
2796 assert(j <= nnet);
2797 if (j > 0) {
2798     err = zone_setattr_network(ZONE_NETWORK_DEFROUTER, zoneid,
2799         linkid, routes, j * sizeof(*routes));
2800 }
2801 done:
2802 free(routes);
2803 for (j = 0; j < naddr; j++)
2804     free(astr[j]);
2805 free(astr);
2806 free(zaddr);
2807 return (err);
2808 }
2809
2811 static int
2812 add_net(zlog_t *zlogp, zoneid_t zoneid, zone_addr_list_t *zalist)
2813 {
2814     zone_addr_list_t *ptr;
2815     datalink_id_t linkid;
2816     int err;
2817
2818     if (zalist == NULL)
2819         return (0);
2820
2821     linkid = zalist->za_linkid;
2822
2823     err = add_net_for_linkid(zlogp, zoneid, zalist);
2824     if (err != 0)
2825         return (err);
2826
2827     for (ptr = zalist; ptr != NULL; ptr = ptr->za_next) {
2828         if (ptr->za_linkid == linkid)
2829             continue;
2830         linkid = ptr->za_linkid;

```

```

2831         err = add_net_for_linkid(zlogp, zoneid, ptr);
2832         if (err != 0)
2833             return (err);
2834     }
2835     return (0);
2836 }
2837
2838 /*
2839  * Add "new" to the list of network interfaces to be configured by
2840  * add_net on zone boot in "old". The list of interfaces in "old" is
2841  * sorted by datalink_id_t, with interfaces sorted FIFO for a given
2842  * datalink_id_t.
2843  *
2844  * Returns the merged list of IP interfaces containing "old" and "new"
2845  */
2846 static zone_addr_list_t *
2847 add_ip_interface(zone_addr_list_t *old, zone_addr_list_t *new)
2848 {
2849     zone_addr_list_t *ptr, *next;
2850     datalink_id_t linkid = new->za_linkid;
2851
2852     assert(old != new);
2853
2854     if (old == NULL)
2855         return (new);
2856     for (ptr = old; ptr != NULL; ptr = ptr->za_next) {
2857         if (ptr->za_linkid == linkid)
2858             break;
2859     }
2860     if (ptr == NULL) {
2861         /* linkid does not already exist, add to the beginning */
2862         new->za_next = old;
2863         return (new);
2864     }
2865     /*
2866      * adding to the middle of the list; ptr points at the first
2867      * occurrence of linkid. Find the last occurrence.
2868      */
2869     while ((next = ptr->za_next) != NULL) {
2870         if (next->za_linkid != linkid)
2871             break;
2872         ptr = next;
2873     }
2874     /* insert new after ptr */
2875     new->za_next = next;
2876     ptr->za_next = new;
2877     return (old);
2878 }
2879
2880 void
2881 free_ip_interface(zone_addr_list_t *zalist)
2882 {
2883     zone_addr_list_t *ptr, *new;
2884
2885     for (ptr = zalist; ptr != NULL; ) {
2886         new = ptr;
2887         ptr = ptr->za_next;
2888         free(new);
2889     }
2890 }
2891
2892 /*
2893  * Add the kernel access control information for the interface names.
2894  * If anything goes wrong, we log a general error message, attempt to tear down
2895  * whatever we set up, and return an error.
2896  */

```



```

2897 static int
2898 configure_exclusive_network_interfaces(zlog_t *zlogp, zoneid_t zoneid)
2899 {
2900     zone_dochandle_t handle;
2901     struct zone_nwifTAB nwifTAB;
2902     char rootpath[MAXPATHLEN];
2903     char path[MAXPATHLEN];
2904     datalink_id_t linkid;
2905     di_prof_t prof = NULL;
2906     boolean_t added = B_FALSE;
2907     zone_addr_list_t *zalist = NULL, *new;
2908
2909     if ((handle = zonecfg_init_handle()) == NULL) {
2910         zerror(zlogp, B_TRUE, "getting zone configuration handle");
2911         return (-1);
2912     }
2913     if (zonecfg_get_snapshot(handle, zone_name, handle) != Z_OK) {
2914         zerror(zlogp, B_FALSE, "invalid configuration");
2915         zonecfg_fini_handle(handle);
2916         return (-1);
2917     }
2918
2919     if (zonecfg_setnwifent(handle) != Z_OK) {
2920         zonecfg_fini_handle(handle);
2921         return (0);
2922     }
2923
2924     for (;;) {
2925         if (zonecfg_getnwifent(handle, &nwifTAB) != Z_OK)
2926             break;
2927
2928         if (prof == NULL) {
2929             if (zone_get_devroot(zone_name, rootpath,
2930                 sizeof(rootpath)) != Z_OK) {
2931                 (void) zonecfg_endnwifent(handle);
2932                 zonecfg_fini_handle(handle);
2933                 zerror(zlogp, B_TRUE,
2934                     "unable to determine dev root");
2935                 return (-1);
2936             }
2937             (void) snprintf(path, sizeof(path), "%s%s", rootpath,
2938                 "/dev");
2939             if (di_prof_init(path, &prof) != 0) {
2940                 (void) zonecfg_endnwifent(handle);
2941                 zonecfg_fini_handle(handle);
2942                 zerror(zlogp, B_TRUE,
2943                     "failed to initialize profile");
2944                 return (-1);
2945             }
2946         }
2947
2948         /*
2949         * Create the /dev entry for backward compatibility.
2950         * Only create the /dev entry if it's not in use.
2951         * Note that the zone still boots when the assigned
2952         * interface is inaccessible, used by others, etc.
2953         * Also, when vanity naming is used, some interface do
2954         * not have corresponding /dev node names (for example,
2955         * vanity named aggregations). The /dev entry is not
2956         * created in that case. The /dev/net entry is always
2957         * accessible.
2958         */
2959         if (dladm_name2info(dld_handle, nwifTAB.zone_nwif_physical,
2960             &linkid, NULL, NULL) == DLADM_STATUS_OK &&
2961             add_datalink(zlogp, zone_name, linkid,
2962                 nwifTAB.zone_nwif_physical) == 0) {

```

```

2963         added = B_TRUE;
2964     } else {
2965         (void) zonecfg_endnwifent(handle);
2966         zonecfg_fini_handle(handle);
2967         zerror(zlogp, B_TRUE, "failed to add network device");
2968         return (-1);
2969     }
2970     /* set up the new IP interface, and add them all later */
2971     new = malloc(sizeof(*new));
2972     if (new == NULL) {
2973         zerror(zlogp, B_TRUE, "no memory for %s",
2974             nwifTAB.zone_nwif_physical);
2975         zonecfg_fini_handle(handle);
2976         free_ip_interface(zalist);
2977     }
2978     bzero(new, sizeof(*new));
2979     new->za_nwifTAB = nwifTAB;
2980     new->za_linkid = linkid;
2981     zalist = add_ip_interface(zalist, new);
2982
2983     if (zalist != NULL) {
2984         if ((errno = add_net(zlogp, zoneid, zalist)) != 0) {
2985             (void) zonecfg_endnwifent(handle);
2986             zonecfg_fini_handle(handle);
2987             zerror(zlogp, B_TRUE, "failed to add address");
2988             free_ip_interface(zalist);
2989             return (-1);
2990         }
2991         free_ip_interface(zalist);
2992     }
2993     (void) zonecfg_endnwifent(handle);
2994     zonecfg_fini_handle(handle);
2995
2996     if (prof != NULL && added) {
2997         if (di_prof_commit(prof) != 0) {
2998             zerror(zlogp, B_TRUE, "failed to commit profile");
2999             return (-1);
3000         }
3001     }
3002     if (prof != NULL)
3003         di_prof_fini(prof);
3004
3005     return (0);
3006 }
3007
3008 static int
3009 remove_datalink_pool(zlog_t *zlogp, zoneid_t zoneid)
3010 {
3011     ushort_t flags;
3012     zone_ipType_t ipType;
3013     int i, dlnum = 0;
3014     datalink_id_t *dlink, *dlinks = NULL;
3015     dladm_status_t err;
3016
3017     if (strlen(pool_name) == 0)
3018         return (0);
3019
3020     if (zone_getattr(zoneid, ZONE_ATTR_FLAGS, &flags,
3021         sizeof(flags)) < 0) {
3022         if (vplat_get_ipType(zlogp, &ipType) < 0) {
3023             zerror(zlogp, B_FALSE, "unable to determine ip-type");
3024             return (-1);
3025         }
3026     } else {
3027         if (flags & ZF_NET_EXCL)
3028             ipType = ZS_EXCLUSIVE;

```

```

3029     else
3030         iptype = ZS_SHARED;
3031     }
3032
3033     if (iptype == ZS_EXCLUSIVE) {
3034         /*
3035          * Get the datalink count and for each datalink,
3036          * attempt to clear the pool property and clear
3037          * the pool_name.
3038          */
3039         if (zone_list_datalink(zoneid, &dlnum, NULL) != 0) {
3040             zerror(zlogp, B_TRUE, "unable to count network "
3041                 "interfaces");
3042             return (-1);
3043         }
3044
3045         if (dlnum == 0)
3046             return (0);
3047
3048         if ((dllinks = malloc(dlnum * sizeof (datalink_id_t)))
3049             == NULL) {
3050             zerror(zlogp, B_TRUE, "memory allocation failed");
3051             return (-1);
3052         }
3053         if (zone_list_datalink(zoneid, &dlnum, dllinks) != 0) {
3054             zerror(zlogp, B_TRUE, "unable to list network "
3055                 "interfaces");
3056             return (-1);
3057         }
3058
3059         bzero(pool_name, sizeof (pool_name));
3060         for (i = 0, dlink = dllinks; i < dlnum; i++, dlink++) {
3061             err = dladm_set_linkprop(dld_handle, *dlink, "pool",
3062                 NULL, 0, DLADM_OPT_ACTIVE);
3063             if (err != DLADM_STATUS_OK) {
3064                 zerror(zlogp, B_TRUE,
3065                     "WARNING: unable to clear pool");
3066             }
3067         }
3068         free(dllinks);
3069     }
3070     return (0);
3071 }
3072
3073 static int
3074 remove_datalink_protect(zlog_t *zlogp, zoneid_t zoneid)
3075 {
3076     ushort_t flags;
3077     zone_iptype_t iptype;
3078     int i, dlnum = 0;
3079     dladm_status_t dlstatus;
3080     datalink_id_t *dlink, *dllinks = NULL;
3081
3082     if (zone_getattr(zoneid, ZONE_ATTR_FLAGS, &flags,
3083         sizeof (flags)) < 0) {
3084         if (vplat_get_iptype(zlogp, &iptype) < 0) {
3085             zerror(zlogp, B_FALSE, "unable to determine ip-type");
3086             return (-1);
3087         }
3088     } else {
3089         if (flags & ZF_NET_EXCL)
3090             iptype = ZS_EXCLUSIVE;
3091         else
3092             iptype = ZS_SHARED;
3093     }

```

```

3095     if (iptype != ZS_EXCLUSIVE)
3096         return (0);
3097
3098     /*
3099     * Get the datalink count and for each datalink,
3100     * attempt to clear the pool property and clear
3101     * the pool_name.
3102     */
3103     if (zone_list_datalink(zoneid, &dlnum, NULL) != 0) {
3104         zerror(zlogp, B_TRUE, "unable to count network interfaces");
3105         return (-1);
3106     }
3107
3108     if (dlnum == 0)
3109         return (0);
3110
3111     if ((dllinks = malloc(dlnum * sizeof (datalink_id_t))) == NULL) {
3112         zerror(zlogp, B_TRUE, "memory allocation failed");
3113         return (-1);
3114     }
3115     if (zone_list_datalink(zoneid, &dlnum, dllinks) != 0) {
3116         zerror(zlogp, B_TRUE, "unable to list network interfaces");
3117         free(dllinks);
3118         return (-1);
3119     }
3120
3121     for (i = 0, dlink = dllinks; i < dlnum; i++, dlink++) {
3122         char dlerr[DLADM_STRSIZE];
3123
3124         dlstatus = dladm_set_linkprop(dld_handle, *dlink,
3125             "protection", NULL, 0, DLADM_OPT_ACTIVE);
3126         if (dlstatus == DLADM_STATUS_NOTFOUND) {
3127             /* datalink does not belong to the GZ */
3128             continue;
3129         }
3130         if (dlstatus != DLADM_STATUS_OK) {
3131             zerror(zlogp, B_FALSE,
3132                 dladm_status2str(dlstatus, dlerr));
3133             free(dllinks);
3134             return (-1);
3135         }
3136         dlstatus = dladm_set_linkprop(dld_handle, *dlink,
3137             "allowed-ips", NULL, 0, DLADM_OPT_ACTIVE);
3138         if (dlstatus != DLADM_STATUS_OK) {
3139             zerror(zlogp, B_FALSE,
3140                 dladm_status2str(dlstatus, dlerr));
3141             free(dllinks);
3142             return (-1);
3143         }
3144     }
3145     free(dllinks);
3146     return (0);
3147 }
3148
3149 static int
3150 unconfigure_exclusive_network_interfaces(zlog_t *zlogp, zoneid_t zoneid)
3151 {
3152     int dlnum = 0;
3153
3154     /*
3155     * The kernel shutdown callback for the dls module should have removed
3156     * all datalinks from this zone. If any remain, then there's a
3157     * problem.
3158     */
3159     if (zone_list_datalink(zoneid, &dlnum, NULL) != 0) {
3160         zerror(zlogp, B_TRUE, "unable to list network interfaces");

```

```

3161         return (-1);
3162     }
3163     if (dlnum != 0) {
3164         zerror(zlogp, B_FALSE,
3165             "datalinks remain in zone after shutdown");
3166         return (-1);
3167     }
3168     return (0);
3169 }

3171 static int
3172 tcp_abort_conn(zlog_t *zlogp, zoneid_t zoneid,
3173     const struct sockaddr_storage *local, const struct sockaddr_storage *remote)
3174 {
3175     int fd;
3176     struct striocctl ioc;
3177     tcp_ioc_abort_conn_t conn;
3178     int error;

3180     conn.ac_local = *local;
3181     conn.ac_remote = *remote;
3182     conn.ac_start = TCPS_SYN_SENT;
3183     conn.ac_end = TCPS_TIME_WAIT;
3184     conn.ac_zoneid = zoneid;

3186     ioc.ic_cmd = TCP_IOC_ABORT_CONN;
3187     ioc.ic_timeout = -1; /* infinite timeout */
3188     ioc.ic_len = sizeof (conn);
3189     ioc.ic_dp = (char *)&conn;

3191     if ((fd = open("/dev/tcp", O_RDONLY)) < 0) {
3192         zerror(zlogp, B_TRUE, "unable to open %s", "/dev/tcp");
3193         return (-1);
3194     }

3196     error = ioctl(fd, I_STR, &ioc);
3197     (void) close(fd);
3198     if (error == 0 || errno == ENOENT) /* ENOENT is not an error */
3199         return (0);
3200     return (-1);
3201 }

3203 static int
3204 tcp_abort_connections(zlog_t *zlogp, zoneid_t zoneid)
3205 {
3206     struct sockaddr_storage l, r;
3207     struct sockaddr_in *local, *remote;
3208     struct sockaddr_in6 *local6, *remote6;
3209     int error;

3211     /*
3212      * Abort IPv4 connections.
3213      */
3214     bzero(&l, sizeof (*local));
3215     local = (struct sockaddr_in *)&l;
3216     local->sin_family = AF_INET;
3217     local->sin_addr.s_addr = INADDR_ANY;
3218     local->sin_port = 0;

3220     bzero(&r, sizeof (*remote));
3221     remote = (struct sockaddr_in *)&r;
3222     remote->sin_family = AF_INET;
3223     remote->sin_addr.s_addr = INADDR_ANY;
3224     remote->sin_port = 0;

3226     if ((error = tcp_abort_conn(zlogp, zoneid, &l, &r)) != 0)

```

```

3227         return (error);

3229     /*
3230      * Abort IPv6 connections.
3231      */
3232     bzero(&l, sizeof (*local6));
3233     local6 = (struct sockaddr_in6 *)&l;
3234     local6->sin6_family = AF_INET6;
3235     local6->sin6_port = 0;
3236     local6->sin6_addr = in6addr_any;

3238     bzero(&r, sizeof (*remote6));
3239     remote6 = (struct sockaddr_in6 *)&r;
3240     remote6->sin6_family = AF_INET6;
3241     remote6->sin6_port = 0;
3242     remote6->sin6_addr = in6addr_any;

3244     if ((error = tcp_abort_conn(zlogp, zoneid, &l, &r)) != 0)
3245         return (error);
3246     return (0);
3247 }

3249 static int
3250 get_privset(zlog_t *zlogp, priv_set_t *privs, zone_mnt_t mount_cmd)
3251 {
3252     int error = -1;
3253     zone_dochandle_t handle;
3254     char *privname = NULL;

3256     if ((handle = zonecfg_init_handle()) == NULL) {
3257         zerror(zlogp, B_TRUE, "getting zone configuration handle");
3258         return (-1);
3259     }
3260     if (zonecfg_get_snapshot_handle(zone_name, handle) != Z_OK) {
3261         zerror(zlogp, B_FALSE, "invalid configuration");
3262         zonecfg_fini_handle(handle);
3263         return (-1);
3264     }

3266     if (ALT_MOUNT(mount_cmd)) {
3267         zone_ipctype_t ipctype;
3268         const char *curr_ipctype;

3270         if (zonecfg_get_ipctype(handle, &ipctype) != Z_OK) {
3271             zerror(zlogp, B_TRUE, "unable to determine ip-type");
3272             zonecfg_fini_handle(handle);
3273             return (-1);
3274         }

3276         switch (ipctype) {
3277             case ZS_SHARED:
3278                 curr_ipctype = "shared";
3279                 break;
3280             case ZS_EXCLUSIVE:
3281                 curr_ipctype = "exclusive";
3282                 break;
3283         }

3285         if (zonecfg_default_privset(privs, curr_ipctype) == Z_OK) {
3286             zonecfg_fini_handle(handle);
3287             return (0);
3288         }
3289         zerror(zlogp, B_FALSE,
3290             "failed to determine the zone's default privilege set");
3291         zonecfg_fini_handle(handle);
3292         return (-1);

```

```

3293     }
3295     switch (zonecfg_get_privset(handle, privs, &privname)) {
3296     case Z_OK:
3297         error = 0;
3298         break;
3299     case Z_PRIV_PROHIBITED:
3300         zerror(zlogp, B_FALSE, "privilege \"%s\" is not permitted "
3301             "within the zone's privilege set", privname);
3302         break;
3303     case Z_PRIV_REQUIRED:
3304         zerror(zlogp, B_FALSE, "required privilege \"%s\" is missing "
3305             "from the zone's privilege set", privname);
3306         break;
3307     case Z_PRIV_UNKNOWN:
3308         zerror(zlogp, B_FALSE, "unknown privilege \"%s\" specified "
3309             "in the zone's privilege set", privname);
3310         break;
3311     default:
3312         zerror(zlogp, B_FALSE, "failed to determine the zone's "
3313             "privilege set");
3314         break;
3315     }
3317     free(privname);
3318     zonecfg_fini_handle(handle);
3319     return (error);
3320 }
3322 static int
3323 get_rctls(zlog_t *zlogp, char **bufp, size_t *bufsizep)
3324 {
3325     nvlist_t *nvl = NULL;
3326     char *nvl_packed = NULL;
3327     size_t nvl_size = 0;
3328     nvlist_t **nvlv = NULL;
3329     int rctlcount = 0;
3330     int error = -1;
3331     zone_dochandle_t handle;
3332     struct zone_rctltab rctltab;
3333     rctlblk_t *rctlblk = NULL;
3334     uint64_t maxlwps;
3335     uint64_t maxprocs;
3337     *bufp = NULL;
3338     *bufsizep = 0;
3340     if ((handle = zonecfg_init_handle()) == NULL) {
3341         zerror(zlogp, B_TRUE, "getting zone configuration handle");
3342         return (-1);
3343     }
3344     if (zonecfg_get_snapshot_handle(zone_name, handle) != Z_OK) {
3345         zerror(zlogp, B_FALSE, "invalid configuration");
3346         zonecfg_fini_handle(handle);
3347         return (-1);
3348     }
3350     rctltab.zone_rctl_valptr = NULL;
3351     if (nvlist_alloc(&nvl, NV_UNIQUE_NAME, 0) != 0) {
3352         zerror(zlogp, B_TRUE, "%s failed", "nvlist_alloc");
3353         goto out;
3354     }
3356     /*
3357     * Allow the administrator to control both the maximum number of
3358     * process table slots and the maximum number of lwps with just the

```

```

3359     * max-processes property.  If only the max-processes property is set,
3360     * we add a max-lwps property with a limit derived from max-processes.
3361     */
3362     if (zonecfg_get_aliased_rctl(handle, ALIAS_MAXPROCS, &maxprocs)
3363         == Z_OK &&
3364         zonecfg_get_aliased_rctl(handle, ALIAS_MAXLWPS, &maxlwps)
3365         == Z_NO_ENTRY) {
3366         if (zonecfg_set_aliased_rctl(handle, ALIAS_MAXLWPS,
3367             maxprocs * LWPS_PER_PROCESS) != Z_OK) {
3368             zerror(zlogp, B_FALSE, "unable to set max-lwps alias");
3369             goto out;
3370         }
3371     }
3373     if (zonecfg_setrctlent(handle) != Z_OK) {
3374         zerror(zlogp, B_FALSE, "%s failed", "zonecfg_setrctlent");
3375         goto out;
3376     }
3378     if ((rctlblk = malloc(rctlblk_size())) == NULL) {
3379         zerror(zlogp, B_TRUE, "memory allocation failed");
3380         goto out;
3381     }
3382     while (zonecfg_getrctlent(handle, &rctltab) == Z_OK) {
3383         struct zone_rctlvaltab *rctlval;
3384         uint_t i, count;
3385         const char *name = rctltab.zone_rctl_name;
3387         /* zoneadm should have already warned about unknown rctls. */
3388         if (!zonecfg_is_rctl(name)) {
3389             zonecfg_free_rctl_value_list(rctltab.zone_rctl_valptr);
3390             rctltab.zone_rctl_valptr = NULL;
3391             continue;
3392         }
3393         count = 0;
3394         for (rctlval = rctltab.zone_rctl_valptr; rctlval != NULL;
3395             rctlval = rctlval->zone_rctlval_next) {
3396             count++;
3397         }
3398         if (count == 0) { /* ignore */
3399             continue; /* Nothing to free */
3400         }
3401         if ((nvlv = malloc(sizeof(*nvlv) * count)) == NULL)
3402             goto out;
3403         i = 0;
3404         for (rctlval = rctltab.zone_rctl_valptr; rctlval != NULL;
3405             rctlval = rctlval->zone_rctlval_next, i++) {
3406             if (nvlist_alloc(&nvlv[i], NV_UNIQUE_NAME, 0) != 0) {
3407                 zerror(zlogp, B_TRUE, "%s failed",
3408                     "nvlist_alloc");
3409                 goto out;
3410             }
3411             if (zonecfg_construct_rctlblk(rctlval, rctlblk)
3412                 != Z_OK) {
3413                 zerror(zlogp, B_FALSE, "invalid rctl value: "
3414                     "(priv=%s,limit=%s,action=%s)",
3415                     rctlval->zone_rctlval_priv,
3416                     rctlval->zone_rctlval_limit,
3417                     rctlval->zone_rctlval_action);
3418                 goto out;
3419             }
3420             if (!zonecfg_valid_rctl(name, rctlblk)) {
3421                 zerror(zlogp, B_FALSE,
3422                     "(priv=%s,limit=%s,action=%s) is not a "
3423                     "valid value for rctl '%s'",
3424                     rctlval->zone_rctlval_priv,

```

```

3425         rctlval->zone_rctlval_limit,
3426         rctlval->zone_rctlval_action,
3427         name);
3428         goto out;
3429     }
3430     if (nvlist_add_uint64(nvlgv[i], "privilege",
3431         rctlblk_get_privilege(rctlblk)) != 0) {
3432         zerror(zlogp, B_FALSE, "%s failed",
3433             "nvlist_add_uint64");
3434         goto out;
3435     }
3436     if (nvlist_add_uint64(nvlgv[i], "limit",
3437         rctlblk_get_value(rctlblk)) != 0) {
3438         zerror(zlogp, B_FALSE, "%s failed",
3439             "nvlist_add_uint64");
3440         goto out;
3441     }
3442     if (nvlist_add_uint64(nvlgv[i], "action",
3443         (uint_t)rctlblk_get_local_action(rctlblk, NULL))
3444         != 0) {
3445         zerror(zlogp, B_FALSE, "%s failed",
3446             "nvlist_add_uint64");
3447         goto out;
3448     }
3449 }
3450 zonecfg_free_rctl_value_list(rctltab.zone_rctl_valptr);
3451 rctltab.zone_rctl_valptr = NULL;
3452 if (nvlist_add_nvlist_array(nvl, (char *)name, nvlgv, count)
3453     != 0) {
3454     zerror(zlogp, B_FALSE, "%s failed",
3455         "nvlist_add_nvlist_array");
3456     goto out;
3457 }
3458 for (i = 0; i < count; i++)
3459     nvlist_free(nvlgv[i]);
3460 free(nvlgv);
3461 nvlgv = NULL;
3462 rctlcount++;
3463 }
3464 (void) zonecfg_endrctlent(handle);

3466 if (rctlcount == 0) {
3467     error = 0;
3468     goto out;
3469 }
3470 if (nvlist_pack(nvl, &nvl_packed, &nvl_size, NV_ENCODE_NATIVE, 0)
3471     != 0) {
3472     zerror(zlogp, B_FALSE, "%s failed", "nvlist_pack");
3473     goto out;
3474 }

3476 error = 0;
3477 *bufp = nvl_packed;
3478 *bufsizep = nvl_size;

3480 out:
3481 free(rctlblk);
3482 zonecfg_free_rctl_value_list(rctltab.zone_rctl_valptr);
3483 if (error && nvl_packed != NULL)
3484     free(nvl_packed);
3485 nvlist_free(nvl);
3486 if (nvlgv != NULL)
3487     free(nvlgv);
3488 if (handle != NULL)
3489     zonecfg_fini_handle(handle);
3490 return (error);

```

```

3491 }

3493 static int
3494 get_implicit_datasets(zlog_t *zlogp, char **retstr)
3495 {
3496     char cmdbuf[2 * MAXPATHLEN];

3498     if (query_hook[0] == '\0')
3499         return (0);

3501     if (snprintf(cmdbuf, sizeof (cmdbuf), "%s datasets", query_hook)
3502         > sizeof (cmdbuf))
3503         return (-1);

3505     if (do_subproc(zlogp, cmdbuf, retstr) != 0)
3506         return (-1);

3508     return (0);
3509 }

3511 static int
3512 get_datasets(zlog_t *zlogp, char **bufp, size_t *bufsizep)
3513 {
3514     zone_dochandle_t handle;
3515     struct zone_dstab dstab;
3516     size_t total, offset, len;
3517     int error = -1;
3518     char *str = NULL;
3519     char *implicit_datasets = NULL;
3520     int implicit_len = 0;

3522     *bufp = NULL;
3523     *bufsizep = 0;

3525     if ((handle = zonecfg_init_handle()) == NULL) {
3526         zerror(zlogp, B_TRUE, "getting zone configuration handle");
3527         return (-1);
3528     }
3529     if (zonecfg_get_snapshot_handle(zone_name, handle) != Z_OK) {
3530         zerror(zlogp, B_FALSE, "invalid configuration");
3531         zonecfg_fini_handle(handle);
3532         return (-1);
3533     }

3535     if (get_implicit_datasets(zlogp, &implicit_datasets) != 0) {
3536         zerror(zlogp, B_FALSE, "getting implicit datasets failed");
3537         goto out;
3538     }

3540     if (zonecfg_setdsent(handle) != Z_OK) {
3541         zerror(zlogp, B_FALSE, "%s failed", "zonecfg_setdsent");
3542         goto out;
3543     }

3545     total = 0;
3546     while (zonecfg_getdsent(handle, &dstab) == Z_OK)
3547         total += strlen(dstab.zone_dataset_name) + 1;
3548     (void) zonecfg_enddsent(handle);

3550     if (implicit_datasets != NULL)
3551         implicit_len = strlen(implicit_datasets);
3552     if (implicit_len > 0)
3553         total += implicit_len + 1;

3555     if (total == 0) {
3556         error = 0;

```

```

3557         goto out;
3558     }

3560     if ((str = malloc(total)) == NULL) {
3561         zerror(zlogp, B_TRUE, "memory allocation failed");
3562         goto out;
3563     }

3565     if (zonecfg_setdsent(handle) != Z_OK) {
3566         zerror(zlogp, B_FALSE, "%s failed", "zonecfg_setdsent");
3567         goto out;
3568     }
3569     offset = 0;
3570     while (zonecfg_getdsent(handle, &dstab) == Z_OK) {
3571         len = strlen(dstab.zone_dataset_name);
3572         (void) strcpy(str + offset, dstab.zone_dataset_name,
3573             total - offset);
3574         offset += len;
3575         if (offset < total - 1)
3576             str[offset++] = ',';
3577     }
3578     (void) zonecfg_enddsent(handle);

3580     if (implicit_len > 0)
3581         (void) strcpy(str + offset, implicit_datasets, total - offset);

3583     error = 0;
3584     *bufp = str;
3585     *bufsizep = total;

3587 out:
3588     if (error != 0 && str != NULL)
3589         free(str);
3590     if (handle != NULL)
3591         zonecfg_fini_handle(handle);
3592     if (implicit_datasets != NULL)
3593         free(implicit_datasets);

3595     return (error);
3596 }

3598 static int
3599 validate_datasets(zlog_t *zlogp)
3600 {
3601     zone_dochandle_t handle;
3602     struct zone_dstab dstab;
3603     zfs_handle_t *zhp;
3604     libzfs_handle_t *hdl;

3606     if ((handle = zonecfg_init_handle()) == NULL) {
3607         zerror(zlogp, B_TRUE, "getting zone configuration handle");
3608         return (-1);
3609     }
3610     if (zonecfg_get_snapshot_handle(zone_name, handle) != Z_OK) {
3611         zerror(zlogp, B_FALSE, "invalid configuration");
3612         zonecfg_fini_handle(handle);
3613         return (-1);
3614     }

3616     if (zonecfg_setdsent(handle) != Z_OK) {
3617         zerror(zlogp, B_FALSE, "invalid configuration");
3618         zonecfg_fini_handle(handle);
3619         return (-1);
3620     }

3622     if ((hdl = libzfs_init()) == NULL) {

```

```

3623         zerror(zlogp, B_FALSE, "opening ZFS library");
3624         zonecfg_fini_handle(handle);
3625         return (-1);
3626     }

3628     while (zonecfg_getdsent(handle, &dstab) == Z_OK) {
3630         if ((zhp = zfs_open(hdl, dstab.zone_dataset_name,
3631             ZFS_TYPE_FILESYSTEM)) == NULL) {
3632             zerror(zlogp, B_FALSE, "cannot open ZFS dataset '%s'",
3633                 dstab.zone_dataset_name);
3634             zonecfg_fini_handle(handle);
3635             libzfs_fini(hdl);
3636             return (-1);
3637         }

3639         /*
3640          * Automatically set the 'zoned' property. We check the value
3641          * first because we'll get EPERM if it is already set.
3642          */
3643         if (!zfs_prop_get_int(zhp, ZFS_PROP_ZONED) &&
3644             zfs_prop_set(zhp, zfs_prop_to_name(ZFS_PROP_ZONED),
3645                 "on") != 0) {
3646             zerror(zlogp, B_FALSE, "cannot set 'zoned' "
3647                 "property for ZFS dataset '%s'\n",
3648                 dstab.zone_dataset_name);
3649             zonecfg_fini_handle(handle);
3650             zfs_close(zhp);
3651             libzfs_fini(hdl);
3652             return (-1);
3653         }

3655         zfs_close(zhp);
3656     }
3657     (void) zonecfg_enddsent(handle);

3659     zonecfg_fini_handle(handle);
3660     libzfs_fini(hdl);

3662     return (0);
3663 }

3665 /*
3666  * Return true if the path is its own zfs file system. We determine this
3667  * by stat-ing the path to see if it is zfs and stat-ing the parent to see
3668  * if it is a different fs.
3669  */
3670 boolean_t
3671 is_zonepath_zfs(char *zonepath)
3672 {
3673     int res;
3674     char *path;
3675     char *parent;
3676     struct statvfs64 buf1, buf2;

3678     if (statvfs64(zonepath, &buf1) != 0)
3679         return (B_FALSE);

3681     if (strcmp(buf1.f_basetype, "zfs") != 0)
3682         return (B_FALSE);

3684     if ((path = strdup(zonepath)) == NULL)
3685         return (B_FALSE);

3687     parent = dirname(path);
3688     res = statvfs64(parent, &buf2);

```

```

3689     free(path);
3691     if (res != 0)
3692         return (B_FALSE);
3694     if (buf1.f_fsid == buf2.f_fsid)
3695         return (B_FALSE);
3697     return (B_TRUE);
3698 }
3700 /*
3701  * Verify the MAC label in the root dataset for the zone.
3702  * If the label exists, it must match the label configured for the zone.
3703  * Otherwise if there's no label on the dataset, create one here.
3704  */
3706 static int
3707 validate_rootds_label(zlog_t *zlogp, char *rootpath, m_label_t *zone_sl)
3708 {
3709     int            error = -1;
3710     zfs_handle_t  *zhp;
3711     libzfs_handle_t *hdl;
3712     m_label_t     ds_sl;
3713     char          zonepath[MAXPATHLEN];
3714     char          ds_hexsl[MAXNAMELEN];
3716     if (!is_system_labeled())
3717         return (0);
3719     if (zone_get_zonepath(zone_name, zonepath, sizeof (zonepath)) != Z_OK) {
3720         zerror(zlogp, B_TRUE, "unable to determine zone path");
3721         return (-1);
3722     }
3724     if (!is_zonepath_zfs(zonepath))
3725         return (0);
3727     if ((hdl = libzfs_init()) == NULL) {
3728         zerror(zlogp, B_FALSE, "opening ZFS library");
3729         return (-1);
3730     }
3732     if ((zhp = zfs_path_to_zhandle(hdl, rootpath,
3733     ZFS_TYPE_FILESYSTEM)) == NULL) {
3734         zerror(zlogp, B_FALSE, "cannot open ZFS dataset for path '%s'",
3735     rootpath);
3736         libzfs_fini(hdl);
3737         return (-1);
3738     }
3740     /* Get the mlslabel property if it exists. */
3741     if ((zfs_prop_get(zhp, ZFS_PROP_MLSLABEL, ds_hexsl, MAXNAMELEN,
3742     NULL, NULL, 0, B_TRUE) != 0) ||
3743     (strcmp(ds_hexsl, ZFS_MLSLABEL_DEFAULT) == 0)) {
3744         char          *str2 = NULL;
3746         /*
3747          * No label on the dataset (or default only); create one.
3748          * (Only do this automatic labeling for the labeled brand.)
3749          */
3750         if (strcmp(brand_name, LABELED_BRAND_NAME) != 0) {
3751             error = 0;
3752             goto out;
3753         }

```

```

3755         error = l_to_str_internal(zone_sl, &str2);
3756         if (error)
3757             goto out;
3758         if (str2 == NULL) {
3759             error = -1;
3760             goto out;
3761         }
3762         if ((error = zfs_prop_set(zhp,
3763     zfs_prop_to_name(ZFS_PROP_MLSLABEL), str2)) != 0) {
3764             zerror(zlogp, B_FALSE, "cannot set 'mlslabel' "
3765     "property for root dataset at '%s'\n", rootpath);
3766         }
3767         free(str2);
3768         goto out;
3769     }
3771     /* Convert the retrieved dataset label to binary form. */
3772     error = hexstr_to_label(ds_hexsl, &ds_sl);
3773     if (error) {
3774         zerror(zlogp, B_FALSE, "invalid 'mlslabel' "
3775     "property on root dataset at '%s'\n", rootpath);
3776         goto out; /* exit with error */
3777     }
3779     /*
3780      * Perform a MAC check by comparing the zone label with the
3781      * dataset label.
3782      */
3783     error = (!bequal(zone_sl, &ds_sl));
3784     if (error)
3785         zerror(zlogp, B_FALSE, "Rootpath dataset has mismatched label");
3786 out:
3787     zfs_close(zhp);
3788     libzfs_fini(hdl);
3790     return (error);
3791 }
3793 /*
3794  * Mount lower level home directories into/from current zone
3795  * Share exported directories specified in dfstab for zone
3796  */
3797 static int
3798 tsol_mounts(zlog_t *zlogp, char *zone_name, char *rootpath)
3799 {
3800     zoneid_t *zids = NULL;
3801     priv_set_t *zid_privs;
3802     const priv_impl_info_t *ip = NULL;
3803     uint_t nzens_saved;
3804     uint_t nzens;
3805     int i;
3806     char readonly[] = "ro";
3807     struct zone_fstab lower_fstab;
3808     char *argv[4];
3810     if (!is_system_labeled())
3811         return (0);
3813     if (zid_label == NULL) {
3814         zid_label = m_label_alloc(MAC_LABEL);
3815         if (zid_label == NULL)
3816             return (-1);
3817     }
3819     /* Make sure our zone has an /export/home dir */
3820     (void) make_one_dir(zlogp, rootpath, "/export/home",

```

```

3821     DEFAULT_DIR_MODE, DEFAULT_DIR_USER, DEFAULT_DIR_GROUP);
3822
3823     lower_fstab.zone_fs_raw[0] = '\0';
3824     (void) strncpy(lower_fstab.zone_fs_type, MNTTYPE_LOFS,
3825                 sizeof(lower_fstab.zone_fs_type));
3826     lower_fstab.zone_fs_options = NULL;
3827     (void) zonecfg_add_fs_option(&lower_fstab, readonly);
3828
3829     /*
3830      * Get the list of zones from the kernel
3831      */
3832     if (zone_list(NULL, &nzents) != 0) {
3833         zerror(zlogp, B_TRUE, "unable to list zones");
3834         zonecfg_free_fs_option_list(lower_fstab.zone_fs_options);
3835         return (-1);
3836     }
3837     again:
3838     if (nzents == 0) {
3839         zonecfg_free_fs_option_list(lower_fstab.zone_fs_options);
3840         return (-1);
3841     }
3842
3843     zids = malloc(nzents * sizeof(zoneid_t));
3844     if (zids == NULL) {
3845         zerror(zlogp, B_TRUE, "memory allocation failed");
3846         return (-1);
3847     }
3848     nzents_saved = nzents;
3849
3850     if (zone_list(zids, &nzents) != 0) {
3851         zerror(zlogp, B_TRUE, "unable to list zones");
3852         zonecfg_free_fs_option_list(lower_fstab.zone_fs_options);
3853         free(zids);
3854         return (-1);
3855     }
3856     if (nzents != nzents_saved) {
3857         /* list changed, try again */
3858         free(zids);
3859         goto again;
3860     }
3861
3862     ip = getprivimplinfo();
3863     if ((zid_privs = priv_allocset()) == NULL) {
3864         zerror(zlogp, B_TRUE, "%s failed", "priv_allocset");
3865         zonecfg_free_fs_option_list(
3866             lower_fstab.zone_fs_options);
3867         free(zids);
3868         return (-1);
3869     }
3870
3871     for (i = 0; i < nzents; i++) {
3872         char zid_name[ZONENAME_MAX];
3873         zone_state_t zid_state;
3874         char zid_rpath[MAXPATHLEN];
3875         struct stat stat_buf;
3876
3877         if (zids[i] == GLOBAL_ZONEID)
3878             continue;
3879
3880         if (getzonenamebyid(zids[i], zid_name, ZONENAME_MAX) == -1)
3881             continue;
3882
3883         /*
3884          * Do special setup for the zone we are booting
3885          */
3886         if (strcmp(zid_name, zone_name) == 0) {

```

```

3887     struct zone_fstab autofstab;
3888     char map_path[MAXPATHLEN];
3889     int fd;
3890
3891     /*
3892      * Create auto_home_<zone> map for this zone
3893      * in the global zone. The non-global zone entry
3894      * will be created by automount when the zone
3895      * is booted.
3896      */
3897
3898     (void) snprintf(autofstab.zone_fs_special,
3899                   MAXPATHLEN, "auto_home_%s", zid_name);
3900
3901     (void) snprintf(autofstab.zone_fs_dir, MAXPATHLEN,
3902                   "/zone/%s/home", zid_name);
3903
3904     (void) snprintf(map_path, sizeof(map_path),
3905                   "/etc/%s", autofstab.zone_fs_special);
3906     /*
3907      * If the map file doesn't exist create a template
3908      */
3909     if ((fd = open(map_path, O_RDWR | O_CREAT | O_EXCL,
3910                  S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH)) != -1) {
3911         int len;
3912         char map_rec[MAXPATHLEN];
3913
3914         len = snprintf(map_rec, sizeof(map_rec),
3915                       "+%s\n\tfstype=lofs\t:%s/export/home/&\n",
3916                       autofstab.zone_fs_special, rootpath);
3917         (void) write(fd, map_rec, len);
3918         (void) close(fd);
3919     }
3920
3921     /*
3922      * Mount auto_home_<zone> in the global zone if absent.
3923      * If it's already of type autofstab, then
3924      * don't mount it again.
3925      */
3926     if ((stat(autofstab.zone_fs_dir, &stat_buf) == -1) ||
3927         strcmp(stat_buf.st_fstype, MNTTYPE_AUTOFS) != 0) {
3928         char optstr[] = "indirect,ignore,nobrowse";
3929
3930         (void) make_one_dir(zlogp, "",
3931                           autofstab.zone_fs_dir, DEFAULT_DIR_MODE,
3932                           DEFAULT_DIR_USER, DEFAULT_DIR_GROUP);
3933
3934         /*
3935          * Mount will fail if automounter has already
3936          * processed the auto_home_<zonename> map
3937          */
3938         (void) domount(zlogp, MNTTYPE_AUTOFS, optstr,
3939                       autofstab.zone_fs_special,
3940                       autofstab.zone_fs_dir);
3941     }
3942     continue;
3943 }
3944
3945     if (zone_get_state(zid_name, &zid_state) != Z_OK ||
3946         (zid_state != ZONE_STATE_READY &&
3947          zid_state != ZONE_STATE_RUNNING))
3948         /* Skip over zones without mounted filesystems */
3949         continue;
3950
3951     if (zone_getattr(zids[i], ZONE_ATTR_SLBL, zid_label,

```



```

3953         sizeof (m_label_t) < 0)
3954         /* Skip over zones with unspecified label */
3955         continue;

3957     if (zone_getattr(zids[i], ZONE_ATTR_ROOT, zid_rpath,
3958         sizeof (zid_rpath)) == -1)
3959         /* Skip over zones with bad path */
3960         continue;

3962     if (zone_getattr(zids[i], ZONE_ATTR_PRIVSET, zid_privs,
3963         sizeof (priv_chunk_t) * ip->priv_setsize) == -1)
3964         /* Skip over zones with bad privs */
3965         continue;

3967     /*
3968     * Reading down is valid according to our label model
3969     * but some customers want to disable it because it
3970     * allows execute down and other possible attacks.
3971     * Therefore, we restrict this feature to zones that
3972     * have the NET_MAC_AWARE privilege which is required
3973     * for NFS read-down semantics.
3974     */
3975     if ((bldominates(zlabel, zid_label)) &&
3976         (priv_ismember(zprivs, PRIV_NET_MAC_AWARE))) {
3977         /*
3978         * Our zone dominates this one.
3979         * Create a lofs mount from lower zone's /export/home
3980         */
3981         (void) snprintf(lower_fstab.zone_fs_dir, MAXPATHLEN,
3982             "%s/zone/%s/export/home", rootpath, zid_name);

3984         /*
3985         * If the target is already an LOFS mount
3986         * then don't do it again.
3987         */
3988         if ((stat(lower_fstab.zone_fs_dir, &stat_buf) == -1) ||
3989             strcmp(stat_buf.st_fstype, MNTTYPE_LOFS) != 0) {

3991             if (snprintf(lower_fstab.zone_fs_special,
3992                 MAXPATHLEN, "%s/export",
3993                 zid_rpath) > MAXPATHLEN)
3994                 continue;

3996             /*
3997             * Make sure the lower-level home exists
3998             */
3999             if (make_one_dir(zlogp,
4000                 lower_fstab.zone_fs_special, "/home",
4001                 DEFAULT_DIR_MODE, DEFAULT_DIR_USER,
4002                 DEFAULT_DIR_GROUP) != 0)
4003                 continue;

4005             (void) strlcat(lower_fstab.zone_fs_special,
4006                 "/home", MAXPATHLEN);

4008             /*
4009             * Mount can fail because the lower-level
4010             * zone may have already done a mount up.
4011             */
4012             (void) mount_one(zlogp, &lower_fstab, "",
4013                 Z_MNT_BOOT);
4014         }
4015     } else if ((bldominates(zid_label, zlabel)) &&
4016         (priv_ismember(zid_privs, PRIV_NET_MAC_AWARE))) {
4017         /*
4018         * This zone dominates our zone.

```

```

4019         /* Create a lofs mount from our zone's /export/home
4020         */
4021         if (snprintf(lower_fstab.zone_fs_dir, MAXPATHLEN,
4022             "%s/zone/%s/export/home", zid_rpath,
4023             zone_name) > MAXPATHLEN)
4024             continue;

4026         /*
4027         * If the target is already an LOFS mount
4028         * then don't do it again.
4029         */
4030         if ((stat(lower_fstab.zone_fs_dir, &stat_buf) == -1) ||
4031             strcmp(stat_buf.st_fstype, MNTTYPE_LOFS) != 0) {

4033             (void) snprintf(lower_fstab.zone_fs_special,
4034                 MAXPATHLEN, "%s/export/home", rootpath);

4036             /*
4037             * Mount can fail because the higher-level
4038             * zone may have already done a mount down.
4039             */
4040             (void) mount_one(zlogp, &lower_fstab, "",
4041                 Z_MNT_BOOT);
4042         }
4043     }
4044 }
4045 zonecfg_free_fs_option_list(lower_fstab.zone_fs_options);
4046 priv_freeset(zid_privs);
4047 free(zids);

4049 /*
4050 * Now share any exported directories from this zone.
4051 * Each zone can have its own dfstab.
4052 */

4054 argv[0] = "zoneshare";
4055 argv[1] = "-z";
4056 argv[2] = zone_name;
4057 argv[3] = NULL;

4059 (void) forkexec(zlogp, "/usr/lib/zones/zoneshare", argv);
4060 /* Don't check for errors since they don't affect the zone */

4062     return (0);
4063 }

4065 /*
4066 * Unmount lofs mounts from higher level zones
4067 * Unshare nfs exported directories
4068 */
4069 static void
4070 tsol_unmounts(zlog_t *zlogp, char *zone_name)
4071 {
4072     zoneid_t *zids = NULL;
4073     uint_t nzens_saved;
4074     uint_t nzens;
4075     int i;
4076     char *argv[4];
4077     char path[MAXPATHLEN];

4079     if (!is_system_labeled())
4080         return;

4082     /*
4083     * Get the list of zones from the kernel
4084     */

```

```

4085     if (zone_list(NULL, &nzents) != 0) {
4086         return;
4087     }

4089     if (zid_label == NULL) {
4090         zid_label = m_label_alloc(MAC_LABEL);
4091         if (zid_label == NULL)
4092             return;
4093     }

4095 again:
4096     if (nzents == 0)
4097         return;

4099     zids = malloc(nzents * sizeof (zoneid_t));
4100     if (zids == NULL) {
4101         zerror(zlogp, B_TRUE, "memory allocation failed");
4102         return;
4103     }
4104     nzents_saved = nzents;

4106     if (zone_list(zids, &nzents) != 0) {
4107         free(zids);
4108         return;
4109     }
4110     if (nzents != nzents_saved) {
4111         /* list changed, try again */
4112         free(zids);
4113         goto again;
4114     }

4116     for (i = 0; i < nzents; i++) {
4117         char zid_name[ZONENAME_MAX];
4118         zone_state_t zid_state;
4119         char zid_rpath[MAXPATHLEN];

4121         if (zids[i] == GLOBAL_ZONEID)
4122             continue;

4124         if (getzonenamebyid(zids[i], zid_name, ZONENAME_MAX) == -1)
4125             continue;

4127         /*
4128          * Skip the zone we are halting
4129          */
4130         if (strcmp(zid_name, zone_name) == 0)
4131             continue;

4133         if ((zone_getattr(zids[i], ZONE_ATTR_STATUS, &zid_state,
4134             sizeof (zid_state)) < 0) ||
4135             (zid_state < ZONE_IS_READY))
4136             /* Skip over zones without mounted filesystems */
4137             continue;

4139         if (zone_getattr(zids[i], ZONE_ATTR_SLBL, zid_label,
4140             sizeof (m_label_t)) < 0)
4141             /* Skip over zones with unspecified label */
4142             continue;

4144         if (zone_getattr(zids[i], ZONE_ATTR_ROOT, zid_rpath,
4145             sizeof (zid_rpath)) == -1)
4146             /* Skip over zones with bad path */
4147             continue;

4149         if (zlabel != NULL && bldominates(zid_label, zlabel)) {
4150             /*

```

```

4151         * This zone dominates our zone.
4152         * Unmount the lofs mount of our zone's /export/home
4153         */

4155         if (snprintf(path, MAXPATHLEN,
4156             "%s/zone/%s/export/home", zid_rpath,
4157             zone_name) > MAXPATHLEN)
4158             continue;

4160         /* Skip over mount failures */
4161         (void) umount(path);
4162     }
4163 }
4164 free(zids);

4166 /*
4167  * Unmount global zone autofs trigger for this zone
4168  */
4169 (void) snprintf(path, MAXPATHLEN, "/zone/%s/home", zone_name);
4170 /* Skip over mount failures */
4171 (void) umount(path);

4173 /*
4174  * Next unshare any exported directories from this zone.
4175  */

4177     argv[0] = "zoneunshare";
4178     argv[1] = "-z";
4179     argv[2] = zone_name;
4180     argv[3] = NULL;

4182     (void) forkexec(zlogp, "/usr/lib/zones/zoneunshare", argv);
4183     /* Don't check for errors since they don't affect the zone */

4185     /*
4186      * Finally, deallocate any devices in the zone.
4187      */

4189     argv[0] = "deallocate";
4190     argv[1] = "-Isz";
4191     argv[2] = zone_name;
4192     argv[3] = NULL;

4194     (void) forkexec(zlogp, "/usr/sbin/deallocate", argv);
4195     /* Don't check for errors since they don't affect the zone */
4196 }

4198 /*
4199  * Fetch the Trusted Extensions label and multi-level ports (MLPs) for
4200  * this zone.
4201  */
4202 static tsol_zcent_t *
4203 get_zone_label(zlog_t *zlogp, priv_set_t *privs)
4204 {
4205     FILE *fp;
4206     tsol_zcent_t *zcent = NULL;
4207     char line[MAXTNZLEN];

4209     if ((fp = fopen(TNZONECFG_PATH, "r")) == NULL) {
4210         zerror(zlogp, B_TRUE, "%s", TNZONECFG_PATH);
4211         return (NULL);
4212     }

4214     while (fgets(line, sizeof (line), fp) != NULL) {
4215         /*
4216          * Check for malformed database

```

```

4217     */
4218     if (strlen(line) == MAXTNZLEN - 1)
4219         break;
4220     if ((zcent = tsol_sgetzcent(line, NULL, NULL)) == NULL)
4221         continue;
4222     if (strcmp(zcent->zname, zone_name) == 0)
4223         break;
4224     tsol_freezcent(zcent);
4225     zcent = NULL;
4226 }
4227 (void) fclose(fp);

4229 if (zcent == NULL) {
4230     zerror(zlogp, B_FALSE, "zone requires a label assignment. "
4231         "See tnzonecfg(4)");
4232 } else {
4233     if (zlabel == NULL)
4234         zlabel = m_label_alloc(MAC_LABEL);
4235     /*
4236      * Save this zone's privileges for later read-down processing
4237      */
4238     if ((zprivs = priv_allocset()) == NULL) {
4239         zerror(zlogp, B_TRUE, "%s failed", "priv_allocset");
4240         return (NULL);
4241     } else {
4242         priv_copyset(privs, zprivs);
4243     }
4244 }
4245 return (zcent);
4246 }

4248 /*
4249  * Add the Trusted Extensions multi-level ports for this zone.
4250  */
4251 static void
4252 set_mlps(zlog_t *zlogp, zoneid_t zoneid, tsol_zcent_t *zcent)
4253 {
4254     tsol_mlp_t *mlp;
4255     tsol_mlpent_t tsme;

4257     if (!is_system_labeled())
4258         return;

4260     tsme.tsme_zoneid = zoneid;
4261     tsme.tsme_flags = 0;
4262     for (mlp = zcent->zprivate_mlp; !TSOL_MLP_END(mlp); mlp++) {
4263         tsme.tsme_mlp = *mlp;
4264         if (tnmlp(TNDB_LOAD, &tsme) != 0) {
4265             zerror(zlogp, B_TRUE, "cannot set zone-specific MLP "
4266                 "on %d-%d/%d", mlp->mlp_port,
4267                 mlp->mlp_port_upper, mlp->mlp_ipp);
4268         }
4269     }

4271     tsme.tsme_flags = TSOL_MEF_SHARED;
4272     for (mlp = zcent->zshared_mlp; !TSOL_MLP_END(mlp); mlp++) {
4273         tsme.tsme_mlp = *mlp;
4274         if (tnmlp(TNDB_LOAD, &tsme) != 0) {
4275             zerror(zlogp, B_TRUE, "cannot set shared MLP "
4276                 "on %d-%d/%d", mlp->mlp_port,
4277                 mlp->mlp_port_upper, mlp->mlp_ipp);
4278         }
4279     }
4280 }

4282 static void

```

```

4283 remove_mlps(zlog_t *zlogp, zoneid_t zoneid)
4284 {
4285     tsol_mlpent_t tsme;

4287     if (!is_system_labeled())
4288         return;

4290     (void) memset(&tsme, 0, sizeof (tsme));
4291     tsme.tsme_zoneid = zoneid;
4292     if (tnmlp(TNDB_FLUSH, &tsme) != 0)
4293         zerror(zlogp, B_TRUE, "cannot flush MLPs");
4294 }

4296 int
4297 prtmount(const struct mnttab *fs, void *x)
4298 {
4299     zerror((zlog_t *)x, B_FALSE, " %s", fs->mnt_mountp);
4300     return (0);
4301 }

4303 /*
4304  * Look for zones running on the main system that are using this root (or any
4305  * subdirectory of it). Return B_TRUE and print an error if a conflicting zone
4306  * is found or if we can't tell.
4307  */
4308 static boolean_t
4309 duplicate_zone_root(zlog_t *zlogp, const char *rootpath)
4310 {
4311     zoneid_t *zids = NULL;
4312     uint_t nzids = 0;
4313     boolean_t retv;
4314     int rlen, zlen;
4315     char zroot[MAXPATHLEN];
4316     char zonename[ZONENAME_MAX];

4318     for (;;) {
4319         nzids += 10;
4320         zids = malloc(nzids * sizeof (*zids));
4321         if (zids == NULL) {
4322             zerror(zlogp, B_TRUE, "memory allocation failed");
4323             return (B_TRUE);
4324         }
4325         if (zone_list(zids, &nzids) == 0)
4326             break;
4327         free(zids);
4328     }
4329     retv = B_FALSE;
4330     rlen = strlen(rootpath);
4331     while (nzids > 0) {
4332         /*
4333          * Ignore errors; they just mean that the zone has disappeared
4334          * while we were busy.
4335          */
4336         if (zone_getattr(zids[--nzids], ZONE_ATTR_ROOT, zroot,
4337             sizeof (zroot)) == -1)
4338             continue;
4339         zlen = strlen(zroot);
4340         if (zlen > rlen)
4341             zlen = rlen;
4342         if (strcmp(rootpath, zroot, zlen) == 0 &&
4343             (zroot[zlen] == '\\0' || zroot[zlen] == '/') &&
4344             (rootpath[zlen] == '\\0' || rootpath[zlen] == '/')) {
4345             if (getzonenamebyid(zids[nzids], zonename,
4346                 sizeof (zonename)) == -1)
4347                 (void) snprintf(zonename, sizeof (zonename),
4348                     "id %d", (int)zids[nzids]);

```

```

4349         zerror(zlogp, B_FALSE,
4350                "zone root %s already in use by zone %s",
4351                rootpath, zonename);
4352         retv = B_TRUE;
4353         break;
4354     }
4355 }
4356 free(zids);
4357 return (retv);
4358 }

4360 /*
4361  * Search for loopback mounts that use this same source node (same device and
4362  * inode). Return B_TRUE if there is one or if we can't tell.
4363  */
4364 static boolean_t
4365 duplicate_reachable_path(zlog_t *zlogp, const char *rootpath)
4366 {
4367     struct stat64 rst, zst;
4368     struct mnttab *mnp;

4370     if (stat64(rootpath, &rst) == -1) {
4371         zerror(zlogp, B_TRUE, "can't stat %s", rootpath);
4372         return (B_TRUE);
4373     }
4374     if (resolve_lofs_mnts == NULL && lofs_read_mnttab(zlogp) == -1)
4375         return (B_TRUE);
4376     for (mnp = resolve_lofs_mnts; mnp < resolve_lofs_mnt_max; mnp++) {
4377         if (mnp->mnt_fstype == NULL ||
4378             strcmp(MNTTYPE_LOFS, mnp->mnt_fstype) != 0)
4379             continue;
4380         /* We're looking at a loopback mount. Stat it. */
4381         if (mnp->mnt_special != NULL &&
4382             stat64(mnp->mnt_special, &zst) != -1 &&
4383             rst.st_dev == zst.st_dev && rst.st_ino == zst.st_ino) {
4384             zerror(zlogp, B_FALSE,
4385                    "zone root %s is reachable through %s",
4386                    rootpath, mnp->mnt_mountp);
4387             return (B_TRUE);
4388         }
4389     }
4390     return (B_FALSE);
4391 }

4393 /*
4394  * Set memory cap and pool info for the zone's resource management
4395  * configuration.
4396  */
4397 static int
4398 setup_zone_rm(zlog_t *zlogp, char *zone_name, zoneid_t zoneid)
4399 {
4400     int res;
4401     uint64_t tmp;
4402     struct zone_mcaptab mcap;
4403     char sched[MAXNAMELEN];
4404     zone_dochandle_t handle = NULL;
4405     char pool_err[128];

4407     if ((handle = zonecfg_init_handle()) == NULL) {
4408         zerror(zlogp, B_TRUE, "getting zone configuration handle");
4409         return (Z_BAD_HANDLE);
4410     }

4412     if ((res = zonecfg_get_snapshot_handle(zone_name, handle)) != Z_OK) {
4413         zerror(zlogp, B_FALSE, "invalid configuration");
4414         zonecfg_fini_handle(handle);

```

```

4415         return (res);
4416     }

4418     /*
4419     * If a memory cap is configured, set the cap in the kernel using
4420     * zone_setattr() and make sure the rcapd SMF service is enabled.
4421     */
4422     if (zonecfg_getmcapent(handle, &mcap) == Z_OK) {
4423         uint64_t num;
4424         char smf_err[128];

4426         num = (uint64_t)strtoull(mcap.zone_physmem_cap, NULL, 10);
4427         if (zone_setattr(zoneid, ZONE_ATTR_PHYS_MCAP, &num, 0) == -1) {
4428             zerror(zlogp, B_TRUE, "could not set zone memory cap");
4429             zonecfg_fini_handle(handle);
4430             return (Z_INVALID);
4431         }

4433         if (zonecfg_enable_rcapd(smf_err, sizeof (smf_err)) != Z_OK) {
4434             zerror(zlogp, B_FALSE, "enabling system/rcap service "
4435                    "failed: %s", smf_err);
4436             zonecfg_fini_handle(handle);
4437             return (Z_INVALID);
4438         }
4439     }

4441     /* Get the scheduling class set in the zone configuration. */
4442     if (zonecfg_get_sched_class(handle, sched, sizeof (sched)) == Z_OK &&
4443         strlen(sched) > 0) {
4444         if (zone_setattr(zoneid, ZONE_ATTR_SCHED_CLASS, sched,
4445             strlen(sched)) == -1)
4446             zerror(zlogp, B_TRUE, "WARNING: unable to set the "
4447                    "default scheduling class");
4449     } else if (zonecfg_get_aliased_rctl(handle, ALIAS_SHARES, &tmp)
4450         == Z_OK) {
4451         /*
4452         * If the zone has the zone.cpu-shares rctl set then we want to
4453         * use the Fair Share Scheduler (FSS) for processes in the
4454         * zone. Check what scheduling class the zone would be running
4455         * in by default so we can print a warning and modify the class
4456         * if we wouldn't be using FSS.
4457         */
4458         char class_name[PC_CLNMSZ];

4460         if (zonecfg_get_dflt_sched_class(handle, class_name,
4461             sizeof (class_name)) != Z_OK) {
4462             zerror(zlogp, B_FALSE, "WARNING: unable to determine "
4463                    "the zone's scheduling class");
4465         } else if (strcmp("FSS", class_name) != 0) {
4466             zerror(zlogp, B_FALSE, "WARNING: The zone.cpu-shares "
4467                    "rctl is set but\nFSS is not the default "
4468                    "scheduling class for\nthis zone. FSS will be "
4469                    "used for processes\nin the zone but to get the "
4470                    "full benefit of FSS,\nit should be the default "
4471                    "scheduling class.\nSee dispadmin(1M) for more "
4472                    "details.");
4474             if (zone_setattr(zoneid, ZONE_ATTR_SCHED_CLASS, "FSS",
4475                 strlen("FSS")) == -1)
4476                 zerror(zlogp, B_TRUE, "WARNING: unable to set "
4477                    "zone scheduling class to FSS");
4478         }
4479     }

```

```

4481  /*
4482  * The next few blocks of code attempt to set up temporary pools as
4483  * well as persistent pools. In all cases we call the functions
4484  * unconditionally. Within each function the code will check if the
4485  * zone is actually configured for a temporary pool or persistent pool
4486  * and just return if there is nothing to do.
4487  *
4488  * If we are rebooting we want to attempt to reuse any temporary pool
4489  * that was previously set up. zonecfg_bind_tmp_pool() will do the
4490  * right thing in all cases (reuse or create) based on the current
4491  * zonecfg.
4492  */
4493  if ((res = zonecfg_bind_tmp_pool(handle, zoneid, pool_err,
4494  sizeof (pool_err))) != Z_OK) {
4495  if (res == Z_POOL || res == Z_POOL_CREATE || res == Z_POOL_BIND)
4496  zerror(zlogp, B_FALSE, "%s: %s\ndedicated-cpu setting "
4497  "cannot be instantiated", zonecfg_strerror(res),
4498  pool_err);
4499  else
4500  zerror(zlogp, B_FALSE, "could not bind zone to "
4501  "temporary pool: %s", zonecfg_strerror(res));
4502  zonecfg_fini_handle(handle);
4503  return (Z_POOL_BIND);
4504  }

4506  /*
4507  * Check if we need to warn about poold not being enabled.
4508  */
4509  if (zonecfg_warn_poold(handle)) {
4510  zerror(zlogp, B_FALSE, "WARNING: A range of dedicated-cpus has "
4511  "been specified\nbut the dynamic pool service is not "
4512  "enabled.\nThe system will not dynamically adjust the\n"
4513  "processor allocation within the specified range\n"
4514  "until svc:/system/pools/dynamic is enabled.\n"
4515  "See poold(1M).");
4516  }

4518  /* The following is a warning, not an error. */
4519  if ((res = zonecfg_bind_pool(handle, zoneid, pool_err,
4520  sizeof (pool_err))) != Z_OK) {
4521  if (res == Z_POOL_BIND)
4522  zerror(zlogp, B_FALSE, "WARNING: unable to bind to "
4523  "pool '%s'; using default pool.", pool_err);
4524  else if (res == Z_POOL)
4525  zerror(zlogp, B_FALSE, "WARNING: %s: %s",
4526  zonecfg_strerror(res), pool_err);
4527  else
4528  zerror(zlogp, B_FALSE, "WARNING: %s",
4529  zonecfg_strerror(res));
4530  }

4532  /* Update saved pool name in case it has changed */
4533  (void) zonecfg_get_poolname(handle, zone_name, pool_name,
4534  sizeof (pool_name));

4536  zonecfg_fini_handle(handle);
4537  return (Z_OK);
4538  }

4540  static void
4541  report_prop_err(zlog_t *zlogp, const char *name, const char *value, int res)
4542  {
4543  switch (res) {
4544  case Z_TOO_BIG:
4545  zerror(zlogp, B_FALSE, "%s property value is too large.", name);
4546  break;

```

```

4548  case Z_INVALID_PROPERTY:
4549  zerror(zlogp, B_FALSE, "%s property value \"%s\" is not valid",
4550  name, value);
4551  break;

4553  default:
4554  zerror(zlogp, B_TRUE, "fetching property %s: %d", name, res);
4555  break;
4556  }
4557  }

4559  /*
4560  * Sets the hostid of the new zone based on its configured value. The zone's
4561  * zone_t structure must already exist in kernel memory. 'zlogp' refers to the
4562  * log used to report errors and warnings and must be non-NULL. 'zone_name'
4563  * is the name of the new zone and must be non-NULL. 'zoneid' is the numeric
4564  * ID of the new zone.
4565  *
4566  * This function returns zero on success and a nonzero error code on failure.
4567  */
4568  static int
4569  setup_zone_hostid(zone_dochandle_t handle, zlog_t *zlogp, zoneid_t zoneid)
4570  {
4571  int res;
4572  char hostidp[HW_HOSTID_LEN];
4573  unsigned int hostid;

4575  res = zonecfg_get_hostid(handle, hostidp, sizeof (hostidp));

4577  if (res == Z_BAD_PROPERTY) {
4578  return (Z_OK);
4579  } else if (res != Z_OK) {
4580  report_prop_err(zlogp, "hostid", hostidp, res);
4581  return (res);
4582  }

4584  hostid = (unsigned int)strtoul(hostidp, NULL, 16);
4585  if ((res = zone_setattr(zoneid, ZONE_ATTR_HOSTID, &hostid,
4586  sizeof (hostid))) != 0) {
4587  zerror(zlogp, B_TRUE,
4588  "zone hostid is not valid: %s: %d", hostidp, res);
4589  return (Z_SYSTEM);
4590  }

4592  return (res);
4593  }

4595  static int
4596  setup_zone_secflags(zone_dochandle_t handle, zlog_t *zlogp, zoneid_t zoneid)
4597  {
4598  psecflags_t secflags;
4599  struct zone_secflagstab tab = {0};
4600  secflagdelta_t delt;
4601  int res;

4603  res = zonecfg_lookup_secflags(handle, &tab);

4605  if ((res != Z_OK) &&
4606  /* The general defaulting code will handle this */
4607  (res != Z_NO_ENTRY) && (res != Z_BAD_PROPERTY)) {
4608  zerror(zlogp, B_FALSE, "security-flags property is "
4609  "invalid: %d", res);
4610  return (res);
4611  }

```

```

4613     if (strlen(tab.zone_secflags_lower) == 0)
4614         (void) strcpy(tab.zone_secflags_lower, "none",
4615             sizeof (tab.zone_secflags_lower));
4616     if (strlen(tab.zone_secflags_default) == 0)
4617         (void) strcpy(tab.zone_secflags_default,
4618             tab.zone_secflags_lower,
4619             sizeof (tab.zone_secflags_default));
4620     if (strlen(tab.zone_secflags_upper) == 0)
4621         (void) strcpy(tab.zone_secflags_upper, "all",
4622             sizeof (tab.zone_secflags_upper));

4624     if (secflags_parse(NULL, tab.zone_secflags_default,
4625         &delt) == -1) {
4626         zerror(zlogp, B_FALSE, "default security-flags: '%s'"
4627             "are invalid", tab.zone_secflags_default);
4628         return (Z_BAD_PROPERTY);
4629     } else if (delt.psd_ass_active != B_TRUE) {
4630         zerror(zlogp, B_FALSE, "relative security-flags are not "
4631             "allowed in zone configuration (default "
4632             "security-flags: '%s')",
4633             tab.zone_secflags_default);
4634         return (Z_BAD_PROPERTY);
4635     } else {
4636         secflags_copy(&secflags.psf_inherit, &delt.psd_assign);
4637         secflags_copy(&secflags.psf_effective, &delt.psd_assign);
4638     }

4640     if (secflags_parse(NULL, tab.zone_secflags_lower,
4641         &delt) == -1) {
4642         zerror(zlogp, B_FALSE, "lower security-flags: '%s'"
4643             "are invalid", tab.zone_secflags_lower);
4644         return (Z_BAD_PROPERTY);
4645     } else if (delt.psd_ass_active != B_TRUE) {
4646         zerror(zlogp, B_FALSE, "relative security-flags are not "
4647             "allowed in zone configuration (lower "
4648             "security-flags: '%s')",
4649             tab.zone_secflags_lower);
4650         return (Z_BAD_PROPERTY);
4651     } else {
4652         secflags_copy(&secflags.psf_lower, &delt.psd_assign);
4653     }

4655     if (secflags_parse(NULL, tab.zone_secflags_upper,
4656         &delt) == -1) {
4657         zerror(zlogp, B_FALSE, "upper security-flags: '%s'"
4658             "are invalid", tab.zone_secflags_upper);
4659         return (Z_BAD_PROPERTY);
4660     } else if (delt.psd_ass_active != B_TRUE) {
4661         zerror(zlogp, B_FALSE, "relative security-flags are not "
4662             "allowed in zone configuration (upper "
4663             "security-flags: '%s')",
4664             tab.zone_secflags_upper);
4665         return (Z_BAD_PROPERTY);
4666     } else {
4667         secflags_copy(&secflags.psf_upper, &delt.psd_assign);
4668     }

4670     if (!psecflags_validate(&secflags)) {
4671         zerror(zlogp, B_TRUE, "security-flags violate invariants");
4672         return (Z_BAD_PROPERTY);
4673     }

4675     if ((res = zone_setattr(zoneid, ZONE_ATTR_SECFLAGS, &secflags,
4676         sizeof (secflags))) != 0) {
4677         zerror(zlogp, B_TRUE,
4678             "security-flags couldn't be set: %d", res);

```

```

4679         return (Z_SYSTEM);
4680     }

4682     return (Z_OK);
4683 }

4685 static int
4686 #endif /* ! codereview */
4687 setup_zone_fs_allowed(zone_dochandle_t handle, zlog_t *zlogp, zoneid_t zoneid)
4688 {
4689     char fsallowed[ZONE_FS_ALLOWED_MAX];
4690     char *fsallowedp = fsallowed;
4691     int len = sizeof (fsallowed);
4692     int res;

4694     res = zonecfg_get_fs_allowed(handle, fsallowed, len);

4696     if (res == Z_BAD_PROPERTY) {
4697         /* No value, set the defaults */
4698         (void) strcpy(fsallowed, DFLT_FS_ALLOWED, len);
4699     } else if (res != Z_OK) {
4700         report_prop_err(zlogp, "fs-allowed", fsallowed, res);
4701         return (res);
4702     } else if (fsallowed[0] == '-') {
4703         /* dropping default filesystems - use remaining list */
4704         /* dropping default privs - use remaining list */
4705         if (fsallowed[1] != ',')
4706             return (Z_OK);
4707         fsallowedp += 2;
4708         len -= 2;
4709     } else {
4710         /* Has a value, append the defaults */
4711         if (strlencat(fsallowed, ",", len) >= len ||
4712             strlencat(fsallowed, DFLT_FS_ALLOWED, len) >= len) {
4713             report_prop_err(zlogp, "fs-allowed", fsallowed,
4714                 Z_TOO_BIG);
4715             return (Z_TOO_BIG);
4716         }
4717     }

4718     if (zone_setattr(zoneid, ZONE_ATTR_FS_ALLOWED, fsallowedp, len) != 0) {
4719         zerror(zlogp, B_TRUE,
4720             "fs-allowed couldn't be set: %s: %d", fsallowedp, res);
4721         return (Z_SYSTEM);
4722     }

4724     return (Z_OK);
4725 }

4727 static int
4728 setup_zone_attrs(zlog_t *zlogp, char *zone_namep, zoneid_t zoneid)
4729 {
4730     zone_dochandle_t handle;
4731     int res = Z_OK;

4733     if ((handle = zonecfg_init_handle()) == NULL) {
4734         zerror(zlogp, B_TRUE, "getting zone configuration handle");
4735         return (Z_BAD_HANDLE);
4736     }
4737     if ((res = zonecfg_get_snapshot_handle(zone_namep, handle)) != Z_OK) {
4738         zerror(zlogp, B_FALSE, "invalid configuration");
4739         goto out;
4740     }

4742     if ((res = setup_zone_hostid(handle, zlogp, zoneid)) != Z_OK)
4743         goto out;

```

```

4745     if ((res = setup_zone_fs_allowed(handle, zlogp, zoneid)) != Z_OK)
4746         goto out;
4748     if ((res = setup_zone_secflags(handle, zlogp, zoneid)) != Z_OK)
4749         goto out;
4751 #endif /* ! codereview */
4752 out:
4753     zonecfg_fini_handle(handle);
4754     return (res);
4755 }
4757 zoneid_t
4758 vplat_create(zlog_t *zlogp, zone_mnt_t mount_cmd)
4759 {
4760     zoneid_t rval = -1;
4761     priv_set_t *privs;
4762     char rootpath[MAXPATHLEN];
4763     char *rctlbuf = NULL;
4764     size_t rctlbufsz = 0;
4765     char *zfsbuf = NULL;
4766     size_t zfsbufsz = 0;
4767     zoneid_t zoneid = -1;
4768     int xerr;
4769     char *kzone;
4770     FILE *fp = NULL;
4771     tsol_zcent_t *zcent = NULL;
4772     int match = 0;
4773     int doi = 0;
4774     int flags;
4775     zone_iptype_t iptype;
4777     if (zone_get_rootpath(zone_name, rootpath, sizeof (rootpath)) != Z_OK) {
4778         zerror(zlogp, B_TRUE, "unable to determine zone root");
4779         return (-1);
4780     }
4781     if (zonecfg_in_alt_root())
4782         resolve_lofs(zlogp, rootpath, sizeof (rootpath));
4784     if (vplat_get_iptype(zlogp, &iptype) < 0) {
4785         zerror(zlogp, B_TRUE, "unable to determine ip-type");
4786         return (-1);
4787     }
4788     switch (iptype) {
4789     case ZS_SHARED:
4790         flags = 0;
4791         break;
4792     case ZS_EXCLUSIVE:
4793         flags = ZCF_NET_EXCL;
4794         break;
4795     }
4797     if ((privs = priv_allocset()) == NULL) {
4798         zerror(zlogp, B_TRUE, "%s failed", "priv_allocset");
4799         return (-1);
4800     }
4801     priv_emptyset(privs);
4802     if (get_privset(zlogp, privs, mount_cmd) != 0)
4803         goto error;
4805     if (mount_cmd == Z_MNT_BOOT &&
4806         get_rctls(zlogp, &rctlbuf, &rctlbufsz) != 0) {
4807         zerror(zlogp, B_FALSE, "Unable to get list of rctls");
4808         goto error;
4809     }

```

```

4811     if (get_datasets(zlogp, &zfsbuf, &zfsbufsz) != 0) {
4812         zerror(zlogp, B_FALSE, "Unable to get list of ZFS datasets");
4813         goto error;
4814     }
4816     if (mount_cmd == Z_MNT_BOOT && is_system_labeled()) {
4817         zcent = get_zone_label(zlogp, privs);
4818         if (zcent != NULL) {
4819             match = zcent->zcent->zmatch;
4820             doi = zcent->zcent->zdoi;
4821             *zlabel = zcent->zcent->zlabel;
4822         } else {
4823             goto error;
4824         }
4825         if (validate_rootds_label(zlogp, rootpath, zlabel) != 0)
4826             goto error;
4827     }
4829     kzone = zone_name;
4831     /*
4832     * We must do this scan twice. First, we look for zones running on the
4833     * main system that are using this root (or any subdirectory of it).
4834     * Next, we reduce to the shortest path and search for loopback mounts
4835     * that use this same source node (same device and inode).
4836     */
4837     if (duplicate_zone_root(zlogp, rootpath))
4838         goto error;
4839     if (duplicate_reachable_path(zlogp, rootpath))
4840         goto error;
4842     if (ALT_MOUNT(mount_cmd)) {
4843         root_to_lu(zlogp, rootpath, sizeof (rootpath), B_TRUE);
4845         /*
4846         * Forge up a special root for this zone. When a zone is
4847         * mounted, we can't let the zone have its own root because the
4848         * tools that will be used in this "scratch zone" need access
4849         * to both the zone's resources and the running machine's
4850         * executables.
4851         * Note that the mkdir here also catches read-only filesystems.
4852         */
4853         if (mkdir(rootpath, 0755) != 0 && errno != EEXIST) {
4854             zerror(zlogp, B_TRUE, "cannot create %s", rootpath);
4855             goto error;
4856         }
4857         if (domount(zlogp, "tmpfs", "", "swap", rootpath) != 0)
4858             goto error;
4859     }
4860 }
4862     if (zonecfg_in_alt_root()) {
4863         /*
4864         * If we are mounting up a zone in an alternate root partition,
4865         * then we have some additional work to do before starting the
4866         * zone. First, resolve the root path down so that we're not
4867         * fooled by duplicates. Then forge up an internal name for
4868         * the zone.
4869         */
4870         if ((fp = zonecfg_open_scratch("", B_TRUE)) == NULL) {
4871             zerror(zlogp, B_TRUE, "cannot open mapfile");
4872             goto error;
4873         }
4874         if (zonecfg_lock_scratch(fp) != 0) {
4875             zerror(zlogp, B_TRUE, "cannot lock mapfile");

```

```

4876         goto error;
4877     }
4878     if (zonecfg_find_scratch(fp, zone_name, zonecfg_get_root(),
4879         NULL, 0) == 0) {
4880         zerror(zlogp, B_FALSE, "scratch zone already running");
4881         goto error;
4882     }
4883     /* This is the preferred name */
4884     (void) snprintf(kernzone, sizeof (kernzone), "SUNWlu-%s",
4885         zone_name);
4886     srandom(getpid());
4887     while (zonecfg_reverse_scratch(fp, kernzone, NULL, 0, NULL,
4888         0) == 0) {
4889         /* This is just an arbitrary name; note "." usage */
4890         (void) snprintf(kernzone, sizeof (kernzone),
4891             "SUNWlu.%08lx%08lx", random(), random());
4892     }
4893     kzone = kernzone;
4894 }

4896 xerr = 0;
4897 if ((zoneid = zone_create(kzone, rootpath, privs, rctlbuf,
4898     rctlbufsz, zfsbuf, zfsbufsz, &xerr, match, doi, zlabel,
4899     flags)) == -1) {
4900     if (xerr == ZE_AREMOUNTS) {
4901         if (zonecfg_find_mounts(rootpath, NULL, NULL) < 1) {
4902             zerror(zlogp, B_FALSE,
4903                 "An unknown file-system is mounted on "
4904                 "a subdirectory of %s", rootpath);
4905         } else {
4906             zerror(zlogp, B_FALSE,
4907                 "These file-systems are mounted on "
4908                 "subdirectories of %s:", rootpath);
4909             (void) zonecfg_find_mounts(rootpath,
4910                 prtmount, zlogp);
4911         }
4912     } else if (xerr == ZE_CHROOTED) {
4913         zerror(zlogp, B_FALSE, "%s: "
4914             "cannot create a zone from a chrooted "
4915             "environment", "zone_create");
4916     } else if (xerr == ZE_LABELINUSE) {
4917         char zonename[ZONENAME_MAX];
4918         (void) getzonenamebyid(getzoneidbylabel(zlabel),
4919             zonename, ZONENAME_MAX);
4920         zerror(zlogp, B_FALSE, "The zone label is already "
4921             "used by the zone '%s'.", zonename);
4922     } else {
4923         zerror(zlogp, B_TRUE, "%s failed", "zone_create");
4924     }
4925     goto error;
4926 }
4927 }

4929 if (zonecfg_in_alt_root() &&
4930     zonecfg_add_scratch(fp, zone_name, kernzone,
4931     zonecfg_get_root()) == -1) {
4932     zerror(zlogp, B_TRUE, "cannot add mapfile entry");
4933     goto error;
4934 }

4936 /*
4937  * The following actions are not performed when merely mounting a zone
4938  * for administrative use.
4939  */
4940 if (mount_cmd == Z_MNT_BOOT) {
4941     brand_handle_t bh;

```

```

4942     struct brand_attr attr;
4943     char modname[MAXPATHLEN];

4945     if (setup_zone_attrs(zlogp, zone_name, zoneid) != Z_OK)
4946         goto error;

4948     if ((bh = brand_open(brand_name)) == NULL) {
4949         zerror(zlogp, B_FALSE,
4950             "unable to determine brand name");
4951         goto error;
4952     }

4954     if (!is_system_labeled() &&
4955         (strcmp(brand_name, LABELED_BRAND_NAME) == 0)) {
4956         brand_close(bh);
4957         zerror(zlogp, B_FALSE,
4958             "cannot boot labeled zone on unlabeled system");
4959         goto error;
4960     }

4962     /*
4963      * If this brand requires any kernel support, now is the time to
4964      * get it loaded and initialized.
4965      */
4966     if (brand_get_modname(bh, modname, MAXPATHLEN) < 0) {
4967         brand_close(bh);
4968         zerror(zlogp, B_FALSE,
4969             "unable to determine brand kernel module");
4970         goto error;
4971     }
4972     brand_close(bh);

4974     if (strlen(modname) > 0) {
4975         (void) strlcpy(attr.ba_brandname, brand_name,
4976             sizeof (attr.ba_brandname));
4977         (void) strlcpy(attr.ba_modname, modname,
4978             sizeof (attr.ba_modname));
4979         if (zone_setattr(zoneid, ZONE_ATTR_BRAND, &attr,
4980             sizeof (attr) != 0) {
4981             zerror(zlogp, B_TRUE,
4982                 "could not set zone brand attribute.");
4983             goto error;
4984         }
4985     }

4987     if (setup_zone_rm(zlogp, zone_name, zoneid) != Z_OK)
4988         goto error;

4990     set_mlps(zlogp, zoneid, zcent);
4991 }

4993     rval = zoneid;
4994     zoneid = -1;

4996 error:
4997     if (zoneid != -1) {
4998         (void) zone_shutdown(zoneid);
4999         (void) zone_destroy(zoneid);
5000     }
5001     if (rctlbuf != NULL)
5002         free(rctlbuf);
5003     priv_freeset(privs);
5004     if (fp != NULL)
5005         zonecfg_close_scratch(fp);
5006     lofs_discard_mnttab();
5007     if (zcent != NULL)

```



```

5140         0) {
5141             lofs_discard_mnttab();
5142             return (-1);
5143         }
5144         break;
5145     }
5146 }

5148 write_index_file(zoneid);

5150 lofs_discard_mnttab();
5151 return (0);
5152 }

5154 static int
5155 lu_root_takedown(zlog_t *zlogp)
5156 {
5157     char zroot[MAXPATHLEN];

5159     if (zone_get_rootpath(zone_name, zroot, sizeof (zroot)) != Z_OK) {
5160         zerror(zlogp, B_FALSE, "unable to determine zone root");
5161         return (-1);
5162     }
5163     root_to_lu(zlogp, zroot, sizeof (zroot), B_FALSE);

5165     /*
5166      * At this point, the processes are gone, the filesystems (save the
5167      * root) are unmounted, and the zone is on death row. But there may
5168      * still be creds floating about in the system that reference the
5169      * zone_t, and which pin down zone_rootvp causing this call to fail
5170      * with EBUSY. Thus, we try for a little while before just giving up.
5171      * (How I wish this were not true, and umount2 just did the right
5172      * thing, or tmpfs supported MS_FORCE This is a gross hack.)
5173      */
5174     if (umount2(zroot, MS_FORCE) != 0) {
5175         if (errno == ENOTSUP && umount2(zroot, 0) == 0)
5176             goto unmounted;
5177         if (errno == EBUSY) {
5178             int tries = 10;

5180             while (--tries >= 0) {
5181                 (void) sleep(1);
5182                 if (umount2(zroot, 0) == 0)
5183                     goto unmounted;
5184                 if (errno != EBUSY)
5185                     break;
5186             }
5187         }
5188         zerror(zlogp, B_TRUE, "unable to unmount '%s'", zroot);
5189         return (-1);
5190     }
5191     unmounted:

5193     /*
5194      * Only zones in an alternate root environment have scratch zone
5195      * entries.
5196      */
5197     if (zonecfg_in_alt_root()) {
5198         FILE *fp;
5199         int retv;

5201         if ((fp = zonecfg_open_scratch("", B_FALSE)) == NULL) {
5202             zerror(zlogp, B_TRUE, "cannot open mapfile");
5203             return (-1);
5204         }
5205         retv = -1;

```

```

5206         if (zonecfg_lock_scratch(fp) != 0)
5207             zerror(zlogp, B_TRUE, "cannot lock mapfile");
5208         else if (zonecfg_delete_scratch(fp, kernzone) != 0)
5209             zerror(zlogp, B_TRUE, "cannot delete map entry");
5210         else
5211             retv = 0;
5212         zonecfg_close_scratch(fp);
5213         return (retv);
5214     } else {
5215         return (0);
5216     }
5217 }

5219 int
5220 vplat_takedown(zlog_t *zlogp, boolean_t unmount_cmd, boolean_t rebooting)
5221 {
5222     char *kzone;
5223     zoneid_t zoneid;
5224     int res;
5225     char pool_err[128];
5226     char zpath[MAXPATHLEN];
5227     char cmdbuf[MAXPATHLEN];
5228     brand_handle_t bh = NULL;
5229     dladm_status_t status;
5230     char errmsg[DLADM_STRSIZE];
5231     ushort_t flags;

5233     kzone = zone_name;
5234     if (zonecfg_in_alt_root()) {
5235         FILE *fp;

5237         if ((fp = zonecfg_open_scratch("", B_FALSE)) == NULL) {
5238             zerror(zlogp, B_TRUE, "unable to open map file");
5239             goto error;
5240         }
5241         if (zonecfg_find_scratch(fp, zone_name, zonecfg_get_root(),
5242             kernzone, sizeof (kernzone)) != 0) {
5243             zerror(zlogp, B_FALSE, "unable to find scratch zone");
5244             zonecfg_close_scratch(fp);
5245             goto error;
5246         }
5247         zonecfg_close_scratch(fp);
5248         kzone = kernzone;
5249     }

5251     if ((zoneid = getzoneidbyname(kzone)) == ZONE_ID_UNDEFINED) {
5252         if (!bringup_failure_recovery)
5253             zerror(zlogp, B_TRUE, "unable to get zoneid");
5254         if (unmount_cmd)
5255             (void) lu_root_takedown(zlogp);
5256         goto error;
5257     }

5259     if (remove_datalink_pool(zlogp, zoneid) != 0) {
5260         zerror(zlogp, B_FALSE, "unable clear datalink pool property");
5261         goto error;
5262     }

5264     if (remove_datalink_protect(zlogp, zoneid) != 0) {
5265         zerror(zlogp, B_FALSE,
5266             "unable clear datalink protect property");
5267         goto error;
5268     }

5270     /*
5271      * The datalinks assigned to the zone will be removed from the NGZ as

```

```

5272  * part of zone_shutdown() so that we need to remove protect/pool etc.
5273  * before zone_shutdown(). Even if the shutdown itself fails, the zone
5274  * will not be able to violate any constraints applied because the
5275  * datalinks are no longer available to the zone.
5276  */
5277  if (zone_shutdown(zoneid) != 0) {
5278      zerror(zlogp, B_TRUE, "unable to shutdown zone");
5279      goto error;
5280  }

5282  /* Get the zonepath of this zone */
5283  if (zone_get_zonepath(zone_name, zpath, sizeof (zpath)) != Z_OK) {
5284      zerror(zlogp, B_FALSE, "unable to determine zone path");
5285      goto error;
5286  }

5288  /* Get a handle to the brand info for this zone */
5289  if ((bh = brand_open(brand_name)) == NULL) {
5290      zerror(zlogp, B_FALSE, "unable to determine zone brand");
5291      return (-1);
5292  }
5293  /*
5294  * If there is a brand 'halt' callback, execute it now to give the
5295  * brand a chance to cleanup any custom configuration.
5296  */
5297  (void) strcpy(cmdbuf, EXEC_PREFIX);
5298  if (brand_get_halt(bh, zone_name, zpath, cmdbuf + EXEC_LEN,
5299      sizeof (cmdbuf) - EXEC_LEN) < 0) {
5300      brand_close(bh);
5301      zerror(zlogp, B_FALSE, "unable to determine branded zone's "
5302          "halt callback.");
5303      goto error;
5304  }
5305  brand_close(bh);

5307  if ((strlen(cmdbuf) > EXEC_LEN) &&
5308      (do_subproc(zlogp, cmdbuf, NULL) != Z_OK)) {
5309      zerror(zlogp, B_FALSE, "%s failed", cmdbuf);
5310      goto error;
5311  }

5313  if (!unmount_cmd) {
5314      zone_iptype_t iptype;

5316      if (zone_getattr(zoneid, ZONE_ATTR_FLAGS, &flags,
5317          sizeof (flags)) < 0) {
5318          if (vplat_get_iptype(zlogp, &iptype) < 0) {
5319              zerror(zlogp, B_TRUE, "unable to determine "
5320                  "ip-type");
5321              goto error;
5322          }
5323      } else {
5324          if (flags & ZF_NET_EXCL)
5325              iptype = ZS_EXCLUSIVE;
5326          else
5327              iptype = ZS_SHARED;
5328      }

5330      switch (iptype) {
5331      case ZS_SHARED:
5332          if (unconfigure_shared_network_interfaces(zlogp,
5333              zoneid) != 0) {
5334              zerror(zlogp, B_FALSE, "unable to unconfigure "
5335                  "network interfaces in zone");
5336              goto error;
5337          }

```

```

5338          break;
5339      case ZS_EXCLUSIVE:
5340          if (unconfigure_exclusive_network_interfaces(zlogp,
5341              zoneid) != 0) {
5342              zerror(zlogp, B_FALSE, "unable to unconfigure "
5343                  "network interfaces in zone");
5344              goto error;
5345          }
5346          status = dladm_zone_halt(dld_handle, zoneid);
5347          if (status != DLADM_STATUS_OK) {
5348              zerror(zlogp, B_FALSE, "unable to notify "
5349                  "dlmgmt of zone halt: %s",
5350                  dladm_status2str(status, errmsg));
5351          }
5352          break;
5353      }
5354  }

5356  if (!unmount_cmd && tcp_abort_connections(zlogp, zoneid) != 0) {
5357      zerror(zlogp, B_TRUE, "unable to abort TCP connections");
5358      goto error;
5359  }

5361  if (unmount_filesystems(zlogp, zoneid, unmount_cmd) != 0) {
5362      zerror(zlogp, B_FALSE,
5363          "unable to unmount file systems in zone");
5364      goto error;
5365  }

5367  /*
5368  * If we are rebooting then we normally don't want to destroy an
5369  * existing temporary pool at this point so that we can just reuse it
5370  * when the zone boots back up. However, it is also possible we were
5371  * running with a temporary pool and the zone configuration has been
5372  * modified to no longer use a temporary pool. In that case we need
5373  * to destroy the temporary pool now. This case looks like the case
5374  * where we never had a temporary pool configured but
5375  * zonecfg_destroy_tmp_pool will do the right thing either way.
5376  */
5377  if (!unmount_cmd) {
5378      boolean_t destroy_tmp_pool = B_TRUE;

5380      if (rebooting) {
5381          struct zone_psettab pset_tab;
5382          zone_dochandle_t handle;

5384          if ((handle = zonecfg_init_handle()) != NULL &&
5385              zonecfg_get_handle(zone_name, handle) == Z_OK &&
5386              zonecfg_lookup_pset(handle, &pset_tab) == Z_OK)
5387              destroy_tmp_pool = B_FALSE;

5389          zonecfg_fini_handle(handle);
5390      }

5392      if (destroy_tmp_pool) {
5393          if ((res = zonecfg_destroy_tmp_pool(zone_name, pool_err,
5394              sizeof (pool_err))) != Z_OK) {
5395              if (res == Z_POOL)
5396                  zerror(zlogp, B_FALSE, pool_err);
5397          }
5398      }
5399  }

5401  remove_mlps(zlogp, zoneid);
5403  if (zone_destroy(zoneid) != 0) {

```

```
5404         zerror(zlogp, B_TRUE, "unable to destroy zone");
5405         goto error;
5406     }
5407
5408     /*
5409     * Special teardown for alternate boot environments: remove the tmpfs
5410     * root for the zone and then remove it from the map file.
5411     */
5412     if (unmount_cmd && lu_root_teardown(zlogp) != 0)
5413         goto error;
5414
5415     lofs_discard_mnttab();
5416     return (0);
5417
5418 error:
5419     lofs_discard_mnttab();
5420     return (-1);
5421 }
```

```

*****
201398 Thu Jun 30 22:01:50 2016
new/usr/src/cmd/zonecfg/zonecfg.c
Code review comments from pmooney (sundry), and igork (screwups in zonecfg refac
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */

22 /*
23  * Copyright (c) 2003, 2010, Oracle and/or its affiliates. All rights reserved.
24  * Copyright 2014 Nexenta Systems, Inc. All rights reserved.
25  * Copyright 2014 Gary Mills
26 */

28 /*
29  * zonecfg is a lex/yacc based command interpreter used to manage zone
30  * configurations. The lexer (see zonecfg_lex.l) builds up tokens, which
31  * the grammar (see zonecfg_grammar.y) builds up into commands, some of
32  * which takes resources and/or properties as arguments. See the block
33  * comments near the end of zonecfg_grammar.y for how the data structures
34  * which keep track of these resources and properties are built up.
35  *
36  * The resource/property data structures are inserted into a command
37  * structure (see zonecfg.h), which also keeps track of command names,
38  * miscellaneous arguments, and function handlers. The grammar selects
39  * the appropriate function handler, each of which takes a pointer to a
40  * command structure as its sole argument, and invokes it. The grammar
41  * itself is "entered" (a la the Matrix) by yyparse(), which is called
42  * from read_input(), our main driving function. That in turn is called
43  * by one of do_interactive(), cmd_file() or one_command_at_a_time(), each
44  * of which is called from main() depending on how the program was invoked.
45  *
46  * The rest of this module consists of the various function handlers and
47  * their helper functions. Some of these functions, particularly the
48  * X_to_str() functions, which maps command, resource and property numbers
49  * to strings, are used quite liberally, as doing so results in a better
50  * program w/rt I18N, reducing the need for translation notes.
51 */

53 #include <sys/mntent.h>
54 #include <sys/varargs.h>
55 #include <sys/sysmacros.h>
56 #include <sys/secflags.h>
57 #endif /* ! codereview */

```

```

59 #include <errno.h>
60 #include <fcntl.h>
61 #include <strings.h>
62 #include <unistd.h>
63 #include <ctype.h>
64 #include <stdlib.h>
65 #include <assert.h>
66 #include <sys/stat.h>
67 #include <zone.h>
68 #include <arpa/inet.h>
69 #include <netdb.h>
70 #include <locale.h>
71 #include <libintl.h>
72 #include <alloca.h>
73 #include <signal.h>
74 #include <wait.h>
75 #include <libtecla.h>
76 #include <libzfs.h>
77 #include <sys/brand.h>
78 #include <libbrand.h>
79 #include <sys/systeminfo.h>
80 #include <libldadm.h>
81 #include <libinetutil.h>
82 #include <pwd.h>
83 #include <inet/ip.h>

85 #include <libzonecfg.h>
86 #include "zonecfg.h"

88 #if !defined(TEXT_DOMAIN) /* should be defined by cc -D */
89 #define TEXT_DOMAIN "SYS_TEST" /* Use this only if it wasn't */
90 #endif

92 #define PAGER "/usr/bin/more"
93 #define EXEC_PREFIX "exec "
94 #define EXEC_LEN (strlen(EXEC_PREFIX))

96 struct help {
97     uint_t cmd_num;
98     char *cmd_name;
99     uint_t flags;
100    char *short_usage;
101 };

103 extern int yyparse(void);
104 extern int lex_lineno;

106 #define MAX_LINE_LEN 1024
107 #define MAX_CMD_HIST 1024
108 #define MAX_CMD_LEN 1024

110 #define ONE_MB 1048576

112 /*
113  * Each SHELP_ should be a simple string.
114 */

116 #define SHELP_ADD "add <resource-type>\n\t(global scope)\n" \
117 "add <property-name> <property-value>\n\t(resource scope)"
118 #define SHELP_CANCEL "cancel"
119 #define SHELP_CLEAR "clear <property-name>"
120 #define SHELP_COMMIT "commit"
121 #define SHELP_CREATE "create [-F] [ -a <path> | -b | -t <template> ]"
122 #define SHELP_DELETE "delete [-F]"
123 #define SHELP_END "end"

```

```

124 #define SHELP_EXIT      "exit [-F]"
125 #define SHELP_EXPORT    "export [-f output-file]"
126 #define SHELP_HELP      "help [commands] [syntax] [usage] [<command-name>]"
127 #define SHELP_INFO      "info [<resource-type> [property-name=property-value]*]"
128 #define SHELP_REMOVE    "remove [-F] <resource-type> " \
129 "    [" <property-name>=<property-value> ]*\n" \
130 "\t(global scope)\n" \
131 "remove <property-name> <property-value>\n" \
132 "\t(resource scope)"
133 #define SHELP_REVERT    "revert [-F]"
134 #define SHELP_SELECT    "select <resource-type> { <property-name>=" \
135 "<property-value> }]"
136 #define SHELP_SET       "set <property-name>=<property-value>"
137 #define SHELP_VERIFY    "verify"

```

```

139 static struct help helptab[] = {
140     { CMD_ADD,      "add",      HELP_RES_PROPS, SHELP_ADD, },
141     { CMD_CANCEL,  "cancel",    0,             SHELP_CANCEL, },
142     { CMD_CLEAR,   "clear",     HELP_PROPS,    SHELP_CLEAR, },
143     { CMD_COMMIT,  "commit",    0,             SHELP_COMMIT, },
144     { CMD_CREATE,  "create",     0,             SHELP_CREATE, },
145     { CMD_DELETE,  "delete",    0,             SHELP_DELETE, },
146     { CMD_END,     "end",        0,             SHELP_END, },
147     { CMD_EXIT,    "exit",       0,             SHELP_EXIT, },
148     { CMD_EXPORT,  "export",    0,             SHELP_EXPORT, },
149     { CMD_HELP,    "help",       0,             SHELP_HELP, },
150     { CMD_INFO,    "info",       HELP_RES_PROPS, SHELP_INFO, },
151     { CMD_REMOVE,  "remove",     HELP_RES_PROPS, SHELP_REMOVE, },
152     { CMD_REVERT,  "revert",     0,             SHELP_REVERT, },
153     { CMD_SELECT,  "select",     HELP_RES_PROPS, SHELP_SELECT, },
154     { CMD_SET,     "set",        HELP_PROPS,    SHELP_SET, },
155     { CMD_VERIFY,  "verify",     0,             SHELP_VERIFY, },
156     { 0,           0,            0,             0, },
157 };

```

```

159 #define MAX_RT_STRLEN 16

```

```

161 /* These *must* match the order of the RT_define's from zoncfg.h */
162 char *res_types[] = {
163     "unknown",
164     "zonename",
165     "zonename",
166     "zonename",
167     "pool",
168     "fs",
169     "net",
170     "device",
171     "rctl",
172     "attr",
173     "dataset",
174     "limitpriv",
175     "bootargs",
176     "brand",
177     "dedicated-cpu",
178     "capped-memory",
179     ALIAS_MAXLWPS,
180     ALIAS_MAXSHMEM,
181     ALIAS_MAXSHMIDS,
182     ALIAS_MAXMSGIDS,
183     ALIAS_MAXSEMIDS,
184     ALIAS_SHARES,
185     "scheduling-class",
186     "ip-type",
187     "capped-cpu",
188     "hostid",
189     "admin",

```

```

190     "fs-allowed",
191     ALIAS_MAXPROCS,
192     "security-flags",
193 #endif /* ! codereview */
194     NULL
195 };

```

```

197 /* These *must* match the order of the PT_define's from zoncfg.h */

```

```

198 char *prop_types[] = {
199     "unknown",
200     "zonename",
201     "zonename",
202     "zonename",
203     "autoboot",
204     "pool",
205     "dir",
206     "special",
207     "type",
208     "options",
209     "address",
210     "physical",
211     "name",
212     "value",
213     "match",
214     "priv",
215     "limit",
216     "action",
217     "raw",
218     "limitpriv",
219     "bootargs",
220     "brand",
221     "ncpus",
222     "importance",
223     "swap",
224     "locked",
225     ALIAS_SHARES,
226     ALIAS_MAXLWPS,
227     ALIAS_MAXSHMEM,
228     ALIAS_MAXSHMIDS,
229     ALIAS_MAXMSGIDS,
230     ALIAS_MAXSEMIDS,
231     ALIAS_MAXLOCKEDMEM,
232     ALIAS_MAXSWAP,
233     "scheduling-class",
234     "ip-type",
235     "defrouter",
236     "hostid",
237     "user",
238     "auths",
239     "fs-allowed",
240     ALIAS_MAXPROCS,
241     "allowed-address",
242     "default",
243     "lower",
244     "upper",
245 #endif /* ! codereview */
246 };

```

```

248 /* These *must* match the order of the PROP_VAL_define's from zoncfg.h */

```

```

249 static char *prop_val_types[] = {
250     "simple",
251     "complex",
252     "list",
253 };
254
255 /*

```

```

256 * The various _cmds[] lists below are for command tab-completion.
257 */
259 /*
260 * remove has a space afterwards because it has qualifiers; the other commands
261 * that have qualifiers (add, select, etc.) don't need a space here because
262 * they have their own _cmds[] lists below.
263 */
264 static const char *global_scope_cmds[] = {
265     "add",
266     "clear",
267     "commit",
268     "create",
269     "delete",
270     "exit",
271     "export",
272     "help",
273     "info",
274     "remove ",
275     "revert",
276     "select",
277     "set",
278     "verify",
279     NULL
280 };
282 static const char *add_cmds[] = {
283     "add fs",
284     "add net",
285     "add device",
286     "add rctl",
287     "add attr",
288     "add dataset",
289     "add dedicated-cpu",
290     "add capped-cpu",
291     "add capped-memory",
292     "add admin",
293     "add security-flags",
294 #endif /* ! codereview */
295     NULL
296 };
298 static const char *clear_cmds[] = {
299     "clear autoboot",
300     "clear pool",
301     "clear limitpriv",
302     "clear bootargs",
303     "clear scheduling-class",
304     "clear ip-type",
305     "clear " ALIAS_MAXLWPS,
306     "clear " ALIAS_MAXSHMMEM,
307     "clear " ALIAS_MAXSHMIDS,
308     "clear " ALIAS_MAXMSGIDS,
309     "clear " ALIAS_MAXSEMIDS,
310     "clear " ALIAS_SHARES,
311     "clear " ALIAS_MAXPROCS,
312     NULL
313 };
315 static const char *remove_cmds[] = {
316     "remove fs ",
317     "remove net ",
318     "remove device ",
319     "remove rctl ",
320     "remove attr ",
321     "remove dataset ",

```

```

322     "remove dedicated-cpu ",
323     "remove capped-cpu ",
324     "remove capped-memory ",
325     "remove admin ",
326     "remove security-flags",
327 #endif /* ! codereview */
328     NULL
329 };
331 static const char *select_cmds[] = {
332     "select fs ",
333     "select net ",
334     "select device ",
335     "select rctl ",
336     "select attr ",
337     "select dataset ",
338     "select dedicated-cpu",
339     "select capped-cpu",
340     "select capped-memory",
341     "select admin",
342     "select security-flags",
343 #endif /* ! codereview */
344     NULL
345 };
347 static const char *set_cmds[] = {
348     "set zonename=",
349     "set zonepath=",
350     "set brand=",
351     "set autoboot=",
352     "set pool=",
353     "set limitpriv=",
354     "set bootargs=",
355     "set scheduling-class=",
356     "set ip-type=",
357     "set " ALIAS_MAXLWPS "=",
358     "set " ALIAS_MAXSHMMEM "=",
359     "set " ALIAS_MAXSHMIDS "=",
360     "set " ALIAS_MAXMSGIDS "=",
361     "set " ALIAS_MAXSEMIDS "=",
362     "set " ALIAS_SHARES "=",
363     "set hostid=",
364     "set fs-allowed=",
365     "set " ALIAS_MAXPROCS "=",
366     NULL
367 };
369 static const char *info_cmds[] = {
370     "info fs ",
371     "info net ",
372     "info device ",
373     "info rctl ",
374     "info attr ",
375     "info dataset ",
376     "info capped-memory",
377     "info dedicated-cpu",
378     "info capped-cpu",
379     "info security-flags",
380 #endif /* ! codereview */
381     "info zonename",
382     "info zonepath",
383     "info autoboot",
384     "info pool",
385     "info limitpriv",
386     "info bootargs",
387     "info brand",

```

```

388     "info scheduling-class",
389     "info ip-type",
390     "info max-lwps",
391     "info max-shm-memory",
392     "info max-shm-ids",
393     "info max-msg-ids",
394     "info max-sem-ids",
395     "info cpu-shares",
396     "info hostid",
397     "info admin",
398     "info fs-allowed",
399     "info max-processes",
400     NULL
401 };

403 static const char *fs_res_scope_cmds[] = {
404     "add options ",
405     "cancel",
406     "end",
407     "exit",
408     "help",
409     "info",
410     "remove options ",
411     "set dir=",
412     "set raw=",
413     "set special=",
414     "set type=",
415     "clear raw",
416     NULL
417 };

419 static const char *net_res_scope_cmds[] = {
420     "cancel",
421     "end",
422     "exit",
423     "help",
424     "info",
425     "set address=",
426     "set physical=",
427     "set defrouter=",
428     NULL
429 };

431 static const char *device_res_scope_cmds[] = {
432     "cancel",
433     "end",
434     "exit",
435     "help",
436     "info",
437     "set match=",
438     NULL
439 };

441 static const char *attr_res_scope_cmds[] = {
442     "cancel",
443     "end",
444     "exit",
445     "help",
446     "info",
447     "set name=",
448     "set type=",
449     "set value=",
450     NULL
451 };

453 static const char *rctl_res_scope_cmds[] = {

```

```

454     "add value ",
455     "cancel",
456     "end",
457     "exit",
458     "help",
459     "info",
460     "remove value ",
461     "set name=",
462     NULL
463 };

465 static const char *dataset_res_scope_cmds[] = {
466     "cancel",
467     "end",
468     "exit",
469     "help",
470     "info",
471     "set name=",
472     NULL
473 };

475 static const char *pset_res_scope_cmds[] = {
476     "cancel",
477     "end",
478     "exit",
479     "help",
480     "info",
481     "set ncpus=",
482     "set importance=",
483     "clear importance",
484     NULL
485 };

487 static const char *pcap_res_scope_cmds[] = {
488     "cancel",
489     "end",
490     "exit",
491     "help",
492     "info",
493     "set ncpus=",
494     NULL
495 };

497 static const char *mcap_res_scope_cmds[] = {
498     "cancel",
499     "end",
500     "exit",
501     "help",
502     "info",
503     "set physical=",
504     "set swap=",
505     "set locked=",
506     "clear physical",
507     "clear swap",
508     "clear locked",
509     NULL
510 };

512 static const char *admin_res_scope_cmds[] = {
513     "cancel",
514     "end",
515     "exit",
516     "help",
517     "info",
518     "set user=",
519     "set auths=",

```



```

520     NULL
521 };

523 static const char *secflags_res_scope_cmds[] = {
524     "cancel",
525     "end",
526     "exit",
527     "set default=",
528     "set lower=",
529     "set upper=",
530     NULL
531 };

533 #endif /* ! codereview */
534 struct xif {
535     struct xif      *xif_next;
536     char            xif_name[LIFNAMSIZ];
537     boolean_t      xif_has_address;
538     boolean_t      xif_has_defrouter;
539 };

541 /* Global variables */

543 /* list of network interfaces specified for exclusive IP zone */
544 struct xif *xif;

546 /* set early in main(), never modified thereafter, used all over the place */
547 static char *execname;

549 /* set in main(), used all over the place */
550 static zone_dochandle_t handle;

552 /* used all over the place */
553 static char zone[ZONENAME_MAX];
554 static char revert_zone[ZONENAME_MAX];

556 /* global brand operations */
557 static brand_handle_t brand;

559 /* set in modifying functions, checked in read_input() */
560 static boolean_t need_to_commit = B_FALSE;
561 boolean_t saw_error;

563 /* set in yacc parser, checked in read_input() */
564 boolean_t newline_terminated;

566 /* set in main(), checked in lex error handler */
567 boolean_t cmd_file_mode;

569 /* set in exit_func(), checked in read_input() */
570 static boolean_t time_to_exit = B_FALSE, force_exit = B_FALSE;

572 /* used in short_usage() and zerr() */
573 static char *cmd_file_name = NULL;

575 /* checked in read_input() and other places */
576 static boolean_t ok_to_prompt = B_FALSE;

578 /* set and checked in initialize() */
579 static boolean_t got_handle = B_FALSE;

581 /* initialized in do_interactive(), checked in initialize() */
582 static boolean_t interactive_mode;

584 /* set if configuring the global zone */
585 static boolean_t global_zone = B_FALSE;

```

```

587 /* set in main(), checked in multiple places */
588 static boolean_t read_only_mode;

590 /* scope is outer/global or inner/resource */
591 static boolean_t global_scope = B_TRUE;
592 static int resource_scope; /* should be in the RT_list from zoncfg.h */
593 static int end_op = -1; /* operation on end is either add or modify */

595 int num_prop_vals; /* for grammar */

597 /*
598  * These are for keeping track of resources as they are specified as part of
599  * the multi-step process. They should be initialized by add_resource() or
600  * select_func() and filled in by add_property() or set_func().
601  */
602 static struct zone_fstab      old_fstab, in_progress_fstab;
603 static struct zone_nwifstab   old_nwifstab, in_progress_nwifstab;
604 static struct zone_devtab     old_devtab, in_progress_devtab;
605 static struct zone_rctltab    old_rctltab, in_progress_rctltab;
606 static struct zone_attrtab    old_attrtab, in_progress_attrtab;
607 static struct zone_dstab     old_dstab, in_progress_dstab;
608 static struct zone_psettab    old_psettab, in_progress_psettab;
609 static struct zone_mcaptab    old_mcaptab, in_progress_mcaptab;
610 static struct zone_admintab   old_admintab, in_progress_admintab;
611 static struct zone_secflagstab old_secflagstab, in_progress_secflagstab;
612 #endif /* ! codereview */

614 static GetLine *gl; /* The gl_get_line() resource object */

616 static void bytes_to_units(char *str, char *buf, int bufsize);

618 /* Functions begin here */

620 static boolean_t
621 initial_match(const char *line1, const char *line2, int word_end)
622 {
623     if (word_end <= 0)
624         return (B_TRUE);
625     return (strncmp(line1, line2, word_end) == 0);
626 }

628 static int
629 add_stuff(WordCompletion *cpl, const char *line1, const char **list,
630          int word_end)
631 {
632     int i, err;

634     for (i = 0; list[i] != NULL; i++) {
635         if (initial_match(line1, list[i], word_end)) {
636             err = cpl_add_completion(cpl, line1, 0, word_end,
637                                   list[i] + word_end, "", "");
638             if (err != 0)
639                 return (err);
640         }
641     }
642     return (0);
643 }

645 static
646 /* ARGSUSED */
647 CPL_MATCH_FN(cmd_cpl_fn)
648 {
649     if (global_scope) {
650         /*
651          * The MAX/MIN tests below are to make sure we have at least

```

```

652     * enough characters to distinguish from other prefixes (MAX)
653     * but only check MIN(what we have, what we're checking).
654     */
655     if (strncmp(line, "add ", MAX(MIN(word_end, 4), 1)) == 0)
656         return (add_stuff(cpl, line, add_cmds, word_end));
657     if (strncmp(line, "clear ", MAX(MIN(word_end, 6), 2)) == 0)
658         return (add_stuff(cpl, line, clear_cmds, word_end));
659     if (strncmp(line, "select ", MAX(MIN(word_end, 7), 3)) == 0)
660         return (add_stuff(cpl, line, select_cmds, word_end));
661     if (strncmp(line, "set ", MAX(MIN(word_end, 4), 3)) == 0)
662         return (add_stuff(cpl, line, set_cmds, word_end));
663     if (strncmp(line, "remove ", MAX(MIN(word_end, 7), 1)) == 0)
664         return (add_stuff(cpl, line, remove_cmds, word_end));
665     if (strncmp(line, "info ", MAX(MIN(word_end, 5), 1)) == 0)
666         return (add_stuff(cpl, line, info_cmds, word_end));
667     return (add_stuff(cpl, line, global_scope_cmds, word_end));
668 }
669 switch (resource_scope) {
670 case RT_FS:
671     return (add_stuff(cpl, line, fs_res_scope_cmds, word_end));
672 case RT_NET:
673     return (add_stuff(cpl, line, net_res_scope_cmds, word_end));
674 case RT_DEVICE:
675     return (add_stuff(cpl, line, device_res_scope_cmds, word_end));
676 case RT_RCTL:
677     return (add_stuff(cpl, line, rctl_res_scope_cmds, word_end));
678 case RT_ATTR:
679     return (add_stuff(cpl, line, attr_res_scope_cmds, word_end));
680 case RT_DATASET:
681     return (add_stuff(cpl, line, dataset_res_scope_cmds, word_end));
682 case RT_DCPU:
683     return (add_stuff(cpl, line, pset_res_scope_cmds, word_end));
684 case RT_PCAP:
685     return (add_stuff(cpl, line, pcap_res_scope_cmds, word_end));
686 case RT_MCAP:
687     return (add_stuff(cpl, line, mcap_res_scope_cmds, word_end));
688 case RT_ADMIN:
689     return (add_stuff(cpl, line, admin_res_scope_cmds, word_end));
690 case RT_SECFLAGS:
691     return (add_stuff(cpl, line, secflags_res_scope_cmds,
692         word_end));
693 }
694 #endif /* ! codereview */
695 }
696 return (0);
697 }
698
699 /*
700  * For the main CMD_func() functions below, several of them call getopt()
701  * then check optind against argc to make sure an extra parameter was not
702  * passed in. The reason this is not caught in the grammar is that the
703  * grammar just checks for a miscellaneous TOKEN, which is *expected* to
704  * be "-F" (for example), but could be anything. So (for example) this
705  * check will prevent "create bogus".
706  */
707
708 cmd_t *
709 alloc_cmd(void)
710 {
711     return (calloc(1, sizeof (cmd_t)));
712 }
713
714 void
715 free_cmd(cmd_t *cmd)
716 {
717     int i;

```

```

719     for (i = 0; i < MAX_EQ_PROP_PAIRS; i++)
720         if (cmd->cmd_property_ptr[i] != NULL) {
721             property_value_ptr_t pp = cmd->cmd_property_ptr[i];
722
723             switch (pp->pv_type) {
724             case PROP_VAL_SIMPLE:
725                 free(pp->pv_simple);
726                 break;
727             case PROP_VAL_COMPLEX:
728                 free_complex(pp->pv_complex);
729                 break;
730             case PROP_VAL_LIST:
731                 free_list(pp->pv_list);
732                 break;
733             }
734         }
735     for (i = 0; i < cmd->cmd_argc; i++)
736         free(cmd->cmd_argv[i]);
737     free(cmd);
738 }
739
740 complex_property_ptr_t
741 alloc_complex(void)
742 {
743     return (calloc(1, sizeof (complex_property_t)));
744 }
745
746 void
747 free_complex(complex_property_ptr_t complex)
748 {
749     if (complex == NULL)
750         return;
751     free_complex(complex->cp_next);
752     if (complex->cp_value != NULL)
753         free(complex->cp_value);
754     free(complex);
755 }
756
757 list_property_ptr_t
758 alloc_list(void)
759 {
760     return (calloc(1, sizeof (list_property_t)));
761 }
762
763 void
764 free_list(list_property_ptr_t list)
765 {
766     if (list == NULL)
767         return;
768     if (list->lp_simple != NULL)
769         free(list->lp_simple);
770     free_complex(list->lp_complex);
771     free_list(list->lp_next);
772     free(list);
773 }
774
775 void
776 free_outer_list(list_property_ptr_t list)
777 {
778     if (list == NULL)
779         return;
780     free_outer_list(list->lp_next);
781     free(list);
782 }

```

```

784 static struct zone_rctlvaltab *
785 alloc_rctlvaltab(void)
786 {
787     return (calloc(1, sizeof (struct zone_rctlvaltab)));
788 }

790 static char *
791 rt_to_str(int res_type)
792 {
793     assert(res_type >= RT_MIN && res_type <= RT_MAX);
794     return (res_types[res_type]);
795 }

797 static char *
798 pt_to_str(int prop_type)
799 {
800     assert(prop_type >= PT_MIN && prop_type <= PT_MAX);
801     return (prop_types[prop_type]);
802 }

804 static char *
805 pvt_to_str(int pv_type)
806 {
807     assert(pv_type >= PROP_VAL_MIN && pv_type <= PROP_VAL_MAX);
808     return (prop_val_types[pv_type]);
809 }

811 static char *
812 cmd_to_str(int cmd_num)
813 {
814     assert(cmd_num >= CMD_MIN && cmd_num <= CMD_MAX);
815     return (helptab[cmd_num].cmd_name);
816 }

818 /* PRINTFLIKE1 */
819 static void
820 zerr(const char *fmt, ...)
821 {
822     va_list alist;
823     static int last_lineno;

825     /* lex_lineno has already been incremented in the lexer; compensate */
826     if (cmd_file_mode && lex_lineno > last_lineno) {
827         if (strcmp(cmd_file_name, "-") == 0)
828             (void) fprintf(stderr, gettext("On line %d:\n"),
829                 lex_lineno - 1);
830         else
831             (void) fprintf(stderr, gettext("On line %d of %s:\n"),
832                 lex_lineno - 1, cmd_file_name);
833         last_lineno = lex_lineno;
834     }
835     va_start(alist, fmt);
836     (void) vfprintf(stderr, fmt, alist);
837     (void) fprintf(stderr, "\n");
838     va_end(alist);
839 }

841 /*
842  * This is a separate function rather than a set of define's because of the
843  * gettext() wrapping.
844  */

846 /*
847  * TRANSLATION_NOTE
848  * Each string below should have \t follow \n whenever needed; the
849  * initial \t and the terminal \n will be provided by the calling function.

```

```

850 */

852 static char *
853 long_help(int cmd_num)
854 {
855     static char line[1024]; /* arbitrary large amount */

857     assert(cmd_num >= CMD_MIN && cmd_num <= CMD_MAX);
858     switch (cmd_num) {
859     case CMD_HELP:
860         return (gettext("Prints help message.));
861     case CMD_CREATE:
862         (void) snprintf(line, sizeof (line),
863             gettext("Creates a configuration for the "
864                 "specified zone. %s should be\n\tused to "
865                 "begin configuring a new zone. If overwriting an "
866                 "existing\n\tconfiguration, the -F flag can be "
867                 "used to force the action. If\n\t-t template is "
868                 "given, creates a configuration identical to the\n"
869                 "\tspecified template, except that the zone name "
870                 "is changed from\n\ttemplate to zonename. 's -a' "
871                 "creates a configuration from a\n\t detached "
872                 "zonepath. 's -b' results in a blank "
873                 "configuration.\n\t't's' with no arguments applies "
874                 "the Sun default settings."),
875             cmd_to_str(CMD_CREATE), cmd_to_str(CMD_CREATE),
876             cmd_to_str(CMD_CREATE), cmd_to_str(CMD_CREATE));
877         return (line);
878     case CMD_EXIT:
879         return (gettext("Exits the program. The -F flag can "
880             "be used to force the action.));
881     case CMD_EXPORT:
882         return (gettext("Prints configuration to standard "
883             "output, or to output-file if\n\tspecified, in "
884             "a form suitable for use in a command-file.));
885     case CMD_ADD:
886         return (gettext("Add specified resource to "
887             "configuration.));
888     case CMD_DELETE:
889         return (gettext("Deletes the specified zone. The -F "
890             "flag can be used to force the\n\taction.));
891     case CMD_REMOVE:
892         return (gettext("Remove specified resource from "
893             "configuration. The -F flag can be used\n\tto "
894             "force the action.));
895     case CMD_SELECT:
896         (void) snprintf(line, sizeof (line),
897             gettext("Selects a resource to modify. "
898                 "Resource modification is completed\n\twith the "
899                 "command \"%s\". The property name/value pairs "
900                 "must uniquely\n\tidentify a resource. Note that "
901                 "the curly braces '{', '}' mean one\n\tor more "
902                 "of whatever is between them."),
903             cmd_to_str(CMD_END));
904         return (line);
905     case CMD_SET:
906         return (gettext("Sets property values.));
907     case CMD_CLEAR:
908         return (gettext("Clears property values.));
909     case CMD_INFO:
910         return (gettext("Displays information about the "
911             "current configuration. If resource\n\ttype is "
912             "specified, displays only information about "
913             "resources of\n\tthe relevant type. If resource "
914             "id is specified, displays only\n\tinformation "
915             "about that resource.));

```

```

916     case CMD_VERIFY:
917         return (gettext("Verifies current configuration "
918             "for correctness (some resource types\n\thave "
919             "required properties)."));
920     case CMD_COMMIT:
921         (void) snprintf(line, sizeof (line),
922             gettext("Commits current configuration. "
923                 "Configuration must be committed to\n\the used by "
924                 "%s. Until the configuration is committed, "
925                 "changes \n\tdcan be removed with the %s "
926                 "command. This operation is\n\tattempted "
927                 "automatically upon completion of a %s "
928                 "session."), "zoneadm", cmd_to_str(CMD_REVERT),
929                 "zoncfg");
930         return (line);
931     case CMD_REVERT:
932         return (gettext("Reverts configuration back to the "
933             "last committed state. The -F flag\n\tdcan be "
934             "used to force the action.));
935     case CMD_CANCEL:
936         return (gettext("Cancels resource/property "
937             "specification.));
938     case CMD_END:
939         return (gettext("Ends resource/property "
940             "specification.));
941 }
942 /* NOTREACHED */
943 return (NULL);
944 }

```

```

946 /*
947 * Return the input filename appended to each component of the path
948 * or the filename itself if it is absolute.
949 * Parameters: path string, file name, output string.
950 */
951 /* Copied almost verbatim from libtnfctl/prb_findexec.c */
952 static const char *
953 exec_cat(const char *s1, const char *s2, char *si)
954 {
955     char *s;
956     /* Number of remaining characters in s */
957     int cnt = PATH_MAX + 1;

```

```

959     s = si;
960     while (*s1 && *s1 != ':') { /* Copy first component of path to si */
961         if (cnt > 0) {
962             *s++ = *s1++;
963             cnt--;
964         } else {
965             s1++;
966         }
967     }
968     if (si != s && cnt > 0) { /* Add slash if s2 is not absolute */
969         *s++ = '/';
970         cnt--;
971     }
972     while (*s2 && cnt > 0) { /* Copy s2 to si */
973         *s++ = *s2++;
974         cnt--;
975     }
976     *s = '\0'; /* Terminate the output string */
977     return (*s1 ? ++s1 : NULL); /* Return next path component or NULL */
978 }

```

```

980 /* Determine that a name exists in PATH */
981 /* Copied with changes from libtnfctl/prb_findexec.c */

```

```

982 static int
983 path_find(const char *name)
984 {
985     const char *pathstr;
986     char fname[PATH_MAX + 2];
987     const char *cp;
988     struct stat stat_buf;

```

```

990     if ((pathstr = getenv("PATH")) == NULL) {
991         if (geteuid() == 0 || getuid() == 0)
992             pathstr = "/usr/sbin:/usr/bin";
993         else
994             pathstr = "/usr/bin:";
995     }
996     cp = strchr(name, '/') ? (const char *) "" : pathstr;

```

```

998     do {
999         cp = exec_cat(cp, name, fname);
1000         if (stat(fname, &stat_buf) != -1) {
1001             /* successful find of the file */
1002             return (0);
1003         }
1004     } while (cp != NULL);

```

```

1006     return (-1);
1007 }

```

```

1009 static FILE *
1010 pager_open(void)
1011 {
1012     FILE *newfp;
1013     char *pager, *space;

```

```

1015     pager = getenv("PAGER");
1016     if (pager == NULL || *pager == '\0')
1017         pager = PAGER;

```

```

1019     space = strchr(pager, ' ');
1020     if (space)
1021         *space = '\0';
1022     if (path_find(pager) == 0) {
1023         if (space)
1024             *space = ' ';
1025         if ((newfp = popen(pager, "w")) == NULL)
1026             zerr(gettext("PAGER open failed (%s)."),
1027                 strerror(errno));
1028         return (newfp);
1029     } else {
1030         zerr(gettext("PAGER %s does not exist (%s)."),
1031             pager, strerror(errno));
1032     }
1033     return (NULL);
1034 }

```

```

1036 static void
1037 pager_close(FILE *fp)
1038 {
1039     pager_close(FILE *fp) {
1040         int status;

```

```

1041         status = pclose(fp);
1042         if (status == -1)
1043             zerr(gettext("PAGER close failed (%s)."),
1044                 strerror(errno));
1045     }

```

```

1047 /*
1048  * Called with verbose TRUE when help is explicitly requested, FALSE for
1049  * unexpected errors.
1050  */

1052 void
1053 usage(boolean_t verbose, uint_t flags)
1054 {
1055     FILE *fp = verbose ? stdout : stderr;
1056     FILE *newfp;
1057     boolean_t need_to_close = B_FALSE;
1058     int i;

1060     /* don't page error output */
1061     if (verbose && interactive_mode) {
1062         if ((newfp = pager_open()) != NULL) {
1063             need_to_close = B_TRUE;
1064             fp = newfp;
1065         }
1066     }

1068     if (flags & HELP_META) {
1069         (void) fprintf(fp, gettext("More help is available for the "
1070             "following:\n"));
1071         (void) fprintf(fp, "\n\tcommands ('%s commands')\n",
1072             cmd_to_str(CMD_HELP));
1073         (void) fprintf(fp, "\tsyntax ('%s syntax')\n",
1074             cmd_to_str(CMD_HELP));
1075         (void) fprintf(fp, "\tusage ('%s usage')\n",
1076             cmd_to_str(CMD_HELP));
1077         (void) fprintf(fp, gettext("You may also obtain help on any "
1078             "command by typing '%s <command-name>.\n"),
1079             cmd_to_str(CMD_HELP));
1080     }

1081     if (flags & HELP_RES_SCOPE) {
1082         switch (resource_scope) {
1083             case RT_FS:
1084                 (void) fprintf(fp, gettext("The '%s' resource scope is "
1085                     "used to configure a file-system.\n"),
1086                     rt_to_str(resource_scope));
1087                 (void) fprintf(fp, gettext("Valid commands:\n"));
1088                 (void) fprintf(fp, "\t%s %s=%s\n", cmd_to_str(CMD_SET),
1089                     pt_to_str(PT_DIR), gettext("<path>"));
1090                 (void) fprintf(fp, "\t%s %s=%s\n", cmd_to_str(CMD_SET),
1091                     pt_to_str(PT_SPECIAL), gettext("<path>"));
1092                 (void) fprintf(fp, "\t%s %s=%s\n", cmd_to_str(CMD_SET),
1093                     pt_to_str(PT_RAW), gettext("<raw-device>"));
1094                 (void) fprintf(fp, "\t%s %s=%s\n", cmd_to_str(CMD_SET),
1095                     pt_to_str(PT_TYPE), gettext("<file-system type>"));
1096                 (void) fprintf(fp, "\t%s %s %s\n", cmd_to_str(CMD_ADD),
1097                     pt_to_str(PT_OPTIONS),
1098                     gettext("<file-system options>"));
1099                 (void) fprintf(fp, "\t%s %s %s\n",
1100                     cmd_to_str(CMD_REMOVE), pt_to_str(PT_OPTIONS),
1101                     gettext("<file-system options>"));
1102                 (void) fprintf(fp, gettext("Consult the file-system "
1103                     "specific manual page, such as mount_ufs(1M), "
1104                     "for\ndetails about file-system options. Note "
1105                     "that any file-system options with an\nembedded "
1106                     "'=' character must be enclosed in double quotes, "
1107                     /*CSTYLED*/
1108                     "such as \"%s=5\".\n"), MNTOPT_RETRY);
1109                 break;
1110             case RT_NET:
1111                 (void) fprintf(fp, gettext("The '%s' resource scope is "

```

```

1112         "used to configure a network interface.\n"),
1113         rt_to_str(resource_scope));
1114         (void) fprintf(fp, gettext("Valid commands:\n"));
1115         (void) fprintf(fp, "\t%s %s=%s\n", cmd_to_str(CMD_SET),
1116             pt_to_str(PT_ADDRESS), gettext("<IP-address>"));
1117         (void) fprintf(fp, "\t%s %s=%s\n", cmd_to_str(CMD_SET),
1118             pt_to_str(PT_ALLOWED_ADDRESS),
1119             gettext("<IP-address>"));
1120         (void) fprintf(fp, "\t%s %s=%s\n", cmd_to_str(CMD_SET),
1121             pt_to_str(PT_PHYSICAL), gettext("<interface>"));
1122         (void) fprintf(fp, gettext("See ifconfig(1M) for "
1123             "details of the <interface> string.\n"));
1124         (void) fprintf(fp, gettext("%s %s is valid "
1125             "if the %s property is set to %s, otherwise it "
1126             "must not be set.\n"),
1127             cmd_to_str(CMD_SET), pt_to_str(PT_ADDRESS),
1128             pt_to_str(PT_IPTYPE), gettext("shared"));
1129         (void) fprintf(fp, gettext("%s %s is valid "
1130             "if the %s property is set to %s, otherwise it "
1131             "must not be set.\n"),
1132             cmd_to_str(CMD_SET), pt_to_str(PT_ALLOWED_ADDRESS),
1133             pt_to_str(PT_IPTYPE), gettext("exclusive"));
1134         (void) fprintf(fp, gettext("\t%s %s=%s\n%s %s "
1135             "is valid if the %s or %s property is set, "
1136             "otherwise it must not be set.\n"),
1137             cmd_to_str(CMD_SET),
1138             pt_to_str(PT_DEFROUTER), gettext("<IP-address>"),
1139             cmd_to_str(CMD_SET), pt_to_str(PT_DEFROUTER),
1140             gettext(pt_to_str(PT_ADDRESS)),
1141             gettext(pt_to_str(PT_ALLOWED_ADDRESS)));
1142         break;
1143     case RT_DEVICE:
1144         (void) fprintf(fp, gettext("The '%s' resource scope is "
1145             "used to configure a device node.\n"),
1146             rt_to_str(resource_scope));
1147         (void) fprintf(fp, gettext("Valid commands:\n"));
1148         (void) fprintf(fp, "\t%s %s=%s\n", cmd_to_str(CMD_SET),
1149             pt_to_str(PT_MATCH), gettext("<device-path>"));
1150         break;
1151     case RT_RCTL:
1152         (void) fprintf(fp, gettext("The '%s' resource scope is "
1153             "used to configure a resource control.\n"),
1154             rt_to_str(resource_scope));
1155         (void) fprintf(fp, gettext("Valid commands:\n"));
1156         (void) fprintf(fp, "\t%s %s=%s\n", cmd_to_str(CMD_SET),
1157             pt_to_str(PT_NAME), gettext("<string>"));
1158         (void) fprintf(fp, "\t%s %s (%s=%s,%s=%s,%s=%s)\n",
1159             cmd_to_str(CMD_ADD), pt_to_str(PT_VALUE),
1160             pt_to_str(PT_PRIV), gettext("<priv-value>"),
1161             pt_to_str(PT_LIMIT), gettext("<number>"),
1162             pt_to_str(PT_ACTION), gettext("<action-value>"));
1163         (void) fprintf(fp, "\t%s %s (%s=%s,%s=%s,%s=%s)\n",
1164             cmd_to_str(CMD_REMOVE), pt_to_str(PT_VALUE),
1165             pt_to_str(PT_PRIV), gettext("<priv-value>"),
1166             pt_to_str(PT_LIMIT), gettext("<number>"),
1167             pt_to_str(PT_ACTION), gettext("<action-value>"));
1168         (void) fprintf(fp, "%s\n\t%s := privileged\n",
1169             "\t%s := none | deny\n", gettext("Where"),
1170             gettext("<priv-value>"), gettext("<action-value>"));
1171         break;
1172     case RT_ATTR:
1173         (void) fprintf(fp, gettext("The '%s' resource scope is "
1174             "used to configure a generic attribute.\n"),
1175             rt_to_str(resource_scope));
1176         (void) fprintf(fp, gettext("Valid commands:\n"));
1177         (void) fprintf(fp, "\t%s %s=%s\n", cmd_to_str(CMD_SET),

```

```

1178     pt_to_str(PT_NAME), gettext("<name>"));
1179     (void) fprintf(fp, "\t%s %s=boolean\n",
1180     cmd_to_str(CMD_SET), pt_to_str(PT_TYPE));
1181     (void) fprintf(fp, "\t%s %s=true | false\n",
1182     cmd_to_str(CMD_SET), pt_to_str(PT_VALUE));
1183     (void) fprintf(fp, gettext("or\n"));
1184     (void) fprintf(fp, "\t%s %s=int\n", cmd_to_str(CMD_SET),
1185     pt_to_str(PT_TYPE));
1186     (void) fprintf(fp, "\t%s %s=%s\n", cmd_to_str(CMD_SET),
1187     pt_to_str(PT_VALUE), gettext("<integer>"));
1188     (void) fprintf(fp, gettext("or\n"));
1189     (void) fprintf(fp, "\t%s %s=string\n",
1190     cmd_to_str(CMD_SET), pt_to_str(PT_TYPE));
1191     (void) fprintf(fp, "\t%s %s=%s\n", cmd_to_str(CMD_SET),
1192     pt_to_str(PT_VALUE), gettext("<string>"));
1193     (void) fprintf(fp, gettext("or\n"));
1194     (void) fprintf(fp, "\t%s %s=uint\n",
1195     cmd_to_str(CMD_SET), pt_to_str(PT_TYPE));
1196     (void) fprintf(fp, "\t%s %s=%s\n", cmd_to_str(CMD_SET),
1197     pt_to_str(PT_VALUE), gettext("<unsigned integer>"));
1198     break;
1199 case RT_DATASET:
1200     (void) fprintf(fp, gettext("The '%s' resource scope is "
1201     "used to export ZFS datasets.\n"),
1202     rt_to_str(resource_scope));
1203     (void) fprintf(fp, gettext("Valid commands:\n"));
1204     (void) fprintf(fp, "\t%s %s=%s\n", cmd_to_str(CMD_SET),
1205     pt_to_str(PT_NAME), gettext("<name>"));
1206     break;
1207 case RT_DCPU:
1208     (void) fprintf(fp, gettext("The '%s' resource scope "
1209     "configures the 'pools' facility to dedicate\na
1210     "subset of the system's processors to this zone "
1211     "while it is running.\n"),
1212     rt_to_str(resource_scope));
1213     (void) fprintf(fp, gettext("Valid commands:\n"));
1214     (void) fprintf(fp, "\t%s %s=%s\n", cmd_to_str(CMD_SET),
1215     pt_to_str(PT_NCPUS),
1216     gettext("<unsigned integer | range>"));
1217     (void) fprintf(fp, "\t%s %s=%s\n", cmd_to_str(CMD_SET),
1218     pt_to_str(PT_IMPORTANCE),
1219     gettext("<unsigned integer>"));
1220     break;
1221 case RT_PCAP:
1222     (void) fprintf(fp, gettext("The '%s' resource scope is "
1223     "used to set an upper limit (a cap) on the\n
1224     "percentage of CPU that can be used by this zone. "
1225     "A '%s' value of 1\ncorresponds to one cpu. The "
1226     "value can be set higher than 1, up to the total\n
1227     "number of CPUs on the system. The value can "
1228     "also be less than 1,\nrepresenting a fraction of "
1229     "a cpu.\n"),
1230     rt_to_str(resource_scope), pt_to_str(PT_NCPUS));
1231     (void) fprintf(fp, gettext("Valid commands:\n"));
1232     (void) fprintf(fp, "\t%s %s=%s\n", cmd_to_str(CMD_SET),
1233     pt_to_str(PT_NCPUS), gettext("<unsigned decimal>"));
1234     break;
1235 case RT_MCAP:
1236     (void) fprintf(fp, gettext("The '%s' resource scope is "
1237     "used to set an upper limit (a cap) on the\n
1238     "amount of physical memory, swap space and locked "
1239     "memory that can be used by\nthis zone.\n"),
1240     rt_to_str(resource_scope));
1241     (void) fprintf(fp, gettext("Valid commands:\n"));
1242     (void) fprintf(fp, "\t%s %s=%s\n", cmd_to_str(CMD_SET),
1243     pt_to_str(PT_PHYSICAL),

```

```

1244     gettext("<qualified unsigned decimal>"));
1245     (void) fprintf(fp, "\t%s %s=%s\n", cmd_to_str(CMD_SET),
1246     pt_to_str(PT_SWAP),
1247     gettext("<qualified unsigned decimal>"));
1248     (void) fprintf(fp, "\t%s %s=%s\n", cmd_to_str(CMD_SET),
1249     pt_to_str(PT_LOCKED),
1250     gettext("<qualified unsigned decimal>"));
1251     break;
1252 case RT_ADMIN:
1253     (void) fprintf(fp, gettext("The '%s' resource scope is "
1254     "used to delegate specific zone management\n
1255     "rights to users and roles. These rights are "
1256     "only applicable to this zone.\n"),
1257     rt_to_str(resource_scope));
1258     (void) fprintf(fp, gettext("Valid commands:\n"));
1259     (void) fprintf(fp, "\t%s %s=%s\n", cmd_to_str(CMD_SET),
1260     pt_to_str(PT_USER),
1261     gettext("<single user or role name>"));
1262     (void) fprintf(fp, "\t%s %s=%s\n", cmd_to_str(CMD_SET),
1263     pt_to_str(PT_AUTHS),
1264     gettext("<comma separated list>"));
1265     break;
1266 case RT_SECFLAGS:
1267     (void) fprintf(fp, gettext("The '%s' resource scope is "
1268     "used to specify the default security-flags\n
1269     "of this zone, and their upper and lower bound.\n"),
1270     rt_to_str(resource_scope));
1271     (void) fprintf(fp, "\t%s %s=%s\n",
1272     cmd_to_str(CMD_SET), pt_to_str(PT_DEFAULT),
1273     gettext("<security flags>"));
1274     (void) fprintf(fp, "\t%s %s=%s\n",
1275     cmd_to_str(CMD_SET), pt_to_str(PT_LOWER),
1276     gettext("<security flags>"));
1277     (void) fprintf(fp, "\t%s %s=%s\n",
1278     cmd_to_str(CMD_SET), pt_to_str(PT_UPPER),
1279     gettext("<security flags>"));
1280     break;
1281 #endif /* ! codereview */
1282     }
1283     (void) fprintf(fp, gettext("And from any resource scope, you "
1284     "can:\n"));
1285     (void) fprintf(fp, "\t%s\t%s\n", cmd_to_str(CMD_END),
1286     gettext("(to conclude this operation)"));
1287     (void) fprintf(fp, "\t%s\t%s\n", cmd_to_str(CMD_CANCEL),
1288     gettext("(to cancel this operation)"));
1289     (void) fprintf(fp, "\t%s\t%s\n", cmd_to_str(CMD_EXIT),
1290     gettext("(to exit the zoncfg utility)"));
1291     }
1292     if (flags & HELP_USAGE) {
1293         (void) fprintf(fp, "%s:\t%s %s\n", gettext("usage"),
1294         execname, cmd_to_str(CMD_HELP));
1295         (void) fprintf(fp, "\t%s -z <zone>\t\t\t%s\n",
1296         execname, gettext("interactive"));
1297         (void) fprintf(fp, "\t%s -z <zone> <command>\n", execname);
1298         (void) fprintf(fp, "\t%s -z <zone> -f <command-file>\n",
1299         execname);
1300     }
1301     if (flags & HELP_SUBCMDS) {
1302         (void) fprintf(fp, "%s:\n\n", gettext("Commands"));
1303         for (i = 0; i <= CMD_MAX; i++) {
1304             (void) fprintf(fp, "%s\n", helptab[i].short_usage);
1305             if (verbose)
1306                 (void) fprintf(fp, "\t%s\n\n", long_help(i));
1307         }
1308     }
1309     if (flags & HELP_SYNTAX) {

```

```

1310     if (!verbose)
1311         (void) fprintf(fp, "\n");
1312     (void) fprintf(fp, "<zname> := [A-Za-z0-9][A-Za-z0-9_-]*\n");
1313     (void) fprintf(fp, gettext("\t(except the reserved words "
1314         "'%s' and anything starting with '%s'\n)", "global",
1315         "SUNW"));
1316     (void) fprintf(fp,
1317         gettext("\tName must be less than %d characters.\n"),
1318         ZONENAME_MAX);
1319     if (verbose)
1320         (void) fprintf(fp, "\n");
1321 }
1322 if (flags & HELP_NETADDR) {
1323     (void) fprintf(fp, gettext("\n<net-addr> :="));
1324     (void) fprintf(fp,
1325         gettext("\t<IPv4-address>[/<IPv4-prefix-length>] |\n"));
1326     (void) fprintf(fp,
1327         gettext("\t<IPv6-address>/<IPv6-prefix-length> |\n"));
1328     (void) fprintf(fp,
1329         gettext("\t<hostname>[/<IPv4-prefix-length>] |\n"));
1330     (void) fprintf(fp, gettext("See inet(3SOCKET) for IPv4 and
1331         "IPv6 address syntax.\n"));
1332     (void) fprintf(fp, gettext("<IPv4-prefix-length> := [0-32]\n"));
1333     (void) fprintf(fp,
1334         gettext("<IPv6-prefix-length> := [0-128]\n"));
1335     (void) fprintf(fp,
1336         gettext("<hostname> := [A-Za-z0-9][A-Za-z0-9-.*]\n"));
1337 }
1338 if (flags & HELP_RESOURCES) {
1339     (void) fprintf(fp, "<rs> := %s | %s | %s | %s | %s |\n\t"
1340         "%s | %s | %s | %s | %s\n",
1341         gettext("resource type"), rt_to_str(RT_FS),
1342         rt_to_str(RT_NET), rt_to_str(RT_DEVICE),
1343         rt_to_str(RT_RCTL), rt_to_str(RT_ATTR),
1344         rt_to_str(RT_DATASET), rt_to_str(RT_DCPU),
1345         rt_to_str(RT_PCAP), rt_to_str(RT_MCAP),
1346         rt_to_str(RT_ADMIN), rt_to_str(RT_SECFLAGS));
1347     rt_to_str(RT_ADMIN));
1348 }
1349 if (flags & HELP_PROPS) {
1350     (void) fprintf(fp, gettext("For resource type ... there are "
1351         "property types ...:\n"));
1352     (void) fprintf(fp, "\t%s\t%s\n", gettext("global"),
1353         pt_to_str(PT_ZONENAME));
1354     (void) fprintf(fp, "\t%s\t%s\n", gettext("global"),
1355         pt_to_str(PT_ZONEPATH));
1356     (void) fprintf(fp, "\t%s\t%s\n", gettext("global"),
1357         pt_to_str(PT_BRAND));
1358     (void) fprintf(fp, "\t%s\t%s\n", gettext("global"),
1359         pt_to_str(PT_AUTOBOOT));
1360     (void) fprintf(fp, "\t%s\t%s\n", gettext("global"),
1361         pt_to_str(PT_BOOTARGS));
1362     (void) fprintf(fp, "\t%s\t%s\n", gettext("global"),
1363         pt_to_str(PT_POOL));
1364     (void) fprintf(fp, "\t%s\t%s\n", gettext("global"),
1365         pt_to_str(PT_LIMITPRIV));
1366     (void) fprintf(fp, "\t%s\t%s\n", gettext("global"),
1367         pt_to_str(PT_SCHED));
1368     (void) fprintf(fp, "\t%s\t%s\n", gettext("global"),
1369         pt_to_str(PT_IPTYPE));
1370     (void) fprintf(fp, "\t%s\t%s\n", gettext("global"),
1371         pt_to_str(PT_HOSTID));
1372     (void) fprintf(fp, "\t%s\t%s\n", gettext("global"),
1373         pt_to_str(PT_FS_ALLOWED));
1374     (void) fprintf(fp, "\t%s\t%s\n", gettext("global"),
1375         pt_to_str(PT_MAXLWPS));

```

```

1375     (void) fprintf(fp, "\t%s\t%s\n", gettext("global"),
1376         pt_to_str(PT_MAXPROCS));
1377     (void) fprintf(fp, "\t%s\t%s\n", gettext("global"),
1378         pt_to_str(PT_MAXSHMEM));
1379     (void) fprintf(fp, "\t%s\t%s\n", gettext("global"),
1380         pt_to_str(PT_MAXSHMIDS));
1381     (void) fprintf(fp, "\t%s\t%s\n", gettext("global"),
1382         pt_to_str(PT_MAXMSGIDS));
1383     (void) fprintf(fp, "\t%s\t%s\n", gettext("global"),
1384         pt_to_str(PT_MAXSEMIDS));
1385     (void) fprintf(fp, "\t%s\t%s\n", gettext("global"),
1386         pt_to_str(PT_SHARES));
1387     (void) fprintf(fp, "\t%s\t\t%s, %s, %s, %s, %s\n",
1388         rt_to_str(RT_FS), pt_to_str(PT_DIR),
1389         pt_to_str(PT_SPECIAL), pt_to_str(PT_RAW),
1390         pt_to_str(PT_TYPE), pt_to_str(PT_OPTIONS));
1391     (void) fprintf(fp, "\t%s\t\t%s, %s, %s\n", rt_to_str(RT_NET),
1392         pt_to_str(PT_ADDRESS), pt_to_str(PT_ALLOWED_ADDRESS),
1393         pt_to_str(PT_PHYSICAL), pt_to_str(PT_DEFROUTER));
1394     (void) fprintf(fp, "\t%s\t\t%s\n", rt_to_str(RT_DEVICE),
1395         pt_to_str(PT_MATCH));
1396     (void) fprintf(fp, "\t%s\t\t%s, %s\n", rt_to_str(RT_RCTL),
1397         pt_to_str(PT_NAME), pt_to_str(PT_VALUE));
1398     (void) fprintf(fp, "\t%s\t\t%s, %s, %s\n", rt_to_str(RT_ATTR),
1399         pt_to_str(PT_NAME), pt_to_str(PT_TYPE),
1400         pt_to_str(PT_VALUE));
1401     (void) fprintf(fp, "\t%s\t\t%s\n", rt_to_str(RT_DATASET),
1402         pt_to_str(PT_NAME));
1403     (void) fprintf(fp, "\t%s\t\t%s, %s\n", rt_to_str(RT_DCPU),
1404         pt_to_str(PT_NCPU), pt_to_str(PT_IMPORTANCE));
1405     (void) fprintf(fp, "\t%s\t\t%s\n", rt_to_str(RT_PCAP),
1406         pt_to_str(PT_NCPU));
1407     (void) fprintf(fp, "\t%s\t\t%s, %s, %s\n", rt_to_str(RT_MCAP),
1408         pt_to_str(PT_PHYSICAL), pt_to_str(PT_SWAP),
1409         pt_to_str(PT_LOCKED));
1410     (void) fprintf(fp, "\t%s\t\t\t%s, %s\n", rt_to_str(RT_ADMIN),
1411         pt_to_str(PT_USER), pt_to_str(PT_AUTHS));
1412     (void) fprintf(fp, "\t%s\t\t\t%s, %s, %s\n",
1413         rt_to_str(RT_SECFLAGS), pt_to_str(PT_DEFAULT),
1414         pt_to_str(PT_LOWER), pt_to_str(PT_UPPER));
1415 #endif /* ! codereview */
1416     }
1417     if (need_to_close)
1418         (void) pager_close(fp);
1419 }
1420
1421 static void
1422 zone_perror(char *prefix, int err, boolean_t set_saw)
1423 {
1424     zerr("%s: %s", prefix, zoncfg_strerror(err));
1425     if (set_saw)
1426         saw_error = B_TRUE;
1427 }
1428
1429 /*
1430  * zone_perror() expects a single string, but for remove and select
1431  * we have both the command and the resource type, so this wrapper
1432  * function serves the same purpose in a slightly different way.
1433  */
1434
1435 static void
1436 z_cmd_rt_perror(int cmd_num, int res_num, int err, boolean_t set_saw)
1437 {
1438     zerr("%s %s: %s", cmd_to_str(cmd_num), rt_to_str(res_num),
1439         zoncfg_strerror(err));
1440     if (set_saw)

```

```

1441         saw_error = B_TRUE;
1442     }

1444 /* returns Z_OK if successful, Z_foo from <libzonecfg.h> otherwise */
1445 static int
1446 initialize(boolean_t handle_expected)
1447 {
1448     int err;
1449     char brandname[MAXNAMELEN];

1451     if (zonecfg_check_handle(handle) != Z_OK) {
1452         if ((err = zonecfg_get_handle(zone, handle)) == Z_OK) {
1453             got_handle = B_TRUE;
1454             if (zonecfg_get_brand(handle, brandname,
1455                 sizeof(brandname)) != Z_OK) {
1456                 zerr("Zone %s is inconsistent: missing "
1457                     "brand attribute", zone);
1458                 exit(Z_ERR);
1459             }
1460             if ((brand = brand_open(brandname)) == NULL) {
1461                 zerr("Zone %s uses non-existent brand \"%s\"."
1462                     " Unable to continue", zone, brandname);
1463                 exit(Z_ERR);
1464             }
1465             /*
1466              * If the user_attr file is newer than
1467              * the zone config file, the admins
1468              * may need to be updated since the
1469              * RBAC files are authoritative for
1470              * authorization checks.
1471              */
1472             err = zonecfg_update_userauths(handle, zone);
1473             if (err == Z_OK) {
1474                 zerr(gettext("The administrative rights "
1475                     "were updated to match "
1476                     "the current RBAC configuration.\n"
1477                     "Use \"info admin\" and \"revert\" to "
1478                     "compare with the previous settings."));
1479                 need_to_commit = B_TRUE;
1480             } else if (err != Z_NO_ENTRY) {
1481                 zerr(gettext("failed to update "
1482                     "admin rights."));
1483                 exit(Z_ERR);
1484             } else if (need_to_commit) {
1485                 zerr(gettext("admin rights were updated "
1486                     "to match RBAC configuration."));
1487             }
1488         } else if (global_zone && err == Z_NO_ZONE && !got_handle &&
1489             !read_only_mode) {
1490             /*
1491              * We implicitly create the global zone config if it
1492              * doesn't exist.
1493              */
1494             zone_dochandle_t tmphandle;

1497             if ((tmphandle = zonecfg_init_handle()) == NULL) {
1498                 zone_perror(execname, Z_NOMEM, B_TRUE);
1499                 exit(Z_ERR);
1500             }

1502             err = zonecfg_get_template_handle("SUNWblank", zone,
1503                 tmphandle);

1505             if (err != Z_OK) {
1506                 zonecfg_fini_handle(tmphandle);

```

```

1507         zone_perror("SUNWblank", err, B_TRUE);
1508         return (err);
1509     }

1511     need_to_commit = B_TRUE;
1512     zonecfg_fini_handle(handle);
1513     handle = tmphandle;
1514     got_handle = B_TRUE;

1516     } else {
1517         zone_perror(zone, err, handle_expected || got_handle);
1518         if (err == Z_NO_ZONE && !got_handle &&
1519             interactive_mode && !read_only_mode)
1520             (void) printf(gettext("Use '%s' to begin "
1521                 "configuring a new zone.\n"),
1522                 cmd_to_str(CMD_CREATE));
1523         return (err);
1524     }
1525     }
1526     return (Z_OK);
1527 }

1529 static boolean_t
1530 state_atleast(zone_state_t state)
1531 {
1532     zone_state_t state_num;
1533     int err;

1535     if ((err = zone_get_state(zone, &state_num)) != Z_OK) {
1536         /* all states are greater than "non-existent" */
1537         if (err == Z_NO_ZONE)
1538             return (B_FALSE);
1539         zerr(gettext("Unexpectedly failed to determine state "
1540             "of zone %s: %s"), zone, zonecfg_strerror(err));
1541         exit(Z_ERR);
1542     }
1543     return (state_num >= state);
1544 }

1546 /*
1547  * short_usage() is for bad syntax: getopt() issues, too many arguments, etc.
1548  */

1550 void
1551 short_usage(int command)
1552 {
1553     /* lex_lineno has already been incremented in the lexer; compensate */
1554     if (cmd_file_mode) {
1555         if (strcmp(cmd_file_name, "-") == 0)
1556             (void) fprintf(stderr,
1557                 gettext("syntax error on line %d\n"),
1558                 lex_lineno - 1);
1559         else
1560             (void) fprintf(stderr,
1561                 gettext("syntax error on line %d of %s\n"),
1562                 lex_lineno - 1, cmd_file_name);
1563     }
1564     (void) fprintf(stderr, "%s:\n%s\n", gettext("usage"),
1565         helptab[command].short_usage);
1566     saw_error = B_TRUE;
1567 }

1569 /*
1570  * long_usage() is for bad semantics: e.g., wrong property type for a given
1571  * resource type. It is also used by longer_usage() below.
1572  */

```



```

1574 void
1575 long_usage(uint_t cmd_num, boolean_t set_saw)
1576 {
1577     (void) fprintf(set_saw ? stderr : stdout, "%s:\n%s\n", gettext("usage"),
1578                 helptab[cmd_num].short_usage);
1579     (void) fprintf(set_saw ? stderr : stdout, "\t%s\n", long_help(cmd_num));
1580     if (set_saw)
1581         saw_error = B_TRUE;
1582 }

1584 /*
1585  * longer_usage() is for 'help foo' and 'foo -?': call long_usage() and also
1586  * any extra usage() flags as appropriate for whatever command.
1587  */

1589 void
1590 longer_usage(uint_t cmd_num)
1591 {
1592     long_usage(cmd_num, B_FALSE);
1593     if (helptab[cmd_num].flags != 0) {
1594         (void) printf("\n");
1595         usage(B_TRUE, helptab[cmd_num].flags);
1596     }
1597 }

1599 /*
1600  * scope_usage() is simply used when a command is called from the wrong scope.
1601  */

1603 static void
1604 scope_usage(uint_t cmd_num)
1605 {
1606     zerr(gettext("The %s command only makes sense in the %s scope."),
1607         cmd_to_str(cmd_num),
1608         global_scope ? gettext("resource") : gettext("global"));
1609     saw_error = B_TRUE;
1610 }

1612 /*
1613  * On input, B_TRUE => yes, B_FALSE => no.
1614  * On return, B_TRUE => 1, B_FALSE => no, could not ask => -1.
1615  */

1617 static int
1618 ask_yesno(boolean_t default_answer, const char *question)
1619 {
1620     char line[64]; /* should be enough to answer yes or no */

1622     if (!ok_to_prompt) {
1623         saw_error = B_TRUE;
1624         return (-1);
1625     }
1626     for (;;) {
1627         if (printf("%s (%s)? ", question,
1628                 default_answer ? "[y]/n" : "[y]/[n]") < 0)
1629             return (-1);
1630         if (fgets(line, sizeof (line), stdin) == NULL)
1631             return (-1);

1633         if (line[0] == '\n')
1634             return (default_answer ? 1 : 0);
1635         if (tolower(line[0]) == 'y')
1636             return (1);
1637         if (tolower(line[0]) == 'n')
1638             return (0);

```

```

1639     }
1640 }

1642 /*
1643  * Prints warning if zone already exists.
1644  * In interactive mode, prompts if we should continue anyway and returns Z_OK
1645  * if so, Z_ERR if not. In non-interactive mode, exits with Z_ERR.
1646  *
1647  * Note that if a zone exists and its state is >= INSTALLED, an error message
1648  * will be printed and this function will return Z_ERR regardless of mode.
1649  */

1651 static int
1652 check_if_zone_already_exists(boolean_t force)
1653 {
1654     char line[ZONENAME_MAX + 128]; /* enough to ask a question */
1655     zone_dohandle_t tmphandle;
1656     int res, answer;

1658     if ((tmphandle = zonecfg_init_handle()) == NULL) {
1659         zone_perror(execname, Z_NOMEM, B_TRUE);
1660         exit(Z_ERR);
1661     }
1662     res = zonecfg_get_handle(zone, tmphandle);
1663     zonecfg_fini_handle(tmphandle);
1664     if (res != Z_OK)
1665         return (Z_OK);

1667     if (state_atleast(ZONE_STATE_INSTALLED)) {
1668         zerr(gettext("Zone %s already installed; %s not allowed."),
1669             zone, cmd_to_str(CMD_CREATE));
1670         return (Z_ERR);
1671     }

1673     if (force) {
1674         (void) printf(gettext("Zone %s already exists; overwriting.\n"),
1675             zone);
1676         return (Z_OK);
1677     }
1678     (void) snprintf(line, sizeof (line),
1679         gettext("Zone %s already exists; %s anyway"), zone,
1680         cmd_to_str(CMD_CREATE));
1681     if ((answer = ask_yesno(B_FALSE, line)) == -1) {
1682         zerr(gettext("Zone exists, input not from terminal and -F not "
1683             "specified:\n%s command ignored, exiting."),
1684             cmd_to_str(CMD_CREATE));
1685         exit(Z_ERR);
1686     }
1687     return (answer == 1 ? Z_OK : Z_ERR);
1688 }

1690 static boolean_t
1691 zone_is_read_only(int cmd_num)
1692 {
1693     if (strncmp(zone, "SUNW", 4) == 0) {
1694         zerr(gettext("%s: zones beginning with SUNW are read-only."),
1695             zone);
1696         saw_error = B_TRUE;
1697         return (B_TRUE);
1698     }
1699     if (read_only_mode) {
1700         zerr(gettext("%s: cannot %s in read-only mode."), zone,
1701             cmd_to_str(cmd_num));
1702         saw_error = B_TRUE;
1703         return (B_TRUE);
1704     }

```

```

1705     return (B_FALSE);
1706 }

1708 /*
1709  * Create a new configuration.
1710  */
1711 void
1712 create_func(cmd_t *cmd)
1713 {
1714     int err, arg;
1715     char zone_template[ZONENAME_MAX];
1716     char attach_path[MAXPATHLEN];
1717     zone_dochandle_t tmphandle;
1718     boolean_t force = B_FALSE;
1719     boolean_t attach = B_FALSE;
1720     boolean_t arg_err = B_FALSE;

1722     assert(cmd != NULL);

1724     /* This is the default if no arguments are given. */
1725     (void) strlcpy(zone_template, "SUNWdefault", sizeof (zone_template));

1727     optind = 0;
1728     while ((arg = getopt(cmd->cmd_argc, cmd->cmd_argv, "?a:bft:"))
1729         != EOF) {
1730         switch (arg) {
1731             case '?':
1732                 if (optopt == '?')
1733                     longer_usage(CMD_CREATE);
1734                 else
1735                     short_usage(CMD_CREATE);
1736                 arg_err = B_TRUE;
1737                 break;
1738             case 'a':
1739                 (void) strlcpy(attach_path, optarg,
1740                     sizeof (attach_path));
1741                 attach = B_TRUE;
1742                 break;
1743             case 'b':
1744                 (void) strlcpy(zone_template, "SUNWblank",
1745                     sizeof (zone_template));
1746                 break;
1747             case 'F':
1748                 force = B_TRUE;
1749                 break;
1750             case 't':
1751                 (void) strlcpy(zone_template, optarg,
1752                     sizeof (zone_template));
1753                 break;
1754             default:
1755                 short_usage(CMD_CREATE);
1756                 arg_err = B_TRUE;
1757                 break;
1758         }
1759     }
1760     if (arg_err)
1761         return;

1763     if (optind != cmd->cmd_argc) {
1764         short_usage(CMD_CREATE);
1765         return;
1766     }

1768     if (zone_is_read_only(CMD_CREATE))
1769         return;

```

```

1771     if (check_if_zone_already_exists(force) != Z_OK)
1772         return;

1774     /*
1775     * Get a temporary handle first. If that fails, the old handle
1776     * will not be lost. Then finish whichever one we don't need,
1777     * to avoid leaks. Then get the handle for zone_template, and
1778     * set the name to zone: this "copy, rename" method is how
1779     * create -[b|t] works.
1780     */
1781     if ((tmphandle = zonecfg_init_handle()) == NULL) {
1782         zone_perror(execname, Z_NOMEM, B_TRUE);
1783         exit(Z_ERR);
1784     }

1786     if (attach)
1787         err = zonecfg_get_attach_handle(attach_path, ZONE_DETACHED,
1788             zone, B_FALSE, tmphandle);
1789     else
1790         err = zonecfg_get_template_handle(zone_template, zone,
1791             tmphandle);

1793     if (err != Z_OK) {
1794         zonecfg_fini_handle(tmphandle);
1795         if (attach && err == Z_NO_ZONE)
1796             (void) fprintf(stderr, gettext("invalid path to "
1797                 "detached zone\n"));
1798         else if (attach && err == Z_INVALID_DOCUMENT)
1799             (void) fprintf(stderr, gettext("Cannot attach to an "
1800                 "earlier release of the operating system\n"));
1801         else
1802             zone_perror(zone_template, err, B_TRUE);
1803         return;
1804     }

1806     need_to_commit = B_TRUE;
1807     zonecfg_fini_handle(handle);
1808     handle = tmphandle;
1809     got_handle = B_TRUE;
1810 }

1812 /*
1813  * This malloc()'s memory, which must be freed by the caller.
1814  */
1815 static char *
1816 quoteit(char *instr)
1817 {
1818     char *outstr;
1819     size_t outstrsize = strlen(instr) + 3; /* 2 quotes + '\0' */

1821     if ((outstr = malloc(outstrsize)) == NULL) {
1822         zone_perror(zone, Z_NOMEM, B_FALSE);
1823         exit(Z_ERR);
1824     }
1825     if (strchr(instr, ' ') == NULL) {
1826         (void) strlcpy(outstr, instr, outstrsize);
1827         return (outstr);
1828     }
1829     (void) snprintf(outstr, outstrsize, "\"%s\"", instr);
1830     return (outstr);
1831 }

1833 static void
1834 export_prop(FILE *of, int prop_num, char *prop_id)
1835 {
1836     char *quote_str;

```

```

1838     if (strlen(prop_id) == 0)
1839         return;
1840     quote_str = quoteit(prop_id);
1841     (void) fprintf(of, "%s %s=%s\n", cmd_to_str(CMD_SET),
1842                 pt_to_str(prop_num), quote_str);
1843     free(quote_str);
1844 }

1846 void
1847 export_func(cmd_t *cmd)
1848 {
1849     struct zone_nwifstab nwifstab;
1850     struct zone_fstab fstab;
1851     struct zone_devtab devtab;
1852     struct zone_attrtab attrtab;
1853     struct zone_rctltab rctltab;
1854     struct zone_dstab dstab;
1855     struct zone_psettab psettab;
1856     struct zone_mcaptab mcaptab;
1857     struct zone_rctlvaltab *valp;
1858     struct zone_admintab admintab;
1859     struct zone_secflagstab secflagstab;
1860 #endif /* ! codereview */
1861     int err, arg;
1862     char zonepath[MAXPATHLEN], outfile[MAXPATHLEN], pool[MAXNAMELEN];
1863     char bootargs[BOOTARGS_MAX];
1864     char sched[MAXNAMELEN];
1865     char brand[MAXNAMELEN];
1866     char hostidp[HW_HOSTID_LEN];
1867     char fsallowedp[ZONE_FS_ALLOWED_MAX];
1868     char *limitpriv;
1869     FILE *of;
1870     boolean_t autoboot;
1871     zone_iptype_t iptype;
1872     boolean_t need_to_close = B_FALSE;
1873     boolean_t arg_err = B_FALSE;

1875     assert(cmd != NULL);

1877     outfile[0] = '\0';
1878     optind = 0;
1879     while ((arg = getopt(cmd->cmd_argc, cmd->cmd_argv, "?f:")) != EOF) {
1880         switch (arg) {
1881             case '?':
1882                 if (optopt == '?')
1883                     longer_usage(CMD_EXPORT);
1884                 else
1885                     short_usage(CMD_EXPORT);
1886                 arg_err = B_TRUE;
1887                 break;
1888             case 'f':
1889                 (void) strcpy(outfile, optarg, sizeof (outfile));
1890                 break;
1891             default:
1892                 short_usage(CMD_EXPORT);
1893                 arg_err = B_TRUE;
1894                 break;
1895         }
1896     }
1897     if (arg_err)
1898         return;

1900     if (optind != cmd->cmd_argc) {
1901         short_usage(CMD_EXPORT);
1902         return;

```

```

1903     }
1904     if (strlen(outfile) == 0) {
1905         of = stdout;
1906     } else {
1907         if ((of = fopen(outfile, "w")) == NULL) {
1908             zerr(gettext("opening file %s: %s"),
1909                 outfile, strerror(errno));
1910             goto done;
1911         }
1912         setbuf(of, NULL);
1913         need_to_close = B_TRUE;
1914     }

1916     if ((err = initialize(B_TRUE)) != Z_OK)
1917         goto done;

1919     (void) fprintf(of, "%s -b\n", cmd_to_str(CMD_CREATE));

1921     if (zonecfg_get_zonepath(handle, zonepath, sizeof (zonepath)) == Z_OK &&
1922         strlen(zonepath) > 0)
1923         (void) fprintf(of, "%s %s=%s\n", cmd_to_str(CMD_SET),
1924                     pt_to_str(PT_ZONEPATH), zonepath);

1926     if ((zone_get_brand(zone, brand, sizeof (brand)) == Z_OK &&
1927         (strcmp(brand, NATIVE_BRAND_NAME) != 0))
1928         (void) fprintf(of, "%s %s=%s\n", cmd_to_str(CMD_SET),
1929                     pt_to_str(PT_BRAND), brand);

1931     if (zonecfg_get_autoboot(handle, &autoboot) == Z_OK)
1932         (void) fprintf(of, "%s %s=%s\n", cmd_to_str(CMD_SET),
1933                     pt_to_str(PT_AUTOBOOT), autoboot ? "true" : "false");

1935     if (zonecfg_get_bootargs(handle, bootargs, sizeof (bootargs)) == Z_OK &&
1936         strlen(bootargs) > 0) {
1937         (void) fprintf(of, "%s %s=%s\n", cmd_to_str(CMD_SET),
1938                     pt_to_str(PT_BOOTARGS), bootargs);
1939     }

1941     if (zonecfg_get_pool(handle, pool, sizeof (pool)) == Z_OK &&
1942         strlen(pool) > 0)
1943         (void) fprintf(of, "%s %s=%s\n", cmd_to_str(CMD_SET),
1944                     pt_to_str(PT_POOL), pool);

1946     if (zonecfg_get_limitpriv(handle, &limitpriv) == Z_OK &&
1947         strlen(limitpriv) > 0) {
1948         (void) fprintf(of, "%s %s=%s\n", cmd_to_str(CMD_SET),
1949                     pt_to_str(PT_LIMITPRIV), limitpriv);
1950         free(limitpriv);
1951     }

1953     if (zonecfg_get_sched_class(handle, sched, sizeof (sched)) == Z_OK &&
1954         strlen(sched) > 0)
1955         (void) fprintf(of, "%s %s=%s\n", cmd_to_str(CMD_SET),
1956                     pt_to_str(PT_SCHED), sched);

1958     if (zonecfg_get_iptype(handle, &iptype) == Z_OK) {
1959         switch (iptype) {
1960             case ZS_SHARED:
1961                 (void) fprintf(of, "%s %s=%s\n", cmd_to_str(CMD_SET),
1962                             pt_to_str(PT_IPTYPE), "shared");
1963                 break;
1964             case ZS_EXCLUSIVE:
1965                 (void) fprintf(of, "%s %s=%s\n", cmd_to_str(CMD_SET),
1966                             pt_to_str(PT_IPTYPE), "exclusive");
1967                 break;
1968         }

```

```

1969     }
1971     if (zoncfg_get_hostid(handle, hostidp, sizeof (hostidp)) == Z_OK) {
1972         (void) fprintf(of, "%s %s=%s\n", cmd_to_str(CMD_SET),
1973             pt_to_str(PT_HOSTID), hostidp);
1974     }
1976     if (zoncfg_get_fs_allowed(handle, fsallowedp,
1977         sizeof (fsallowedp)) == Z_OK) {
1978         (void) fprintf(of, "%s %s=%s\n", cmd_to_str(CMD_SET),
1979             pt_to_str(PT_FS_ALLOWED), fsallowedp);
1980     }
1982     if ((err = zoncfg_setfsent(handle)) != Z_OK) {
1983         zone_perror(zone, err, B_FALSE);
1984         goto done;
1985     }
1986     while (zoncfg_getfsent(handle, &fstab) == Z_OK) {
1987         zone_fsopt_t *optptr;
1989         (void) fprintf(of, "%s %s\n", cmd_to_str(CMD_ADD),
1990             rt_to_str(RT_FS));
1991         export_prop(of, PT_DIR, fstab.zone_fs_dir);
1992         export_prop(of, PT_SPECIAL, fstab.zone_fs_special);
1993         export_prop(of, PT_RAW, fstab.zone_fs_raw);
1994         export_prop(of, PT_TYPE, fstab.zone_fs_type);
1995         for (optptr = fstab.zone_fs_options; optptr != NULL;
1996             optptr = optptr->zone_fsopt_next) {
1997             /*
1998              * Simple property values with embedded equal signs
1999              * need to be quoted to prevent the lexer from
2000              * mis-parsing them as complex name=value pairs.
2001              */
2002             if (strchr(optptr->zone_fsopt_opt, '='))
2003                 (void) fprintf(of, "%s %s \"%s\"\n",
2004                     cmd_to_str(CMD_ADD),
2005                     pt_to_str(PT_OPTIONS),
2006                     optptr->zone_fsopt_opt);
2007             else
2008                 (void) fprintf(of, "%s %s %s\n",
2009                     cmd_to_str(CMD_ADD),
2010                     pt_to_str(PT_OPTIONS),
2011                     optptr->zone_fsopt_opt);
2012         }
2013         (void) fprintf(of, "%s\n", cmd_to_str(CMD_END));
2014         zoncfg_free_fs_option_list(fstab.zone_fs_options);
2015     }
2016     (void) zoncfg_endfsent(handle);
2018     if ((err = zoncfg_setnwifent(handle)) != Z_OK) {
2019         zone_perror(zone, err, B_FALSE);
2020         goto done;
2021     }
2022     while (zoncfg_getnwifent(handle, &nwifstab) == Z_OK) {
2023         (void) fprintf(of, "%s %s\n", cmd_to_str(CMD_ADD),
2024             rt_to_str(RT_NET));
2025         export_prop(of, PT_ADDRESS, nwifstab.zone_nwif_address);
2026         export_prop(of, PT_ALLOWED_ADDRESS,
2027             nwifstab.zone_nwif_allowed_address);
2028         export_prop(of, PT_PHYSICAL, nwifstab.zone_nwif_physical);
2029         export_prop(of, PT_DEFROUTER, nwifstab.zone_nwif_defrouter);
2030         (void) fprintf(of, "%s\n", cmd_to_str(CMD_END));
2031     }
2032     (void) zoncfg_endnwifent(handle);
2034     if ((err = zoncfg_setdevent(handle)) != Z_OK) {

```

```

2035         zone_perror(zone, err, B_FALSE);
2036         goto done;
2037     }
2038     while (zoncfg_getdevent(handle, &devtab) == Z_OK) {
2039         (void) fprintf(of, "%s %s\n", cmd_to_str(CMD_ADD),
2040             rt_to_str(RT_DEVICE));
2041         export_prop(of, PT_MATCH, devtab.zone_dev_match);
2042         (void) fprintf(of, "%s\n", cmd_to_str(CMD_END));
2043     }
2044     (void) zoncfg_enddevent(handle);
2046     if (zoncfg_getmcapent(handle, &mcaptab) == Z_OK) {
2047         char buf[128];
2049         (void) fprintf(of, "%s %s\n", cmd_to_str(CMD_ADD),
2050             rt_to_str(RT_MCAP));
2051         bytes_to_units(mcaptab.zone_physmem_cap, buf, sizeof (buf));
2052         (void) fprintf(of, "%s %s=%s\n", cmd_to_str(CMD_SET),
2053             pt_to_str(PT_PHYSICAL), buf);
2054         (void) fprintf(of, "%s\n", cmd_to_str(CMD_END));
2055     }
2057     if ((err = zoncfg_setrctlent(handle)) != Z_OK) {
2058         zone_perror(zone, err, B_FALSE);
2059         goto done;
2060     }
2061     while (zoncfg_getrctlent(handle, &rctltab) == Z_OK) {
2062         (void) fprintf(of, "%s rctl\n", cmd_to_str(CMD_ADD));
2063         export_prop(of, PT_NAME, rctltab.zone_rctl_name);
2064         for (valptr = rctltab.zone_rctl_valptr; valptr != NULL;
2065             valptr = valptr->zone_rctlval_next) {
2066             fprintf(of, "%s %s (%s=%s,%s=%s,%s=%s)\n",
2067                 cmd_to_str(CMD_ADD), pt_to_str(PT_VALUE),
2068                 pt_to_str(PT_PRIV), valptr->zone_rctlval_priv,
2069                 pt_to_str(PT_LIMIT), valptr->zone_rctlval_limit,
2070                 pt_to_str(PT_ACTION), valptr->zone_rctlval_action);
2071         }
2072         (void) fprintf(of, "%s\n", cmd_to_str(CMD_END));
2073         zoncfg_free_rctl_value_list(rctltab.zone_rctl_valptr);
2074     }
2075     (void) zoncfg_endrctlent(handle);
2077     if ((err = zoncfg_setattrent(handle)) != Z_OK) {
2078         zone_perror(zone, err, B_FALSE);
2079         goto done;
2080     }
2081     while (zoncfg_getattrent(handle, &attrtab) == Z_OK) {
2082         (void) fprintf(of, "%s %s\n", cmd_to_str(CMD_ADD),
2083             rt_to_str(RT_ATTR));
2084         export_prop(of, PT_NAME, attrtab.zone_attr_name);
2085         export_prop(of, PT_TYPE, attrtab.zone_attr_type);
2086         export_prop(of, PT_VALUE, attrtab.zone_attr_value);
2087         (void) fprintf(of, "%s\n", cmd_to_str(CMD_END));
2088     }
2089     (void) zoncfg_endattrent(handle);
2091     if ((err = zoncfg_setdsent(handle)) != Z_OK) {
2092         zone_perror(zone, err, B_FALSE);
2093         goto done;
2094     }
2095     while (zoncfg_getdsent(handle, &dstab) == Z_OK) {
2096         (void) fprintf(of, "%s %s\n", cmd_to_str(CMD_ADD),
2097             rt_to_str(RT_DATASET));
2098         export_prop(of, PT_NAME, dstab.zone_dataset_name);
2099         (void) fprintf(of, "%s\n", cmd_to_str(CMD_END));
2100     }

```

```

2101     (void) zoncfg_enddsent(handle);
2103     if (zoncfg_getpsetent(handle, &psettab) == Z_OK) {
2104         (void) fprintf(of, "%s %s\n", cmd_to_str(CMD_ADD),
2105             rt_to_str(RT_DCPU));
2106         if (strcmp(psettab.zone_ncpu_min, psettab.zone_ncpu_max) == 0)
2107             (void) fprintf(of, "%s %s=%s\n", cmd_to_str(CMD_SET),
2108                 pt_to_str(PT_NCPUS), psettab.zone_ncpu_max);
2109         else
2110             (void) fprintf(of, "%s %s=%s-%s\n", cmd_to_str(CMD_SET),
2111                 pt_to_str(PT_NCPUS), psettab.zone_ncpu_min,
2112                 psettab.zone_ncpu_max);
2113         if (psettab.zone_importance[0] != '\0')
2114             (void) fprintf(of, "%s %s=%s\n", cmd_to_str(CMD_SET),
2115                 pt_to_str(PT_IMPORTANCE), psettab.zone_importance);
2116         (void) fprintf(of, "%s\n", cmd_to_str(CMD_END));
2117     }
2119     if ((err = zoncfg_setadminent(handle)) != Z_OK) {
2120         zone_perror(zone, err, B_FALSE);
2121         goto done;
2122     }
2123     while (zoncfg_getadminent(handle, &admintab) == Z_OK) {
2124         (void) fprintf(of, "%s %s\n", cmd_to_str(CMD_ADD),
2125             rt_to_str(RT_ADMIN));
2126         export_prop(of, PT_USER, admintab.zone_admin_user);
2127         export_prop(of, PT_AUTHS, admintab.zone_admin_auths);
2128         (void) fprintf(of, "%s\n", cmd_to_str(CMD_END));
2129     }
2131 #endif /* ! codereview */
2132     (void) zoncfg_endadminent(handle);
2134     if (zoncfg_getsecflagstent(handle, &secflagstab) == Z_OK) {
2135         (void) fprintf(of, "%s %s\n", cmd_to_str(CMD_ADD),
2136             rt_to_str(RT_SECFLAGS));
2137         export_prop(of, PT_DEFAULT, secflagstab.zone_secflags_default);
2138         export_prop(of, PT_LOWER, secflagstab.zone_secflags_lower);
2139         export_prop(of, PT_UPPER, secflagstab.zone_secflags_upper);
2140         (void) fprintf(of, "%s\n", cmd_to_str(CMD_END));
2141     }
2143 #endif /* ! codereview */
2144     /*
2145      * There is nothing to export for pcap since this resource is just
2146      * a container for an rctl alias.
2147      */
2149 done:
2150     if (need_to_close)
2151         (void) fclose(of);
2152 }
2154 void
2155 exit_func(cmd_t *cmd)
2156 {
2157     int arg, answer;
2158     boolean_t arg_err = B_FALSE;
2160     optind = 0;
2161     while ((arg = getopt(cmd->cmd_argc, cmd->cmd_argv, "?F")) != EOF) {
2162         switch (arg) {
2163             case '?':
2164                 longer_usage(CMD_EXIT);
2165                 arg_err = B_TRUE;
2166                 break;

```

```

2167         case 'F':
2168             force_exit = B_TRUE;
2169             break;
2170         default:
2171             short_usage(CMD_EXIT);
2172             arg_err = B_TRUE;
2173             break;
2174     }
2175 }
2176 if (arg_err)
2177     return;
2179 if (optind < cmd->cmd_argc) {
2180     short_usage(CMD_EXIT);
2181     return;
2182 }
2184 if (global_scope || force_exit) {
2185     time_to_exit = B_TRUE;
2186     return;
2187 }
2189 answer = ask_yesno(B_FALSE, "Resource incomplete; really quit");
2190 if (answer == -1) {
2191     zerr(gettext("Resource incomplete, input "
2192         "not from terminal and -F not specified:\n%s command "
2193         "ignored, but exiting anyway."), cmd_to_str(CMD_EXIT));
2194     exit(Z_ERR);
2195 } else if (answer == 1) {
2196     time_to_exit = B_TRUE;
2197 }
2198 /* (answer == 0) => just return */
2199 }
2201 static int
2202 validate_zonepath_syntax(char *path)
2203 {
2204     if (path[0] != '/') {
2205         zerr(gettext("%s is not an absolute path."), path);
2206         return (Z_ERR);
2207     }
2208     /* If path is all slashes, then fail */
2209     if (strspn(path, "/") == strlen(path)) {
2210         zerr(gettext("/ is not allowed as a %s."),
2211             pt_to_str(PT_ZONEPATH));
2212         return (Z_ERR);
2213     }
2214     return (Z_OK);
2215 }
2217 static void
2218 add_resource(cmd_t *cmd)
2219 {
2220     int type;
2221     struct zone_psettab tmp_psettab;
2222     struct zone_mcaptab tmp_mcaptab;
2223     struct zone_secflagstab tmp_secflagstab;
2224 #endif /* ! codereview */
2225     uint64_t tmp;
2226     uint64_t tmp_mcap;
2227     char pool[MAXNAMELEN];
2229     if ((type = cmd->cmd_res_type) == RT_UNKNOWN) {
2230         long_usage(CMD_ADD, B_TRUE);
2231         goto bad;
2232     }

```

```

2234     switch (type) {
2235     case RT_FS:
2236         bzero(&in_progress_fstab, sizeof (in_progress_fstab));
2237         return;
2238     case RT_NET:
2239         bzero(&in_progress_nwifstab, sizeof (in_progress_nwifstab));
2240         return;
2241     case RT_DEVICE:
2242         bzero(&in_progress_devtab, sizeof (in_progress_devtab));
2243         return;
2244     case RT_RCTL:
2245         if (global_zone)
2246             zerr(gettext("WARNING: Setting a global zone resource "
2247 "control too low could deny\nservice "
2248 "to even the root user; "
2249 "this could render the system impossible\n"
2250 "to administer. Please use caution."));
2251         bzero(&in_progress_rctltab, sizeof (in_progress_rctltab));
2252         return;
2253     case RT_ATTR:
2254         bzero(&in_progress_attrtab, sizeof (in_progress_attrtab));
2255         return;
2256     case RT_DATASET:
2257         bzero(&in_progress_dstab, sizeof (in_progress_dstab));
2258         return;
2259     case RT_DCPU:
2260         /* Make sure there isn't already a cpu-set or cpu-cap entry. */
2261         if (zoncfg_lookup_pset(handle, &tmp_psettab) == Z_OK) {
2262             zerr(gettext("The %s resource already exists."),
2263                 rt_to_str(RT_DCPU));
2264             goto bad;
2265         }
2266         if (zoncfg_get_aliased_rctl(handle, ALIAS_CPUCAP, &tmp) !=
2267             Z_NO_ENTRY) {
2268             zerr(gettext("The %s resource already exists."),
2269                 rt_to_str(RT_PCAP));
2270             goto bad;
2271         }
2272
2273         /* Make sure the pool property isn't set. */
2274         if (zoncfg_get_pool(handle, pool, sizeof (pool)) == Z_OK &&
2275             strlen(pool) > 0) {
2276             zerr(gettext("The %s property is already set. "
2277 "A persistent pool is incompatible with\nthe %s "
2278 "resource."),
2279                 pt_to_str(PT_POOL), rt_to_str(RT_DCPU));
2280             goto bad;
2281         }
2282
2283         bzero(&in_progress_psettab, sizeof (in_progress_psettab));
2284         return;
2285     case RT_PCAP:
2286         /*
2287          * Make sure there isn't already a cpu-set or incompatible
2288          * cpu-cap rctls.
2289          */
2290         if (zoncfg_lookup_pset(handle, &tmp_psettab) == Z_OK) {
2291             zerr(gettext("The %s resource already exists."),
2292                 rt_to_str(RT_DCPU));
2293             goto bad;
2294         }
2295
2296         switch (zoncfg_get_aliased_rctl(handle, ALIAS_CPUCAP, &tmp)) {
2297         case Z_ALIAS_DISALLOW:
2298             zone_perror(rt_to_str(RT_PCAP), Z_ALIAS_DISALLOW,

```

```

2299             B_FALSE);
2300             goto bad;
2301
2302         case Z_OK:
2303             zerr(gettext("The %s resource already exists."),
2304                 rt_to_str(RT_PCAP));
2305             goto bad;
2306
2307         default:
2308             break;
2309     }
2310     return;
2311     case RT_MCAP:
2312         /*
2313          * Make sure there isn't already a mem-cap entry or max-swap
2314          * or max-locked rctl.
2315          */
2316         if (zoncfg_lookup_mcap(handle, &tmp_mcaptab) == Z_OK ||
2317             zoncfg_get_aliased_rctl(handle, ALIAS_MAXSWAP, &tmp_mcap)
2318             == Z_OK ||
2319             zoncfg_get_aliased_rctl(handle, ALIAS_MAXLOCKEDMEM,
2320                 &tmp_mcap) == Z_OK) {
2321             zerr(gettext("The %s resource or a related resource "
2322 "control already exists."), rt_to_str(RT_MCAP));
2323             goto bad;
2324         }
2325         if (global_zone)
2326             zerr(gettext("WARNING: Setting a global zone memory "
2327 "cap too low could deny\nservice "
2328 "to even the root user; "
2329 "this could render the system impossible\n"
2330 "to administer. Please use caution."));
2331         bzero(&in_progress_mcaptab, sizeof (in_progress_mcaptab));
2332         return;
2333     case RT_ADMIN:
2334         bzero(&in_progress_admintab, sizeof (in_progress_admintab));
2335         return;
2336     case RT_SECFLAGS:
2337         /* Make sure we haven't already set this */
2338         if (zoncfg_lookup_secflags(handle, &tmp_secflagstab) == Z_OK)
2339             zerr(gettext("The %s resource already exists."),
2340                 rt_to_str(RT_SECFLAGS));
2341         bzero(&in_progress_secflagstab,
2342             sizeof (in_progress_secflagstab));
2343         return;
2344     #endif /* ! codereview */
2345     default:
2346         zone_perror(rt_to_str(type), Z_NO_RESOURCE_TYPE, B_TRUE);
2347         long_usage(CMD_ADD, B_TRUE);
2348         usage(B_FALSE, HELP_RESOURCES);
2349     }
2350     bad:
2351         global_scope = B_TRUE;
2352         end_op = -1;
2353     }
2354
2355     static void
2356     do_complex_rctl_val(complex_property_ptr_t cp)
2357     {
2358         struct zone_rctlvaltab *rctlvaltab;
2359         complex_property_ptr_t cx;
2360         boolean_t seen_priv = B_FALSE, seen_limit = B_FALSE,
2361             seen_action = B_FALSE;
2362         rctlblk_t *rctlblk;
2363         int err;

```

```

2365     if ((rctlvaltab = alloc_rctlvaltab()) == NULL) {
2366         zone_perror(zone, Z_NOMEM, B_TRUE);
2367         exit(Z_ERR);
2368     }
2369     for (cx = cp; cx != NULL; cx = cx->cp_next) {
2370         switch (cx->cp_type) {
2371             case PT_PRIV:
2372                 if (seen_priv) {
2373                     zerr(gettext("%s already specified"),
2374                         pt_to_str(PT_PRIV));
2375                     goto bad;
2376                 }
2377                 (void) strcpy(rctlvaltab->zone_rctlval_priv,
2378                     cx->cp_value,
2379                     sizeof (rctlvaltab->zone_rctlval_priv));
2380                 seen_priv = B_TRUE;
2381                 break;
2382             case PT_LIMIT:
2383                 if (seen_limit) {
2384                     zerr(gettext("%s already specified"),
2385                         pt_to_str(PT_LIMIT));
2386                     goto bad;
2387                 }
2388                 (void) strcpy(rctlvaltab->zone_rctlval_limit,
2389                     cx->cp_value,
2390                     sizeof (rctlvaltab->zone_rctlval_limit));
2391                 seen_limit = B_TRUE;
2392                 break;
2393             case PT_ACTION:
2394                 if (seen_action) {
2395                     zerr(gettext("%s already specified"),
2396                         pt_to_str(PT_ACTION));
2397                     goto bad;
2398                 }
2399                 (void) strcpy(rctlvaltab->zone_rctlval_action,
2400                     cx->cp_value,
2401                     sizeof (rctlvaltab->zone_rctlval_action));
2402                 seen_action = B_TRUE;
2403                 break;
2404             default:
2405                 zone_perror(pt_to_str(PT_VALUE),
2406                     Z_NO_PROPERTY_TYPE, B_TRUE);
2407                 long_usage(CMD_ADD, B_TRUE);
2408                 usage(B_FALSE, HELP_PROPS);
2409                 zoncfg_free_rctl_value_list(rctlvaltab);
2410                 return;
2411         }
2412     }
2413     if (!seen_priv)
2414         zerr(gettext("%s not specified"), pt_to_str(PT_PRIV));
2415     if (!seen_limit)
2416         zerr(gettext("%s not specified"), pt_to_str(PT_LIMIT));
2417     if (!seen_action)
2418         zerr(gettext("%s not specified"), pt_to_str(PT_ACTION));
2419     if (!seen_priv || !seen_limit || !seen_action)
2420         goto bad;
2421     rctlvaltab->zone_rctlval_next = NULL;
2422     rctlblk = alloca(rctlblk_size());
2423     /*
2424      * Make sure the rctl value looks roughly correct; we won't know if
2425      * it's truly OK until we verify the configuration on the target
2426      * system.
2427      */
2428     if (zoncfg_construct_rctlblk(rctlvaltab, rctlblk) != Z_OK ||
2429         !zoncfg_valid_rctlblk(rctlblk)) {
2430         zerr(gettext("Invalid %s %s specification"), rt_to_str(RT_RCTL),

```

```

2431         pt_to_str(PT_VALUE));
2432         goto bad;
2433     }
2434     err = zoncfg_add_rctl_value(&in_progress_rctltab, rctlvaltab);
2435     if (err != Z_OK)
2436         zone_perror(pt_to_str(PT_VALUE), err, B_TRUE);
2437     return;
2438
2439 bad:
2440     zoncfg_free_rctl_value_list(rctlvaltab);
2441 }
2442
2443 static void
2444 add_property(cmd_t *cmd)
2445 {
2446     char *prop_id;
2447     int err, res_type, prop_type;
2448     property_value_ptr_t pv;
2449     list_property_ptr_t list;
2450
2451     res_type = resource_scope;
2452     prop_type = cmd->cmd_prop_name[0];
2453     if (res_type == RT_UNKNOWN || prop_type == PT_UNKNOWN) {
2454         long_usage(CMD_ADD, B_TRUE);
2455         return;
2456     }
2457
2458     if (cmd->cmd_prop_nv_pairs != 1) {
2459         long_usage(CMD_ADD, B_TRUE);
2460         return;
2461     }
2462
2463     if (initialize(B_TRUE) != Z_OK)
2464         return;
2465
2466     switch (res_type) {
2467     case RT_FS:
2468         if (prop_type != PT_OPTIONS) {
2469             zone_perror(pt_to_str(prop_type), Z_NO_PROPERTY_TYPE,
2470                 B_TRUE);
2471             long_usage(CMD_ADD, B_TRUE);
2472             usage(B_FALSE, HELP_PROPS);
2473             return;
2474         }
2475         pp = cmd->cmd_property_ptr[0];
2476         if (pp->pv_type != PROP_VAL_SIMPLE &&
2477             pp->pv_type != PROP_VAL_LIST) {
2478             zerr(gettext("A %s or %s value was expected here."),
2479                 pvt_to_str(PROP_VAL_SIMPLE),
2480                 pvt_to_str(PROP_VAL_LIST));
2481             saw_error = B_TRUE;
2482             return;
2483         }
2484         if (pp->pv_type == PROP_VAL_SIMPLE) {
2485             if (pp->pv_simple == NULL) {
2486                 long_usage(CMD_ADD, B_TRUE);
2487                 return;
2488             }
2489             prop_id = pp->pv_simple;
2490             err = zoncfg_add_fs_option(&in_progress_fstab,
2491                 prop_id);
2492             if (err != Z_OK)
2493                 zone_perror(pt_to_str(prop_type), err, B_TRUE);
2494         } else {
2495             list_property_ptr_t list;

```

```

2497     for (list = pp->pv_list; list != NULL;
2498          list = list->lp_next) {
2499         prop_id = list->lp_simple;
2500         if (prop_id == NULL)
2501             break;
2502         err = zoncfg_add_fs_option(
2503             &in_progress_fstab, prop_id);
2504         if (err != Z_OK)
2505             zone_perror(pt_to_str(prop_type), err,
2506                 B_TRUE);
2507     }
2508 }
2509 return;
2510 case RT_RCTL:
2511     if (prop_type != PT_VALUE) {
2512         zone_perror(pt_to_str(prop_type), Z_NO_PROPERTY_TYPE,
2513             B_TRUE);
2514         long_usage(CMD_ADD, B_TRUE);
2515         usage(B_FALSE, HELP_PROPS);
2516         return;
2517     }
2518     pp = cmd->cmd_property_ptr[0];
2519     if (pp->pv_type != PROP_VAL_COMPLEX &&
2520         pp->pv_type != PROP_VAL_LIST) {
2521         zerr(gettext("A %s or %s value was expected here."),
2522             pvt_to_str(PROP_VAL_COMPLEX),
2523             pvt_to_str(PROP_VAL_LIST));
2524         saw_error = B_TRUE;
2525         return;
2526     }
2527     if (pp->pv_type == PROP_VAL_COMPLEX) {
2528         do_complex_rctl_val(pp->pv_complex);
2529         return;
2530     }
2531     for (l = pp->pv_list; l != NULL; l = l->lp_next)
2532         do_complex_rctl_val(l->lp_complex);
2533     return;
2534 default:
2535     zone_perror(rt_to_str(res_type), Z_NO_RESOURCE_TYPE, B_TRUE);
2536     long_usage(CMD_ADD, B_TRUE);
2537     usage(B_FALSE, HELP_RESOURCES);
2538     return;
2539 }
2540 }
2542 static boolean_t
2543 gz_invalid_resource(int type)
2544 {
2545     return (global_zone && (type == RT_FS ||
2546         type == RT_NET || type == RT_DEVICE || type == RT_ATTR ||
2547         type == RT_DATASET));
2548 }
2550 static boolean_t
2551 gz_invalid_rt_property(int type)
2552 {
2553     return (global_zone && (type == RT_ZONEPATH || type == RT_ZONEPATH ||
2554         type == RT_AUTOBOOT || type == RT_LIMITPRIV ||
2555         type == RT_BOOTARGS || type == RT_BRAND || type == RT_SCHED ||
2556         type == RT_IPTYPE || type == RT_HOSTID || type == RT_FS_ALLOWED));
2557 }
2559 static boolean_t
2560 gz_invalid_property(int type)
2561 {
2562     return (global_zone && (type == PT_ZONEPATH ||

```

```

2563         type == PT_AUTOBOOT || type == PT_LIMITPRIV ||
2564         type == PT_BOOTARGS || type == PT_BRAND || type == PT_SCHED ||
2565         type == PT_IPTYPE || type == PT_HOSTID || type == PT_FS_ALLOWED));
2566 }
2568 void
2569 add_func(cmd_t *cmd)
2570 {
2571     int arg;
2572     boolean_t arg_err = B_FALSE;
2574     assert(cmd != NULL);
2576     optind = 0;
2577     while ((arg = getopt(cmd->cmd_argc, cmd->cmd_argv, "?") != EOF) {
2578         switch (arg) {
2579             case '?':
2580                 longer_usage(CMD_ADD);
2581                 arg_err = B_TRUE;
2582                 break;
2583             default:
2584                 short_usage(CMD_ADD);
2585                 arg_err = B_TRUE;
2586                 break;
2587         }
2588     }
2589     if (arg_err)
2590         return;
2592     if (optind != cmd->cmd_argc) {
2593         short_usage(CMD_ADD);
2594         return;
2595     }
2597     if (zone_is_read_only(CMD_ADD))
2598         return;
2600     if (initialize(B_TRUE) != Z_OK)
2601         return;
2602     if (global_scope) {
2603         if (gz_invalid_resource(cmd->cmd_res_type)) {
2604             zerr(gettext("Cannot add a %s resource to the "
2605                 "global zone."), rt_to_str(cmd->cmd_res_type));
2606             saw_error = B_TRUE;
2607             return;
2608         }
2610         global_scope = B_FALSE;
2611         resource_scope = cmd->cmd_res_type;
2612         end_op = CMD_ADD;
2613         add_resource(cmd);
2614     } else
2615         add_property(cmd);
2616 }
2618 /*
2619  * This routine has an unusual implementation, because it tries very
2620  * hard to succeed in the face of a variety of failure modes.
2621  * The most common and most vexing occurs when the index file and
2622  * the /etc/zones/<zonename.xml> file are not both present. In
2623  * this case, delete must eradicate as much of the zone state as is left
2624  * so that the user can later create a new zone with the same name.
2625  */
2626 void
2627 delete_func(cmd_t *cmd)
2628 {

```



```

2629     int err, arg, answer;
2630     char line[ZONENAME_MAX + 128]; /* enough to ask a question */
2631     boolean_t force = B_FALSE;
2632     boolean_t arg_err = B_FALSE;

2634     optind = 0;
2635     while ((arg = getopt(cmd->cmd_argc, cmd->cmd_argv, "?F")) != EOF) {
2636         switch (arg) {
2637             case '?':
2638                 longer_usage(CMD_DELETE);
2639                 arg_err = B_TRUE;
2640                 break;
2641             case 'F':
2642                 force = B_TRUE;
2643                 break;
2644             default:
2645                 short_usage(CMD_DELETE);
2646                 arg_err = B_TRUE;
2647                 break;
2648         }
2649     }
2650     if (arg_err)
2651         return;

2653     if (optind != cmd->cmd_argc) {
2654         short_usage(CMD_DELETE);
2655         return;
2656     }

2658     if (zone_is_read_only(CMD_DELETE))
2659         return;

2661     if (!force) {
2662         /*
2663          * Initialize sets up the global called "handle" and warns the
2664          * user if the zone is not configured. In force mode, we don't
2665          * trust that evaluation, and hence skip it. (We don't need the
2666          * handle to be loaded anyway, since zoncfg_destroy is done by
2667          * zonename). However, we also have to take care to emulate the
2668          * messages spit out by initialize; see below.
2669          */
2670         if (initialize(B_TRUE) != Z_OK)
2671             return;

2673         (void) snprintf(line, sizeof (line),
2674             gettext("Are you sure you want to delete zone %s"), zone);
2675         if ((answer = ask_yesno(B_FALSE, line)) == -1) {
2676             zerr(gettext("Input not from terminal and -F not "
2677                 "specified:\n%s command ignored, exiting."),
2678                 cmd_to_str(CMD_DELETE));
2679             exit(Z_ERR);
2680         }
2681         if (answer != 1)
2682             return;
2683     }

2685     /*
2686      * This function removes the authorizations from user_attr
2687      * that correspond to those specified in the configuration
2688      */
2689     if (initialize(B_TRUE) == Z_OK) {
2690         (void) zoncfg_deauthorize_users(handle, zone);
2691     }
2692     if ((err = zoncfg_destroy(zone, force)) != Z_OK) {
2693         if ((err == Z_BAD_ZONE_STATE) && !force) {
2694             zerr(gettext("Zone %s not in %s state; %s not "

```

```

2695         "allowed. Use -F to force %s."),
2696         zone, zone_state_str(ZONE_STATE_CONFIGURED),
2697         cmd_to_str(CMD_DELETE), cmd_to_str(CMD_DELETE));
2698     } else {
2699         zone_perror(zone, err, B_TRUE);
2700     }
2701 }
2702 need_to_commit = B_FALSE;

2704 /*
2705  * Emulate initialize's messaging; if there wasn't a valid handle to
2706  * begin with, then user had typed delete (or delete -F) multiple
2707  * times. So we emit a message.
2708  *
2709  * We only do this in the 'force' case because normally, initialize()
2710  * takes care of this for us.
2711  */
2712 if (force && zoncfg_check_handle(handle) != Z_OK && interactive_mode)
2713     (void) printf(gettext("Use 's' to begin "
2714         "configuring a new zone.\n"), cmd_to_str(CMD_CREATE));

2716 /*
2717  * Time for a new handle: finish the old one off first
2718  * then get a new one properly to avoid leaks.
2719  */
2720 if (got_handle) {
2721     zoncfg_fini_handle(handle);
2722     if ((handle = zoncfg_init_handle()) == NULL) {
2723         zone_perror(execname, Z_NOMEM, B_TRUE);
2724         exit(Z_ERR);
2725     }
2726     if ((err = zoncfg_get_handle(zone, handle)) != Z_OK) {
2727         /* If there was no zone before, that's OK */
2728         if (err != Z_NO_ZONE)
2729             zone_perror(zone, err, B_TRUE);
2730         got_handle = B_FALSE;
2731     }
2732 }
2733 }

2735 static int
2736 fill_in_fstab(cmd_t *cmd, struct zone_fstab *fstab, boolean_t fill_in_only)
2737 {
2738     int err, i;
2739     property_value_ptr_t pp;

2741     if ((err = initialize(B_TRUE)) != Z_OK)
2742         return (err);

2744     bzero(fstab, sizeof (*fstab));
2745     for (i = 0; i < cmd->cmd_prop_nv_pairs; i++) {
2746         pp = cmd->cmd_property_ptr[i];
2747         if (pp->pv_type != PROP_VAL_SIMPLE || pp->pv_simple == NULL) {
2748             zerr(gettext("A simple value was expected here."));
2749             saw_error = B_TRUE;
2750             return (Z_INSUFFICIENT_SPEC);
2751         }
2752         switch (cmd->cmd_prop_name[i]) {
2753             case PT_DIR:
2754                 (void) strlcpy(fstab->zone_fs_dir, pp->pv_simple,
2755                     sizeof (fstab->zone_fs_dir));
2756                 break;
2757             case PT_SPECIAL:
2758                 (void) strlcpy(fstab->zone_fs_special, pp->pv_simple,
2759                     sizeof (fstab->zone_fs_special));
2760                 break;

```

```

2761     case PT_RAW:
2762         (void) strncpy(fstab->zone_fs_raw, pp->pv_simple,
2763             sizeof (fstab->zone_fs_raw));
2764         break;
2765     case PT_TYPE:
2766         (void) strncpy(fstab->zone_fs_type, pp->pv_simple,
2767             sizeof (fstab->zone_fs_type));
2768         break;
2769     default:
2770         zone_perror(pt_to_str(cmd->cmd_prop_name[i]),
2771             Z_NO_PROPERTY_TYPE, B_TRUE);
2772         return (Z_INSUFFICIENT_SPEC);
2773     }
2774 }
2775 if (fill_in_only)
2776     return (Z_OK);
2777 return (zonecfg_lookup_filesystem(handle, fstab));
2778 }

2780 static int
2781 fill_in_nwifstab(cmd_t *cmd, struct zone_nwifstab *nwifstab,
2782     boolean_t fill_in_only)
2783 {
2784     int err, i;
2785     property_value_ptr_t pp;

2787     if ((err = initialize(B_TRUE)) != Z_OK)
2788         return (err);

2790     bzero(nwifstab, sizeof (*nwifstab));
2791     for (i = 0; i < cmd->cmd_prop_nv_pairs; i++) {
2792         pp = cmd->cmd_property_ptr[i];
2793         if (pp->pv_type != PROP_VAL_SIMPLE || pp->pv_simple == NULL) {
2794             zerr(gettext("A simple value was expected here.));
2795             saw_error = B_TRUE;
2796             return (Z_INSUFFICIENT_SPEC);
2797         }
2798         switch (cmd->cmd_prop_name[i]) {
2799         case PT_ADDRESS:
2800             (void) strncpy(nwifstab->zone_nwif_address,
2801                 pp->pv_simple, sizeof (nwifstab->zone_nwif_address));
2802             break;
2803         case PT_ALLOWED_ADDRESS:
2804             (void) strncpy(nwifstab->zone_nwif_allowed_address,
2805                 pp->pv_simple,
2806                 sizeof (nwifstab->zone_nwif_allowed_address));
2807             break;
2808         case PT_PHYSICAL:
2809             (void) strncpy(nwifstab->zone_nwif_physical,
2810                 pp->pv_simple,
2811                 sizeof (nwifstab->zone_nwif_physical));
2812             break;
2813         case PT_DEFROUTER:
2814             (void) strncpy(nwifstab->zone_nwif_defrouter,
2815                 pp->pv_simple,
2816                 sizeof (nwifstab->zone_nwif_defrouter));
2817             break;
2818         default:
2819             zone_perror(pt_to_str(cmd->cmd_prop_name[i]),
2820                 Z_NO_PROPERTY_TYPE, B_TRUE);
2821             return (Z_INSUFFICIENT_SPEC);
2822         }
2823     }
2824     if (fill_in_only)
2825         return (Z_OK);
2826     err = zonecfg_lookup_nwif(handle, nwifstab);

```

```

2827         return (err);
2828     }

2830 static int
2831 fill_in_devtab(cmd_t *cmd, struct zone_devtab *devtab, boolean_t fill_in_only)
2832 {
2833     int err, i;
2834     property_value_ptr_t pp;

2836     if ((err = initialize(B_TRUE)) != Z_OK)
2837         return (err);

2839     bzero(devtab, sizeof (*devtab));
2840     for (i = 0; i < cmd->cmd_prop_nv_pairs; i++) {
2841         pp = cmd->cmd_property_ptr[i];
2842         if (pp->pv_type != PROP_VAL_SIMPLE || pp->pv_simple == NULL) {
2843             zerr(gettext("A simple value was expected here.));
2844             saw_error = B_TRUE;
2845             return (Z_INSUFFICIENT_SPEC);
2846         }
2847         switch (cmd->cmd_prop_name[i]) {
2848         case PT_MATCH:
2849             (void) strncpy(devtab->zone_dev_match, pp->pv_simple,
2850                 sizeof (devtab->zone_dev_match));
2851             break;
2852         default:
2853             zone_perror(pt_to_str(cmd->cmd_prop_name[i]),
2854                 Z_NO_PROPERTY_TYPE, B_TRUE);
2855             return (Z_INSUFFICIENT_SPEC);
2856         }
2857     }
2858     if (fill_in_only)
2859         return (Z_OK);
2860     err = zonecfg_lookup_dev(handle, devtab);
2861     return (err);
2862 }

2864 static int
2865 fill_in_rctltab(cmd_t *cmd, struct zone_rctltab *rctltab,
2866     boolean_t fill_in_only)
2867 {
2868     int err, i;
2869     property_value_ptr_t pp;

2871     if ((err = initialize(B_TRUE)) != Z_OK)
2872         return (err);

2874     bzero(rctltab, sizeof (*rctltab));
2875     for (i = 0; i < cmd->cmd_prop_nv_pairs; i++) {
2876         pp = cmd->cmd_property_ptr[i];
2877         if (pp->pv_type != PROP_VAL_SIMPLE || pp->pv_simple == NULL) {
2878             zerr(gettext("A simple value was expected here.));
2879             saw_error = B_TRUE;
2880             return (Z_INSUFFICIENT_SPEC);
2881         }
2882         switch (cmd->cmd_prop_name[i]) {
2883         case PT_NAME:
2884             (void) strncpy(rctltab->zone_rctl_name, pp->pv_simple,
2885                 sizeof (rctltab->zone_rctl_name));
2886             break;
2887         default:
2888             zone_perror(pt_to_str(cmd->cmd_prop_name[i]),
2889                 Z_NO_PROPERTY_TYPE, B_TRUE);
2890             return (Z_INSUFFICIENT_SPEC);
2891         }
2892     }

```

```

2893     if (fill_in_only)
2894         return (Z_OK);
2895     err = zoncfg_lookup_rctl(handle, rctltab);
2896     return (err);
2897 }

2899 static int
2900 fill_in_attrtab(cmd_t *cmd, struct zone_attrtab *attrtab,
2901               boolean_t fill_in_only)
2902 {
2903     int err, i;
2904     property_value_ptr_t pp;

2906     if ((err = initialize(B_TRUE)) != Z_OK)
2907         return (err);

2909     bzero(attrtab, sizeof (*attrtab));
2910     for (i = 0; i < cmd->cmd_prop_nv_pairs; i++) {
2911         pp = cmd->cmd_property_ptr[i];
2912         if (pp->pv_type != PROP_VAL_SIMPLE || pp->pv_simple == NULL) {
2913             zerr(gettext("A simple value was expected here.));
2914             saw_error = B_TRUE;
2915             return (Z_INSUFFICIENT_SPEC);
2916         }
2917         switch (cmd->cmd_prop_name[i]) {
2918             case PT_NAME:
2919                 (void) strncpy(attrtab->zone_attr_name, pp->pv_simple,
2920                               sizeof (attrtab->zone_attr_name));
2921                 break;
2922             case PT_TYPE:
2923                 (void) strncpy(attrtab->zone_attr_type, pp->pv_simple,
2924                               sizeof (attrtab->zone_attr_type));
2925                 break;
2926             case PT_VALUE:
2927                 (void) strncpy(attrtab->zone_attr_value, pp->pv_simple,
2928                               sizeof (attrtab->zone_attr_value));
2929                 break;
2930             default:
2931                 zone_perror(pt_to_str(cmd->cmd_prop_name[i]),
2932                             Z_NO_PROPERTY_TYPE, B_TRUE);
2933                 return (Z_INSUFFICIENT_SPEC);
2934         }
2935     }
2936     if (fill_in_only)
2937         return (Z_OK);
2938     err = zoncfg_lookup_attr(handle, attrtab);
2939     return (err);
2940 }

2942 static int
2943 fill_in_dstab(cmd_t *cmd, struct zone_dstab *dstab, boolean_t fill_in_only)
2944 {
2945     int err, i;
2946     property_value_ptr_t pp;

2948     if ((err = initialize(B_TRUE)) != Z_OK)
2949         return (err);

2951     dstab->zone_dataset_name[0] = '\0';
2952     for (i = 0; i < cmd->cmd_prop_nv_pairs; i++) {
2953         pp = cmd->cmd_property_ptr[i];
2954         if (pp->pv_type != PROP_VAL_SIMPLE || pp->pv_simple == NULL) {
2955             zerr(gettext("A simple value was expected here.));
2956             saw_error = B_TRUE;
2957             return (Z_INSUFFICIENT_SPEC);
2958         }

```

```

2959     switch (cmd->cmd_prop_name[i]) {
2960         case PT_NAME:
2961             (void) strncpy(dstab->zone_dataset_name, pp->pv_simple,
2962                           sizeof (dstab->zone_dataset_name));
2963             break;
2964         default:
2965             zone_perror(pt_to_str(cmd->cmd_prop_name[i]),
2966                         Z_NO_PROPERTY_TYPE, B_TRUE);
2967             return (Z_INSUFFICIENT_SPEC);
2968     }
2969 }
2970 if (fill_in_only)
2971     return (Z_OK);
2972 return (zoncfg_lookup_ds(handle, dstab));
2973 }

2975 static int
2976 fill_in_admintab(cmd_t *cmd, struct zone_admintab *admintab,
2977                boolean_t fill_in_only)
2978 {
2979     int err, i;
2980     property_value_ptr_t pp;

2982     if ((err = initialize(B_TRUE)) != Z_OK)
2983         return (err);

2985     bzero(admintab, sizeof (*admintab));
2986     for (i = 0; i < cmd->cmd_prop_nv_pairs; i++) {
2987         pp = cmd->cmd_property_ptr[i];
2988         if (pp->pv_type != PROP_VAL_SIMPLE || pp->pv_simple == NULL) {
2989             zerr(gettext("A simple value was expected here.));
2990             saw_error = B_TRUE;
2991             return (Z_INSUFFICIENT_SPEC);
2992         }
2993         switch (cmd->cmd_prop_name[i]) {
2994             case PT_USER:
2995                 (void) strncpy(admintab->zone_admin_user, pp->pv_simple,
2996                               sizeof (admintab->zone_admin_user));
2997                 break;
2998             case PT_AUTHS:
2999                 (void) strncpy(admintab->zone_admin_auths,
3000                               pp->pv_simple, sizeof (admintab->zone_admin_auths));
3001                 break;
3002             default:
3003                 zone_perror(pt_to_str(cmd->cmd_prop_name[i]),
3004                             Z_NO_PROPERTY_TYPE, B_TRUE);
3005                 return (Z_INSUFFICIENT_SPEC);
3006         }
3007     }
3008     if (fill_in_only)
3009         return (Z_OK);
3010     err = zoncfg_lookup_admin(handle, admintab);
3011     return (err);
3012 }

3014 static int
3015 fill_in_secflagstab(cmd_t *cmd, struct zone_secflagstab *secflagstab,
3016                   boolean_t fill_in_only)
3017 {
3018     int err, i;
3019     property_value_ptr_t pp;

3021     if ((err = initialize(B_TRUE)) != Z_OK)
3022         return (err);

3024     bzero(secflagstab, sizeof (*secflagstab));

```

```

3025     for (i = 0; i < cmd->cmd_prop_nv_pairs; i++) {
3026         pp = cmd->cmd_property_ptr[i];
3027         if (pp->pv_type != PROP_VAL_SIMPLE || pp->pv_simple == NULL) {
3028             zerr(gettext("A simple value was expected here."));
3029             saw_error = B_TRUE;
3030             return (Z_INSUFFICIENT_SPEC);
3031         }
3032         switch (cmd->cmd_prop_name[i]) {
3033             case PT_DEFAULT:
3034                 (void) strcpy(secflagstab->zone_secflags_default,
3035                     pp->pv_simple,
3036                     sizeof (secflagstab->zone_secflags_default));
3037                 break;
3038             case PT_LOWER:
3039                 (void) strcpy(secflagstab->zone_secflags_lower,
3040                     pp->pv_simple,
3041                     sizeof (secflagstab->zone_secflags_lower));
3042                 break;
3043             case PT_UPPER:
3044                 (void) strcpy(secflagstab->zone_secflags_upper,
3045                     pp->pv_simple,
3046                     sizeof (secflagstab->zone_secflags_upper));
3047                 break;
3048             default:
3049                 zone_perror(pt_to_str(cmd->cmd_prop_name[i]),
3050                     Z_NO_PROPERTY_TYPE, B_TRUE);
3051                 return (Z_INSUFFICIENT_SPEC);
3052         }
3053     }
3054     if (fill_in_only)
3055         return (Z_OK);
3057     err = zoncfg_lookup_secflags(handle, secflagstab);
3059     return (err);
3060 }
3062 #endif /* ! codereview */
3063 static void
3064 remove_aliased_rctl(int type, char *name)
3065 {
3066     int err;
3067     uint64_t tmp;
3069     if ((err = zoncfg_get_aliased_rctl(handle, name, &tmp)) != Z_OK) {
3070         zerr("%s %s: %s", cmd_to_str(CMD_CLEAR), pt_to_str(type),
3071             zoncfg_strerror(err));
3072         saw_error = B_TRUE;
3073         return;
3074     }
3075     if ((err = zoncfg_rm_aliased_rctl(handle, name)) != Z_OK) {
3076         zerr("%s %s: %s", cmd_to_str(CMD_CLEAR), pt_to_str(type),
3077             zoncfg_strerror(err));
3078         saw_error = B_TRUE;
3079     } else {
3080         need_to_commit = B_TRUE;
3081     }
3082 }
3084 static boolean_t
3085 prompt_remove_resource(cmd_t *cmd, char *rsrc)
3086 {
3087     int num;
3088     int answer;
3089     int arg;
3090     boolean_t force = B_FALSE;

```

```

3091     char prompt[128];
3092     boolean_t arg_err = B_FALSE;
3094     optind = 0;
3095     while ((arg = getopt(cmd->cmd_argc, cmd->cmd_argv, "F")) != EOF) {
3096         switch (arg) {
3097             case 'F':
3098                 force = B_TRUE;
3099                 break;
3100             default:
3101                 arg_err = B_TRUE;
3102                 break;
3103         }
3104     }
3105     if (arg_err)
3106         return (B_FALSE);
3109     num = zoncfg_num_resources(handle, rsrc);
3111     if (num == 0) {
3112         z_cmd_rt_perror(CMD_REMOVE, cmd->cmd_res_type, Z_NO_ENTRY,
3113             B_TRUE);
3114         return (B_FALSE);
3115     }
3116     if (num > 1 && !force) {
3117         if (!interactive_mode) {
3118             zerr(gettext("There are multiple instances of this "
3119                 "resource. Either qualify the resource to\n"
3120                 "remove a single instance or use the -F option to "
3121                 "remove all instances."));
3122             saw_error = B_TRUE;
3123             return (B_FALSE);
3124         }
3125         (void) snprintf(prompt, sizeof (prompt), gettext(
3126             "Are you sure you want to remove ALL '%s' resources"),
3127             rsrc);
3128         answer = ask_yesno(B_FALSE, prompt);
3129         if (answer == -1) {
3130             zerr(gettext("Resource incomplete."));
3131             return (B_FALSE);
3132         }
3133         if (answer != 1)
3134             return (B_FALSE);
3135     }
3136     return (B_TRUE);
3137 }
3139 static void
3140 remove_fs(cmd_t *cmd)
3141 {
3142     int err;
3144     /* traditional, qualified fs removal */
3145     if (cmd->cmd_prop_nv_pairs > 0) {
3146         struct zone_fstab fstab;
3148         if ((err = fill_in_fstab(cmd, &fstab, B_FALSE)) != Z_OK) {
3149             z_cmd_rt_perror(CMD_REMOVE, RT_FS, err, B_TRUE);
3150             return;
3151         }
3152         if ((err = zoncfg_delete_filesystem(handle, &fstab)) != Z_OK)
3153             z_cmd_rt_perror(CMD_REMOVE, RT_FS, err, B_TRUE);
3154     } else
3155         need_to_commit = B_TRUE;
3156     zoncfg_free_fs_option_list(fstab.zone_fs_options);

```

```

3157         return;
3158     }
3160     /*
3161     * unqualified fs removal.  remove all fs's but prompt if more
3162     * than one.
3163     */
3164     if (!prompt_remove_resource(cmd, "fs"))
3165         return;
3167     if ((err = zonecfg_del_all_resources(handle, "fs")) != Z_OK)
3168         z_cmd_rt_perror(CMD_REMOVE, RT_FS, err, B_TRUE);
3169     else
3170         need_to_commit = B_TRUE;
3171 }
3173 static void
3174 remove_net(cmd_t *cmd)
3175 {
3176     int err;
3178     /* traditional, qualified net removal */
3179     if (cmd->cmd_prop_nv_pairs > 0) {
3180         struct zone_nwifstab nwifstab;
3182         if ((err = fill_in_nwifstab(cmd, &nwifstab, B_FALSE)) != Z_OK) {
3183             z_cmd_rt_perror(CMD_REMOVE, RT_NET, err, B_TRUE);
3184             return;
3185         }
3186         if ((err = zonecfg_delete_nwif(handle, &nwifstab)) != Z_OK)
3187             z_cmd_rt_perror(CMD_REMOVE, RT_NET, err, B_TRUE);
3188         else
3189             need_to_commit = B_TRUE;
3190         return;
3191     }
3193     /*
3194     * unqualified net removal.  remove all nets but prompt if more
3195     * than one.
3196     */
3197     if (!prompt_remove_resource(cmd, "net"))
3198         return;
3200     if ((err = zonecfg_del_all_resources(handle, "net")) != Z_OK)
3201         z_cmd_rt_perror(CMD_REMOVE, RT_NET, err, B_TRUE);
3202     else
3203         need_to_commit = B_TRUE;
3204 }
3206 static void
3207 remove_device(cmd_t *cmd)
3208 {
3209     int err;
3211     /* traditional, qualified device removal */
3212     if (cmd->cmd_prop_nv_pairs > 0) {
3213         struct zone_devtab devtab;
3215         if ((err = fill_in_devtab(cmd, &devtab, B_FALSE)) != Z_OK) {
3216             z_cmd_rt_perror(CMD_REMOVE, RT_DEVICE, err, B_TRUE);
3217             return;
3218         }
3219         if ((err = zonecfg_delete_dev(handle, &devtab)) != Z_OK)
3220             z_cmd_rt_perror(CMD_REMOVE, RT_DEVICE, err, B_TRUE);
3221         else
3222             need_to_commit = B_TRUE;

```

```

3223         return;
3224     }
3226     /*
3227     * unqualified device removal.  remove all devices but prompt if more
3228     * than one.
3229     */
3230     if (!prompt_remove_resource(cmd, "device"))
3231         return;
3233     if ((err = zonecfg_del_all_resources(handle, "device")) != Z_OK)
3234         z_cmd_rt_perror(CMD_REMOVE, RT_DEVICE, err, B_TRUE);
3235     else
3236         need_to_commit = B_TRUE;
3237 }
3239 static void
3240 remove_attr(cmd_t *cmd)
3241 {
3242     int err;
3244     /* traditional, qualified attr removal */
3245     if (cmd->cmd_prop_nv_pairs > 0) {
3246         struct zone_attrtab attrtab;
3248         if ((err = fill_in_attrtab(cmd, &attrtab, B_FALSE)) != Z_OK) {
3249             z_cmd_rt_perror(CMD_REMOVE, RT_ATTR, err, B_TRUE);
3250             return;
3251         }
3252         if ((err = zonecfg_delete_attr(handle, &attrtab)) != Z_OK)
3253             z_cmd_rt_perror(CMD_REMOVE, RT_ATTR, err, B_TRUE);
3254         else
3255             need_to_commit = B_TRUE;
3256         return;
3257     }
3259     /*
3260     * unqualified attr removal.  remove all attrs but prompt if more
3261     * than one.
3262     */
3263     if (!prompt_remove_resource(cmd, "attr"))
3264         return;
3266     if ((err = zonecfg_del_all_resources(handle, "attr")) != Z_OK)
3267         z_cmd_rt_perror(CMD_REMOVE, RT_ATTR, err, B_TRUE);
3268     else
3269         need_to_commit = B_TRUE;
3270 }
3272 static void
3273 remove_dataset(cmd_t *cmd)
3274 {
3275     int err;
3277     /* traditional, qualified dataset removal */
3278     if (cmd->cmd_prop_nv_pairs > 0) {
3279         struct zone_dstab dstab;
3281         if ((err = fill_in_dstab(cmd, &dstab, B_FALSE)) != Z_OK) {
3282             z_cmd_rt_perror(CMD_REMOVE, RT_DATASET, err, B_TRUE);
3283             return;
3284         }
3285         if ((err = zonecfg_delete_ds(handle, &dstab)) != Z_OK)
3286             z_cmd_rt_perror(CMD_REMOVE, RT_DATASET, err, B_TRUE);
3287         else
3288             need_to_commit = B_TRUE;

```

```

3289         return;
3290     }

3292     /*
3293     * unqualified dataset removal.  remove all datasets but prompt if more
3294     * than one.
3295     */
3296     if (!prompt_remove_resource(cmd, "dataset"))
3297         return;

3299     if ((err = zonecfg_del_all_resources(handle, "dataset")) != Z_OK)
3300         z_cmd_rt_perror(CMD_REMOVE, RT_DATASET, err, B_TRUE);
3301     else
3302         need_to_commit = B_TRUE;
3303 }

3305 static void
3306 remove_rctl(cmd_t *cmd)
3307 {
3308     int err;

3310     /* traditional, qualified rctl removal */
3311     if (cmd->cmd_prop_nv_pairs > 0) {
3312         struct zone_rctltab rctltab;

3314         if ((err = fill_in_rctltab(cmd, &rctltab, B_FALSE)) != Z_OK) {
3315             z_cmd_rt_perror(CMD_REMOVE, RT_RCTL, err, B_TRUE);
3316         }
3317         if ((err = zonecfg_delete_rctl(handle, &rctltab)) != Z_OK)
3318             z_cmd_rt_perror(CMD_REMOVE, RT_RCTL, err, B_TRUE);
3319         else
3320             need_to_commit = B_TRUE;
3321         zonecfg_free_rctl_value_list(rctltab.zone_rctl_valptr);
3322         return;
3323     }

3326     /*
3327     * unqualified rctl removal.  remove all rctls but prompt if more
3328     * than one.
3329     */
3330     if (!prompt_remove_resource(cmd, "rctl"))
3331         return;

3333     if ((err = zonecfg_del_all_resources(handle, "rctl")) != Z_OK)
3334         z_cmd_rt_perror(CMD_REMOVE, RT_RCTL, err, B_TRUE);
3335     else
3336         need_to_commit = B_TRUE;
3337 }

3339 static void
3340 remove_pset()
3341 {
3342     int err;
3343     struct zone_psettab psettab;

3345     if ((err = zonecfg_lookup_pset(handle, &psettab)) != Z_OK) {
3346         z_cmd_rt_perror(CMD_REMOVE, RT_DCPU, err, B_TRUE);
3347     }
3348     if ((err = zonecfg_delete_pset(handle)) != Z_OK)
3349         z_cmd_rt_perror(CMD_REMOVE, RT_DCPU, err, B_TRUE);
3350     else
3351         need_to_commit = B_TRUE;
3352 }
3353 }

```

```

3355 static void
3356 remove_pcap()
3357 {
3358     int err;
3359     uint64_t tmp;

3361     if (zonecfg_get_aliased_rctl(handle, ALIAS_CPUCAP, &tmp) != Z_OK) {
3362         zerr("%s %s: %s", cmd_to_str(CMD_REMOVE), rt_to_str(RT_PCAP),
3363             zonecfg_strerror(Z_NO_RESOURCE_TYPE));
3364         saw_error = B_TRUE;
3365         return;
3366     }

3368     if ((err = zonecfg_rm_aliased_rctl(handle, ALIAS_CPUCAP)) != Z_OK)
3369         z_cmd_rt_perror(CMD_REMOVE, RT_PCAP, err, B_TRUE);
3370     else
3371         need_to_commit = B_TRUE;
3372 }

3374 static void
3375 remove_mcap()
3376 {
3377     int err, res1, res2, res3;
3378     uint64_t tmp;
3379     struct zone_mcaptab mcaptab;
3380     boolean_t revert = B_FALSE;

3382     res1 = zonecfg_lookup_mcap(handle, &mcaptab);
3383     res2 = zonecfg_get_aliased_rctl(handle, ALIAS_MAXSWAP, &tmp);
3384     res3 = zonecfg_get_aliased_rctl(handle, ALIAS_MAXLOCKEDMEM, &tmp);

3386     /* if none of these exist, there is no resource to remove */
3387     if (res1 != Z_OK && res2 != Z_OK && res3 != Z_OK) {
3388         zerr("%s %s: %s", cmd_to_str(CMD_REMOVE), rt_to_str(RT_MCAP),
3389             zonecfg_strerror(Z_NO_RESOURCE_TYPE));
3390         saw_error = B_TRUE;
3391         return;
3392     }
3393     if (res1 == Z_OK) {
3394         if ((err = zonecfg_delete_mcap(handle)) != Z_OK) {
3395             z_cmd_rt_perror(CMD_REMOVE, RT_MCAP, err, B_TRUE);
3396             revert = B_TRUE;
3397         } else {
3398             need_to_commit = B_TRUE;
3399         }
3400     }
3401     if (res2 == Z_OK) {
3402         if ((err = zonecfg_rm_aliased_rctl(handle, ALIAS_MAXSWAP))
3403             != Z_OK) {
3404             z_cmd_rt_perror(CMD_REMOVE, RT_MCAP, err, B_TRUE);
3405             revert = B_TRUE;
3406         } else {
3407             need_to_commit = B_TRUE;
3408         }
3409     }
3410     if (res3 == Z_OK) {
3411         if ((err = zonecfg_rm_aliased_rctl(handle, ALIAS_MAXLOCKEDMEM))
3412             != Z_OK) {
3413             z_cmd_rt_perror(CMD_REMOVE, RT_MCAP, err, B_TRUE);
3414             revert = B_TRUE;
3415         } else {
3416             need_to_commit = B_TRUE;
3417         }
3418     }

3420     if (revert)

```

```

3421         need_to_commit = B_FALSE;
3422     }

3424 static void
3425 remove_admin(cmd_t *cmd)
3426 {
3427     int err;

3429     /* traditional, qualified attr removal */
3430     if (cmd->cmd_prop_nv_pairs > 0) {
3431         struct zone_adminTAB adminTAB;

3433         if ((err = fill_in_adminTAB(cmd, &adminTAB, B_FALSE)) != Z_OK) {
3434             z_cmd_rt_perror(CMD_REMOVE, RT_ADMIN,
3435                 err, B_TRUE);
3436             return;
3437         }
3438         if ((err = zoncfg_delete_admin(handle, &adminTAB,
3439             zone))
3440             != Z_OK)
3441             z_cmd_rt_perror(CMD_REMOVE, RT_ADMIN,
3442                 err, B_TRUE);
3443         else
3444             need_to_commit = B_TRUE;
3445         return;
3446     } else {
3447         /*
3448          * unqualified admin removal.
3449          * remove all admins but prompt if more
3450          * than one.
3451          */
3452         if (!prompt_remove_resource(cmd, "admin"))
3453             return;

3455         if ((err = zoncfg_delete_admins(handle, zone))
3456             != Z_OK)
3457             z_cmd_rt_perror(CMD_REMOVE, RT_ADMIN,
3458                 err, B_TRUE);
3459         else
3460             need_to_commit = B_TRUE;
3461     }
3462 }

3464 static void
3465 remove_secflags()
3466 {
3467     int err;
3468     struct zone_secflagTAB sectab = { 0 };

3470     if (zoncfg_lookup_secflags(handle, &sectab) != Z_OK) {
3471         zerr("%s %s: %s", cmd_to_str(CMD_REMOVE),
3472             rt_to_str(RT_SECFLAGS),
3473             zoncfg_strerror(Z_NO_RESOURCE_TYPE));
3474         return;
3475     }

3477     if ((err = zoncfg_delete_secflags(handle, &sectab)) != Z_OK) {
3478         z_cmd_rt_perror(CMD_REMOVE, RT_SECFLAGS, err, B_TRUE);
3479         return;
3480     }

3482     need_to_commit = B_TRUE;
3483 }

3485 static void
3486 #endif /* ! codereview */

```

```

3487 remove_resource(cmd_t *cmd)
3488 {
3489     int type;
3490     int arg;
3491     boolean_t arg_err = B_FALSE;

3493     if ((type = cmd->cmd_res_type) == RT_UNKNOWN) {
3494         long_usage(CMD_REMOVE, B_TRUE);
3495         return;
3496     }

3498     optind = 0;
3499     while ((arg = getopt(cmd->cmd_argc, cmd->cmd_argv, "?F")) != EOF) {
3500         switch (arg) {
3501             case '?':
3502                 longer_usage(CMD_REMOVE);
3503                 arg_err = B_TRUE;
3504                 break;
3505             case 'F':
3506                 break;
3507             default:
3508                 short_usage(CMD_REMOVE);
3509                 arg_err = B_TRUE;
3510                 break;
3511         }
3512     }
3513     if (arg_err)
3514         return;

3516     if (initialize(B_TRUE) != Z_OK)
3517         return;

3519     switch (type) {
3520     case RT_FS:
3521         remove_fs(cmd);
3522         return;
3523     case RT_NET:
3524         remove_net(cmd);
3525         return;
3526     case RT_DEVICE:
3527         remove_device(cmd);
3528         return;
3529     case RT_RCTL:
3530         remove_rctl(cmd);
3531         return;
3532     case RT_ATTR:
3533         remove_attr(cmd);
3534         return;
3535     case RT_DATASET:
3536         remove_dataset(cmd);
3537         return;
3538     case RT_DCPU:
3539         remove_pset();
3540         return;
3541     case RT_PCAP:
3542         remove_pcap();
3543         return;
3544     case RT_MCAP:
3545         remove_mcap();
3546         return;
3547     case RT_ADMIN:
3548         remove_admin(cmd);
3549         return;
3550     case RT_SECFLAGS:
3551         remove_secflags();
3552         return;

```

```

3553 #endif /* ! codereview */
3554     default:
3555         zone_perror(rt_to_str(type), Z_NO_RESOURCE_TYPE, B_TRUE);
3556         long_usage(CMD_REMOVE, B_TRUE);
3557         usage(B_FALSE, HELP_RESOURCES);
3558         return;
3559     }
3560 }

3562 static void
3563 remove_property(cmd_t *cmd)
3564 {
3565     char *prop_id;
3566     int err, res_type, prop_type;
3567     property_value_ptr_t pp;
3568     struct zone_rctlvaltab *rctlvaltab;
3569     complex_property_ptr_t cx;

3571     res_type = resource_scope;
3572     prop_type = cmd->cmd_prop_name[0];
3573     if (res_type == RT_UNKNOWN || prop_type == PT_UNKNOWN) {
3574         long_usage(CMD_REMOVE, B_TRUE);
3575         return;
3576     }

3578     if (cmd->cmd_prop_nv_pairs != 1) {
3579         long_usage(CMD_ADD, B_TRUE);
3580         return;
3581     }

3583     if (initialize(B_TRUE) != Z_OK)
3584         return;

3586     switch (res_type) {
3587     case RT_FS:
3588         if (prop_type != PT_OPTIONS) {
3589             zone_perror(pt_to_str(prop_type), Z_NO_PROPERTY_TYPE,
3590                 B_TRUE);
3591             long_usage(CMD_REMOVE, B_TRUE);
3592             usage(B_FALSE, HELP_PROPS);
3593             return;
3594         }
3595         pp = cmd->cmd_property_ptr[0];
3596         if (pp->pv_type == PROP_VAL_COMPLEX) {
3597             zerr(gettext("A %s or %s value was expected here."),
3598                 pvt_to_str(PROP_VAL_SIMPLE),
3599                 pvt_to_str(PROP_VAL_LIST));
3600             saw_error = B_TRUE;
3601             return;
3602         }
3603         if (pp->pv_type == PROP_VAL_SIMPLE) {
3604             if (pp->pv_simple == NULL) {
3605                 long_usage(CMD_ADD, B_TRUE);
3606                 return;
3607             }
3608             prop_id = pp->pv_simple;
3609             err = zoncfg_remove_fs_option(&in_progress_fstab,
3610                 prop_id);
3611             if (err != Z_OK)
3612                 zone_perror(pt_to_str(prop_type), err, B_TRUE);
3613         } else {
3614             list_property_ptr_t list;

3616             for (list = pp->pv_list; list != NULL;
3617                 list = list->lp_next) {
3618                 prop_id = list->lp_simple;

```

```

3619         if (prop_id == NULL)
3620             break;
3621         err = zoncfg_remove_fs_option(
3622             &in_progress_fstab, prop_id);
3623         if (err != Z_OK)
3624             zone_perror(pt_to_str(prop_type), err,
3625                 B_TRUE);
3626     }
3627 }
3628 return;
3629 case RT_RCTL:
3630     if (prop_type != PT_VALUE) {
3631         zone_perror(pt_to_str(prop_type), Z_NO_PROPERTY_TYPE,
3632             B_TRUE);
3633         long_usage(CMD_REMOVE, B_TRUE);
3634         usage(B_FALSE, HELP_PROPS);
3635         return;
3636     }
3637     pp = cmd->cmd_property_ptr[0];
3638     if (pp->pv_type != PROP_VAL_COMPLEX) {
3639         zerr(gettext("A %s value was expected here."),
3640             pvt_to_str(PROP_VAL_COMPLEX));
3641         saw_error = B_TRUE;
3642         return;
3643     }
3644     if ((rctlvaltab = alloc_rctlvaltab()) == NULL) {
3645         zone_perror(zone, Z_NOMEM, B_TRUE);
3646         exit(Z_ERR);
3647     }
3648     for (cx = pp->pv_complex; cx != NULL; cx = cx->cp_next) {
3649         switch (cx->cp_type) {
3650         case PT_PRIV:
3651             (void) strcpy(rctlvaltab->zone_rctlval_priv,
3652                 cx->cp_value,
3653                 sizeof(rctlvaltab->zone_rctlval_priv));
3654             break;
3655         case PT_LIMIT:
3656             (void) strcpy(rctlvaltab->zone_rctlval_limit,
3657                 cx->cp_value,
3658                 sizeof(rctlvaltab->zone_rctlval_limit));
3659             break;
3660         case PT_ACTION:
3661             (void) strcpy(rctlvaltab->zone_rctlval_action,
3662                 cx->cp_value,
3663                 sizeof(rctlvaltab->zone_rctlval_action));
3664             break;
3665         default:
3666             zone_perror(pt_to_str(prop_type),
3667                 Z_NO_PROPERTY_TYPE, B_TRUE);
3668             long_usage(CMD_ADD, B_TRUE);
3669             usage(B_FALSE, HELP_PROPS);
3670             zoncfg_free_rctl_value_list(rctlvaltab);
3671             return;
3672         }
3673     }
3674     rctlvaltab->zone_rctlval_next = NULL;
3675     err = zoncfg_remove_rctl_value(&in_progress_rctltab,
3676         rctlvaltab);
3677     if (err != Z_OK)
3678         zone_perror(pt_to_str(prop_type), err, B_TRUE);
3679     zoncfg_free_rctl_value_list(rctlvaltab);
3680     return;
3681 case RT_NET:
3682     if (prop_type != PT_DEFROUTER) {
3683         zone_perror(pt_to_str(prop_type), Z_NO_PROPERTY_TYPE,
3684             B_TRUE);

```



```

3685         long_usage(CMD_REMOVE, B_TRUE);
3686         usage(B_FALSE, HELP_PROPS);
3687         return;
3688     } else {
3689         bzero(&in_progress_nwifstab.zone_nwif_defrouter,
3690             sizeof (in_progress_nwifstab.zone_nwif_defrouter));
3691         return;
3692     }
3693     default:
3694         zone_perror(rt_to_str(res_type), Z_NO_RESOURCE_TYPE, B_TRUE);
3695         long_usage(CMD_REMOVE, B_TRUE);
3696         usage(B_FALSE, HELP_RESOURCES);
3697         return;
3698     }
3699 }

3701 void
3702 remove_func(cmd_t *cmd)
3703 {
3704     if (zone_is_read_only(CMD_REMOVE))
3705         return;

3707     assert(cmd != NULL);

3709     if (global_scope) {
3710         if (gz_invalid_resource(cmd->cmd_res_type)) {
3711             zerr(gettext("%s is not a valid resource for the "
3712                 "global zone."), rt_to_str(cmd->cmd_res_type));
3713             saw_error = B_TRUE;
3714             return;
3715         }
3716         remove_resource(cmd);
3717     } else {
3718         remove_property(cmd);
3719     }
3720 }

3722 static void
3723 clear_property(cmd_t *cmd)
3724 {
3725     int res_type, prop_type;

3727     res_type = resource_scope;
3728     prop_type = cmd->cmd_res_type;
3729     if (res_type == RT_UNKNOWN || prop_type == PT_UNKNOWN) {
3730         long_usage(CMD_CLEAR, B_TRUE);
3731         return;
3732     }

3734     if (initialize(B_TRUE) != Z_OK)
3735         return;

3737     switch (res_type) {
3738     case RT_FS:
3739         if (prop_type == PT_RAW) {
3740             in_progress_fstab.zone_fs_raw[0] = '\0';
3741             need_to_commit = B_TRUE;
3742             return;
3743         }
3744         break;
3745     case RT_DCPU:
3746         if (prop_type == PT_IMPORTANCE) {
3747             in_progress_psettab.zone_importance[0] = '\0';
3748             need_to_commit = B_TRUE;
3749             return;
3750         }

```

```

3751         break;
3752     case RT_MCAP:
3753         switch (prop_type) {
3754         case PT_PHYSICAL:
3755             in_progress_mcapstab.zone_physmem_cap[0] = '\0';
3756             need_to_commit = B_TRUE;
3757             return;
3758         case PT_SWAP:
3759             remove_aliased_rctl(PT_SWAP, ALIAS_MAXSWAP);
3760             return;
3761         case PT_LOCKED:
3762             remove_aliased_rctl(PT_LOCKED, ALIAS_MAXLOCKEDMEM);
3763             return;
3764         }
3765         break;
3766     case RT_SECFLAGS:
3767         switch (prop_type) {
3768         case PT_LOWER:
3769             in_progress_secflagstab.zone_secflags_lower[0] = '\0';
3770             need_to_commit = B_TRUE;
3771             return;
3772         case PT_DEFAULT:
3773             in_progress_secflagstab.zone_secflags_default[0] = '\0';
3774             need_to_commit = B_TRUE;
3775             return;
3776         case PT_UPPER:
3777             in_progress_secflagstab.zone_secflags_upper[0] = '\0';
3778             need_to_commit = B_TRUE;
3779             return;
3780         }
3781         break;
3782     #endif /* ! codereview */
3783     default:
3784         break;
3785     }

3787     zone_perror(pt_to_str(prop_type), Z_CLEAR_DISALLOW, B_TRUE);
3788 }

3790 static void
3791 clear_global(cmd_t *cmd)
3792 {
3793     int err, type;

3795     if ((type = cmd->cmd_res_type) == RT_UNKNOWN) {
3796         long_usage(CMD_CLEAR, B_TRUE);
3797         return;
3798     }

3800     if (initialize(B_TRUE) != Z_OK)
3801         return;

3803     switch (type) {
3804     case PT_ZONENAME:
3805         /* FALLTHRU */
3806     case PT_ZONEPATH:
3807         /* FALLTHRU */
3808     case PT_BRAND:
3809         zone_perror(pt_to_str(type), Z_CLEAR_DISALLOW, B_TRUE);
3810         return;
3811     case PT_AUTOBOOT:
3812         /* false is default; we'll treat as equivalent to clearing */
3813         if ((err = zoncfg_set_autoboot(handle, B_FALSE)) != Z_OK)
3814             z_cmd_rt_perror(CMD_CLEAR, RT_AUTOBOOT, err, B_TRUE);
3815         else
3816             need_to_commit = B_TRUE;

```

```

3817     return;
3818 case PT_POOL:
3819     if ((err = zoncfg_set_pool(handle, NULL)) != Z_OK)
3820         z_cmd_rt_perror(CMD_CLEAR, RT_POOL, err, B_TRUE);
3821     else
3822         need_to_commit = B_TRUE;
3823     return;
3824 case PT_LIMITPRIV:
3825     if ((err = zoncfg_set_limitpriv(handle, NULL)) != Z_OK)
3826         z_cmd_rt_perror(CMD_CLEAR, RT_LIMITPRIV, err, B_TRUE);
3827     else
3828         need_to_commit = B_TRUE;
3829     return;
3830 case PT_BOOTARGS:
3831     if ((err = zoncfg_set_bootargs(handle, NULL)) != Z_OK)
3832         z_cmd_rt_perror(CMD_CLEAR, RT_BOOTARGS, err, B_TRUE);
3833     else
3834         need_to_commit = B_TRUE;
3835     return;
3836 case PT_SCHED:
3837     if ((err = zoncfg_set_sched(handle, NULL)) != Z_OK)
3838         z_cmd_rt_perror(CMD_CLEAR, RT_SCHED, err, B_TRUE);
3839     else
3840         need_to_commit = B_TRUE;
3841     return;
3842 case PT_IPTYPE:
3843     /* shared is default; we'll treat as equivalent to clearing */
3844     if ((err = zoncfg_set_iptype(handle, ZS_SHARED)) != Z_OK)
3845         z_cmd_rt_perror(CMD_CLEAR, RT_IPTYPE, err, B_TRUE);
3846     else
3847         need_to_commit = B_TRUE;
3848     return;
3849 case PT_MAXLWPS:
3850     remove_aliased_rctl(PT_MAXLWPS, ALIAS_MAXLWPS);
3851     return;
3852 case PT_MAXPROCS:
3853     remove_aliased_rctl(PT_MAXPROCS, ALIAS_MAXPROCS);
3854     return;
3855 case PT_MAXSHMEM:
3856     remove_aliased_rctl(PT_MAXSHMEM, ALIAS_MAXSHMEM);
3857     return;
3858 case PT_MAXSHMIDS:
3859     remove_aliased_rctl(PT_MAXSHMIDS, ALIAS_MAXSHMIDS);
3860     return;
3861 case PT_MAXMSGIDS:
3862     remove_aliased_rctl(PT_MAXMSGIDS, ALIAS_MAXMSGIDS);
3863     return;
3864 case PT_MAXSEMIDS:
3865     remove_aliased_rctl(PT_MAXSEMIDS, ALIAS_MAXSEMIDS);
3866     return;
3867 case PT_SHARES:
3868     remove_aliased_rctl(PT_SHARES, ALIAS_SHARES);
3869     return;
3870 case PT_HOSTID:
3871     if ((err = zoncfg_set_hostid(handle, NULL)) != Z_OK)
3872         z_cmd_rt_perror(CMD_CLEAR, RT_HOSTID, err, B_TRUE);
3873     else
3874         need_to_commit = B_TRUE;
3875     return;
3876 case PT_FS_ALLOWED:
3877     if ((err = zoncfg_set_fs_allowed(handle, NULL)) != Z_OK)
3878         z_cmd_rt_perror(CMD_CLEAR, RT_FS_ALLOWED, err, B_TRUE);
3879     else
3880         need_to_commit = B_TRUE;
3881     return;
3882 default:

```

```

3883         zone_perror(pt_to_str(type), Z_NO_PROPERTY_TYPE, B_TRUE);
3884         long_usage(CMD_CLEAR, B_TRUE);
3885         usage(B_FALSE, HELP_PROPS);
3886         return;
3887     }
3888 }
3889 void
3890 clear_func(cmd_t *cmd)
3891 {
3892     if (zone_is_read_only(CMD_CLEAR))
3893         return;
3894
3895     assert(cmd != NULL);
3896
3897     if (global_scope) {
3898         if (gz_invalid_property(cmd->cmd_res_type)) {
3899             zerr(gettext("%s is not a valid property for the "
3900                "global zone."), pt_to_str(cmd->cmd_res_type));
3901             saw_error = B_TRUE;
3902             return;
3903         }
3904     }
3905
3906     clear_global(cmd);
3907 } else {
3908     clear_property(cmd);
3909 }
3910 }
3911
3912 void
3913 select_func(cmd_t *cmd)
3914 {
3915     int type, err, res;
3916     uint64_t limit;
3917     uint64_t tmp;
3918
3919     if (zone_is_read_only(CMD_SELECT))
3920         return;
3921
3922     assert(cmd != NULL);
3923
3924     if (global_scope) {
3925         global_scope = B_FALSE;
3926         resource_scope = cmd->cmd_res_type;
3927         end_op = CMD_SELECT;
3928     } else {
3929         scope_usage(CMD_SELECT);
3930         return;
3931     }
3932
3933     if ((type = cmd->cmd_res_type) == RT_UNKNOWN) {
3934         long_usage(CMD_SELECT, B_TRUE);
3935         return;
3936     }
3937
3938     if (initialize(B_TRUE) != Z_OK)
3939         return;
3940
3941     switch (type) {
3942     case RT_FS:
3943         if ((err = fill_in_fstab(cmd, &old_fstab, B_FALSE)) != Z_OK) {
3944             z_cmd_rt_perror(CMD_SELECT, RT_FS, err, B_TRUE);
3945             global_scope = B_TRUE;
3946         }
3947         bcopy(&old_fstab, &in_progress_fstab,
3948             sizeof (struct zone_fstab));

```

```

3949     return;
3950 case RT_NET:
3951     if ((err = fill_in_nwifstab(cmd, &old_nwifstab, B_FALSE))
3952         != Z_OK) {
3953         z_cmd_rt_perror(CMD_SELECT, RT_NET, err, B_TRUE);
3954         global_scope = B_TRUE;
3955     }
3956     bcopy(&old_nwifstab, &in_progress_nwifstab,
3957         sizeof (struct zone_nwifstab));
3958     return;
3959 case RT_DEVICE:
3960     if ((err = fill_in_devtab(cmd, &old_devtab, B_FALSE)) != Z_OK) {
3961         z_cmd_rt_perror(CMD_SELECT, RT_DEVICE, err, B_TRUE);
3962         global_scope = B_TRUE;
3963     }
3964     bcopy(&old_devtab, &in_progress_devtab,
3965         sizeof (struct zone_devtab));
3966     return;
3967 case RT_RCTL:
3968     if ((err = fill_in_rctltab(cmd, &old_rctltab, B_FALSE))
3969         != Z_OK) {
3970         z_cmd_rt_perror(CMD_SELECT, RT_RCTL, err, B_TRUE);
3971         global_scope = B_TRUE;
3972     }
3973     bcopy(&old_rctltab, &in_progress_rctltab,
3974         sizeof (struct zone_rctltab));
3975     return;
3976 case RT_ATTR:
3977     if ((err = fill_in_attrtab(cmd, &old_attrtab, B_FALSE))
3978         != Z_OK) {
3979         z_cmd_rt_perror(CMD_SELECT, RT_ATTR, err, B_TRUE);
3980         global_scope = B_TRUE;
3981     }
3982     bcopy(&old_attrtab, &in_progress_attrtab,
3983         sizeof (struct zone_attrtab));
3984     return;
3985 case RT_DATASET:
3986     if ((err = fill_in_dstab(cmd, &old_dstab, B_FALSE)) != Z_OK) {
3987         z_cmd_rt_perror(CMD_SELECT, RT_DATASET, err, B_TRUE);
3988         global_scope = B_TRUE;
3989     }
3990     bcopy(&old_dstab, &in_progress_dstab,
3991         sizeof (struct zone_dstab));
3992     return;
3993 case RT_DCPU:
3994     if ((err = zoncfg_lookup_pset(handle, &old_psettab)) != Z_OK) {
3995         z_cmd_rt_perror(CMD_SELECT, RT_DCPU, err, B_TRUE);
3996         global_scope = B_TRUE;
3997     }
3998     bcopy(&old_psettab, &in_progress_psettab,
3999         sizeof (struct zone_psettab));
4000     return;
4001 case RT_PCAP:
4002     if ((err = zoncfg_get_aliased_rctl(handle, ALIAS_CPUCAP, &tmp))
4003         != Z_OK) {
4004         z_cmd_rt_perror(CMD_SELECT, RT_PCAP, err, B_TRUE);
4005         global_scope = B_TRUE;
4006     }
4007     return;
4008 case RT_MCAP:
4009     /* if none of these exist, there is no resource to select */
4010     if ((res = zoncfg_lookup_mcap(handle, &old_mcaptab)) != Z_OK &&
4011         zoncfg_get_aliased_rctl(handle, ALIAS_MAXSWAP, &limit)
4012         != Z_OK &&
4013         zoncfg_get_aliased_rctl(handle, ALIAS_MAXLOCKEDMEM, &limit)
4014         != Z_OK) {

```

```

4015         z_cmd_rt_perror(CMD_SELECT, RT_MCAP, Z_NO_RESOURCE_TYPE,
4016             B_TRUE);
4017         global_scope = B_TRUE;
4018     }
4019     if (res == Z_OK)
4020         bcopy(&old_mcaptab, &in_progress_mcaptab,
4021             sizeof (struct zone_mcaptab));
4022     else
4023         bzero(&in_progress_mcaptab,
4024             sizeof (in_progress_mcaptab));
4025     return;
4026 case RT_ADMIN:
4027     if ((err = fill_in_admintab(cmd, &old_admintab, B_FALSE))
4028         != Z_OK) {
4029         z_cmd_rt_perror(CMD_SELECT, RT_ADMIN, err,
4030             B_TRUE);
4031         global_scope = B_TRUE;
4032     }
4033     bcopy(&old_admintab, &in_progress_admintab,
4034         sizeof (struct zone_admintab));
4035     return;
4036 case RT_SECFLAGS:
4037     if ((err = fill_in_secflagstab(cmd, &old_secflagstab, B_FALSE))
4038         != Z_OK) {
4039         z_cmd_rt_perror(CMD_SELECT, RT_SECFLAGS, err,
4040             B_TRUE);
4041         global_scope = B_TRUE;
4042     }
4043     bcopy(&old_secflagstab, &in_progress_secflagstab,
4044         sizeof (struct zone_secflagstab));
4045     return;
4046 #endif /* ! codereview */
4047     default:
4048         zone_perror(rt_to_str(type), Z_NO_RESOURCE_TYPE, B_TRUE);
4049         long_usage(CMD_SELECT, B_TRUE);
4050         usage(B_FALSE, HELP_RESOURCES);
4051         return;
4052     }
4053 }
4054
4055 /*
4056 * Network "addresses" can be one of the following forms:
4057 * <IPv4 address>
4058 * <IPv4 address>/<prefix length>
4059 * <IPv6 address>/<prefix length>
4060 * <host name>
4061 * <host name>/<prefix length>
4062 * In other words, the "/" followed by a prefix length is allowed but not
4063 * required for IPv4 addresses and host names, and required for IPv6 addresses.
4064 * If a prefix length is given, it must be in the allowable range: 0 to 32 for
4065 * IPv4 addresses and host names, 0 to 128 for IPv6 addresses.
4066 * Host names must start with an alpha-numeric character, and all subsequent
4067 * characters must be either alpha-numeric or "-".
4068 *
4069 * In some cases, e.g., the nexthop for the defrouter, the context indicates
4070 * that this is the IPV4_ABITS or IPV6_ABITS netmask, in which case we don't
4071 * require the /<prefix length> (and should ignore it if provided).
4072 */
4073
4074 static int
4075 validate_net_address_syntax(char *address, boolean_t ishost)
4076 {
4077     char *slashp, part1[MAXHOSTNAMELEN];
4078     struct in6_addr in6;
4079     struct in_addr in4;
4080     int prefixlen, i;

```

```

4082  /*
4083  * Copy the part before any '/' into part1 or copy the whole
4084  * thing if there is no '/'.
4085  */
4086  if ((slashp = strchr(address, '/')) != NULL) {
4087      *slashp = '\0';
4088      (void) strncpy(part1, address, sizeof (part1));
4089      *slashp = '/';
4090      prefixlen = atoi(++slashp);
4091  } else {
4092      (void) strncpy(part1, address, sizeof (part1));
4093  }
4094
4095  if (ishost && slashp != NULL) {
4096      zerr(gettext("Warning: prefix length in %s is not required and "
4097                "will be ignored. The default host-prefix length "
4098                "will be used"), address);
4099  }
4100
4101  if (inet_pton(AF_INET6, part1, &in6) == 1) {
4102      if (ishost) {
4103          prefixlen = IPV6_ABITS;
4104      } else if (slashp == NULL) {
4105          zerr(gettext("%s: IPv6 addresses "
4106                    "require /prefix-length suffix."), address);
4107          return (Z_ERR);
4108      }
4109      if (prefixlen < 0 || prefixlen > 128) {
4110          zerr(gettext("%s: IPv6 address "
4111                    "prefix lengths must be 0 - 128."), address);
4112          return (Z_ERR);
4113      }
4114      return (Z_OK);
4115  }
4116
4117  /* At this point, any /prefix must be for IPv4. */
4118  if (ishost)
4119      prefixlen = IPV4_ABITS;
4120  else if (slashp != NULL) {
4121      if (prefixlen < 0 || prefixlen > 32) {
4122          zerr(gettext("%s: IPv4 address "
4123                    "prefix lengths must be 0 - 32."), address);
4124          return (Z_ERR);
4125      }
4126  }
4127
4128  if (inet_pton(AF_INET, part1, &in4) == 1)
4129      return (Z_OK);
4130
4131  /* address may also be a host name */
4132  if (!isalnum(part1[0])) {
4133      zerr(gettext("%s: bogus host name or network address syntax"),
4134            part1);
4135      saw_error = B_TRUE;
4136      usage(B_FALSE, HELP_NETADDR);
4137      return (Z_ERR);
4138  }
4139  for (i = 1; part1[i]; i++)
4140      if (!isalnum(part1[i]) && part1[i] != '-' && part1[i] != '.') {
4141          zerr(gettext("%s: bogus host name or "
4142                    "network address syntax"), part1);
4143          saw_error = B_TRUE;
4144          usage(B_FALSE, HELP_NETADDR);
4145          return (Z_ERR);
4146      }

```

```

4147      }
4148      return (Z_OK);
4149  }
4150
4151  static int
4152  validate_net_physical_syntax(const char *ifname)
4153  {
4154      ifspec_t ifnameprop;
4155      zone_itype_t itype;
4156
4157      if (zoncfg_get_itype(handle, &itype) != Z_OK) {
4158          zerr(gettext("zone configuration has an invalid or nonexistent "
4159                    "ip-type property"));
4160          return (Z_ERR);
4161      }
4162      switch (itype) {
4163      case ZS_SHARED:
4164          if (ifparse_ifspec(ifname, &ifnameprop) == B_FALSE) {
4165              zerr(gettext("%s: invalid physical interface name"),
4166                    ifname);
4167              return (Z_ERR);
4168          }
4169          if (ifnameprop.ifsp_lunvalid) {
4170              zerr(gettext("%s: LUNs not allowed in physical "
4171                        "interface names"), ifname);
4172              return (Z_ERR);
4173          }
4174          break;
4175      case ZS_EXCLUSIVE:
4176          if (dladm_valid_linkname(ifname) == B_FALSE) {
4177              if (strchr(ifname, ':') != NULL)
4178                  zerr(gettext("%s: physical interface name "
4179                            "required; logical interface name not "
4180                            "allowed"), ifname);
4181              else
4182                  zerr(gettext("%s: invalid physical interface "
4183                            "name"), ifname);
4184              return (Z_ERR);
4185          }
4186          break;
4187      }
4188      return (Z_OK);
4189  }
4190
4191  static boolean_t
4192  valid_fs_type(const char *type)
4193  {
4194      /*
4195       * Is this a valid path component?
4196       */
4197      if (strlen(type) + 1 > MAXNAMELEN)
4198          return (B_FALSE);
4199      /*
4200       * Make sure a bad value for "type" doesn't make
4201       * /usr/lib/fs/<type>/mount turn into something else.
4202       */
4203      if (strchr(type, '/') != NULL || type[0] == '\0' ||
4204          strcmp(type, ".") == 0 || strcmp(type, "..") == 0)
4205          return (B_FALSE);
4206      /*
4207       * More detailed verification happens later by zoneadm(1m).
4208       */
4209      return (B_TRUE);
4210  }
4211
4212  static boolean_t

```

```

4213 allow_exclusive()
4214 {
4215     brand_handle_t bh;
4216     char brand[MAXNAMELEN];
4217     boolean_t ret;
4218
4219     if (zonecfg_get_brand(handle, brand, sizeof (brand)) != Z_OK) {
4220         zerr("%s: %s\n", zone, gettext("could not get zone brand"));
4221         return (B_FALSE);
4222     }
4223     if ((bh = brand_open(brand)) == NULL) {
4224         zerr("%s: %s\n", zone, gettext("unknown brand."));
4225         return (B_FALSE);
4226     }
4227     ret = brand_allow_exclusive_ip(bh);
4228     brand_close(bh);
4229     if (!ret)
4230         zerr(gettext("%s cannot be '%s' when %s is '%s'."),
4231             pt_to_str(PT_IPTYPE), "exclusive",
4232             pt_to_str(PT_BRAND), brand);
4233     return (ret);
4234 }
4235
4236 static void
4237 set_aliased_rctl(char *alias, int prop_type, char *s)
4238 {
4239     uint64_t limit;
4240     int err;
4241     char tmp[128];
4242
4243     if (global_zone && strcmp(alias, ALIAS_SHARES) != 0)
4244         zerr(gettext("WARNING: Setting a global zone resource "
4245             "control too low could deny\nservice "
4246             "to even the root user; "
4247             "this could render the system impossible\n"
4248             "to administer. Please use caution."));
4249
4250     /* convert memory based properties */
4251     if (prop_type == PT_MAXSHMMEM) {
4252         if (!zonecfg_valid_memlimit(s, &limit)) {
4253             zerr(gettext("A non-negative number with a required "
4254                 "scale suffix (K, M, G or T) was expected\nhere."));
4255             saw_error = B_TRUE;
4256             return;
4257         }
4258
4259         (void) snprintf(tmp, sizeof (tmp), "%llu", limit);
4260         s = tmp;
4261     }
4262
4263     if (!zonecfg_aliased_rctl_ok(handle, alias)) {
4264         zone_perror(pt_to_str(prop_type), Z_ALIAS_DISALLOW, B_FALSE);
4265         saw_error = B_TRUE;
4266     } else if (!zonecfg_valid_alias_limit(alias, s, &limit)) {
4267         zerr(gettext("%s property is out of range."),
4268             pt_to_str(prop_type));
4269         saw_error = B_TRUE;
4270     } else if ((err = zonecfg_set_aliased_rctl(handle, alias, limit))
4271         != Z_OK) {
4272         zone_perror(zone, err, B_TRUE);
4273         saw_error = B_TRUE;
4274     } else {
4275         need_to_commit = B_TRUE;
4276     }
4277 }

```

```

4279 static void
4280 set_in_progress_nwiftab_address(char *prop_id, int prop_type)
4281 {
4282     if (prop_type == PT_ADDRESS) {
4283         (void) strcpy(in_progress_nwiftab.zone_nwif_address, prop_id,
4284             sizeof (in_progress_nwiftab.zone_nwif_address));
4285     } else {
4286         assert(prop_type == PT_ALLOWED_ADDRESS);
4287         (void) strcpy(in_progress_nwiftab.zone_nwif_allowed_address,
4288             prop_id,
4289             sizeof (in_progress_nwiftab.zone_nwif_allowed_address));
4290     }
4291 }
4292
4293 void
4294 set_func(cmd_t *cmd)
4295 {
4296     char *prop_id;
4297     int arg, err, res_type, prop_type;
4298     property_value_ptr_t pp;
4299     boolean_t autoboot;
4300     zone_ip_type_t iptype;
4301     boolean_t force_set = B_FALSE;
4302     size_t physmem_size = sizeof (in_progress_mcaptab.zone_physmem_cap);
4303     uint64_t mem_cap, mem_limit;
4304     float cap;
4305     char *unitp;
4306     struct zone_psettab tmp_psettab;
4307     boolean_t arg_err = B_FALSE;
4308
4309     if (zone_is_read_only(CMD_SET))
4310         return;
4311
4312     assert(cmd != NULL);
4313
4314     optind = opterr = 0;
4315     while ((arg = getopt(cmd->cmd_argc, cmd->cmd_argv, "F")) != EOF) {
4316         switch (arg) {
4317             case 'F':
4318                 force_set = B_TRUE;
4319                 break;
4320             default:
4321                 if (optopt == '?')
4322                     longer_usage(CMD_SET);
4323                 else
4324                     short_usage(CMD_SET);
4325                 arg_err = B_TRUE;
4326                 break;
4327         }
4328     }
4329     if (arg_err)
4330         return;
4331
4332     prop_type = cmd->cmd_prop_name[0];
4333     if (global_scope) {
4334         if (gz_invalid_property(prop_type)) {
4335             zerr(gettext("%s is not a valid property for the "
4336                 "global zone."), pt_to_str(prop_type));
4337             saw_error = B_TRUE;
4338             return;
4339         }
4340
4341         if (prop_type == PT_ZONEENAME) {
4342             res_type = RT_ZONEENAME;
4343         } else if (prop_type == PT_ZONEPATH) {
4344             res_type = RT_ZONEPATH;

```

```

4345     } else if (prop_type == PT_AUTOBOOT) {
4346         res_type = RT_AUTOBOOT;
4347     } else if (prop_type == PT_BRAND) {
4348         res_type = RT_BRAND;
4349     } else if (prop_type == PT_POOL) {
4350         res_type = RT_POOL;
4351     } else if (prop_type == PT_LIMITPRIV) {
4352         res_type = RT_LIMITPRIV;
4353     } else if (prop_type == PT_BOOTARGS) {
4354         res_type = RT_BOOTARGS;
4355     } else if (prop_type == PT_SCHED) {
4356         res_type = RT_SCHED;
4357     } else if (prop_type == PT_IPTYPE) {
4358         res_type = RT_IPTYPE;
4359     } else if (prop_type == PT_MAXLWPS) {
4360         res_type = RT_MAXLWPS;
4361     } else if (prop_type == PT_MAXPROCS) {
4362         res_type = RT_MAXPROCS;
4363     } else if (prop_type == PT_MAXSHMMEM) {
4364         res_type = RT_MAXSHMMEM;
4365     } else if (prop_type == PT_MAXSHMIDS) {
4366         res_type = RT_MAXSHMIDS;
4367     } else if (prop_type == PT_MAXMSGIDS) {
4368         res_type = RT_MAXMSGIDS;
4369     } else if (prop_type == PT_MAXSEMIDS) {
4370         res_type = RT_MAXSEMIDS;
4371     } else if (prop_type == PT_SHARES) {
4372         res_type = RT_SHARES;
4373     } else if (prop_type == PT_HOSTID) {
4374         res_type = RT_HOSTID;
4375     } else if (prop_type == PT_FS_ALLOWED) {
4376         res_type = RT_FS_ALLOWED;
4377     } else {
4378         zerr(gettext("Cannot set a resource-specific property "
4379             "from the global scope."));
4380         saw_error = B_TRUE;
4381         return;
4382     }
4383 } else {
4384     res_type = resource_scope;
4385 }
4387 if (force_set) {
4388     if (res_type != RT_ZONEPATH) {
4389         zerr(gettext("Only zonepath setting can be forced."));
4390         saw_error = B_TRUE;
4391         return;
4392     }
4393     if (!zoncfg_in_alt_root()) {
4394         zerr(gettext("Zonepath is changeable only in an "
4395             "alternate root."));
4396         saw_error = B_TRUE;
4397         return;
4398     }
4399 }
4401 pp = cmd->cmd_property_ptr[0];
4402 /*
4403  * A nasty expression but not that complicated:
4404  * 1. fs options are simple or list (tested below)
4405  * 2. rctl value's are complex or list (tested below)
4406  * Anything else should be simple.
4407  */
4408 if (!(res_type == RT_FS && prop_type == PT_OPTIONS) &&
4409     !(res_type == RT_RCTL && prop_type == PT_VALUE) &&
4410     (pp->pv_type != PROP_VAL_SIMPLE ||

```

```

4411     (prop_id = pp->pv_simple) == NULL)) {
4412         zerr(gettext("A %s value was expected here."),
4413             pvt_to_str(PROP_VAL_SIMPLE));
4414         saw_error = B_TRUE;
4415         return;
4416     }
4417     if (prop_type == PT_UNKNOWN) {
4418         long_usage(CMD_SET, B_TRUE);
4419         return;
4420     }
4422     /*
4423     * Special case: the user can change the zone name prior to 'create';
4424     * if the zone already exists, we fall through letting initialize()
4425     * and the rest of the logic run.
4426     */
4427     if (res_type == RT_ZONENAME && got_handle == B_FALSE &&
4428         !state_atleast(ZONE_STATE_CONFIGURED)) {
4429         if ((err = zoncfg_validate_zonename(prop_id)) != Z_OK) {
4430             zone_perror(prop_id, err, B_TRUE);
4431             usage(B_FALSE, HELP_SYNTAX);
4432             return;
4433         }
4434         (void) strcpy(zone, prop_id, sizeof (zone));
4435         return;
4436     }
4438     if (initialize(B_TRUE) != Z_OK)
4439         return;
4441     switch (res_type) {
4442     case RT_ZONENAME:
4443         if ((err = zoncfg_set_name(handle, prop_id)) != Z_OK) {
4444             /*
4445              * Use prop_id instead of 'zone' here, since we're
4446              * reporting a problem about the *new* zonename.
4447              */
4448             zone_perror(prop_id, err, B_TRUE);
4449             usage(B_FALSE, HELP_SYNTAX);
4450         } else {
4451             need_to_commit = B_TRUE;
4452             (void) strcpy(zone, prop_id, sizeof (zone));
4453         }
4454         return;
4455     case RT_ZONEPATH:
4456         if (!force_set && state_atleast(ZONE_STATE_INSTALLED)) {
4457             zerr(gettext("Zone %s already installed; %s %s not "
4458                 "allowed."), zone, cmd_to_str(CMD_SET),
4459                 rt_to_str(RT_ZONEPATH));
4460             return;
4461         }
4462         if (validate_zonepath_syntax(prop_id) != Z_OK) {
4463             saw_error = B_TRUE;
4464             return;
4465         }
4466         if ((err = zoncfg_set_zonepath(handle, prop_id)) != Z_OK)
4467             zone_perror(zone, err, B_TRUE);
4468         else
4469             need_to_commit = B_TRUE;
4470         return;
4471     case RT_BRAND:
4472         if (state_atleast(ZONE_STATE_INSTALLED)) {
4473             zerr(gettext("Zone %s already installed; %s %s not "
4474                 "allowed."), zone, cmd_to_str(CMD_SET),
4475                 rt_to_str(RT_BRAND));
4476             return;

```

```

4477     }
4478     if ((err = zonecfg_set_brand(handle, prop_id)) != Z_OK)
4479         zone_perror(zone, err, B_TRUE);
4480     else
4481         need_to_commit = B_TRUE;
4482     return;
4483 case RT_AUTOBOOT:
4484     if (strcmp(prop_id, "true") == 0) {
4485         autoboot = B_TRUE;
4486     } else if (strcmp(prop_id, "false") == 0) {
4487         autoboot = B_FALSE;
4488     } else {
4489         zerr(gettext("%s value must be '%s' or '%s'."),
4490             pt_to_str(PT_AUTOBOOT), "true", "false");
4491         saw_error = B_TRUE;
4492         return;
4493     }
4494     if (((err = zonecfg_set_autoboot(handle, autoboot)) != Z_OK)
4495         zone_perror(zone, err, B_TRUE);
4496     else
4497         need_to_commit = B_TRUE;
4498     return;
4499 case RT_POOL:
4500     /* don't allow use of the reserved temporary pool names */
4501     if (strcmp("SUNW", prop_id, 4) == 0) {
4502         zerr(gettext("pool names starting with SUNW are "
4503             "reserved."));
4504         saw_error = B_TRUE;
4505         return;
4506     }
4507
4508     /* can't set pool if dedicated-cpu exists */
4509     if (zonecfg_lookup_pset(handle, &tmp_psettab) == Z_OK) {
4510         zerr(gettext("The %s resource already exists. "
4511             "A persistent pool is incompatible\nwith the %s "
4512             "resource."), rt_to_str(RT_DCPU),
4513             rt_to_str(RT_DCPU));
4514         saw_error = B_TRUE;
4515         return;
4516     }
4517
4518     if ((err = zonecfg_set_pool(handle, prop_id)) != Z_OK)
4519         zone_perror(zone, err, B_TRUE);
4520     else
4521         need_to_commit = B_TRUE;
4522     return;
4523 case RT_LIMITPRIV:
4524     if ((err = zonecfg_set_limitpriv(handle, prop_id)) != Z_OK)
4525         zone_perror(zone, err, B_TRUE);
4526     else
4527         need_to_commit = B_TRUE;
4528     return;
4529 case RT_BOOTARGS:
4530     if ((err = zonecfg_set_bootargs(handle, prop_id)) != Z_OK)
4531         zone_perror(zone, err, B_TRUE);
4532     else
4533         need_to_commit = B_TRUE;
4534     return;
4535 case RT_SCHED:
4536     if ((err = zonecfg_set_sched(handle, prop_id)) != Z_OK)
4537         zone_perror(zone, err, B_TRUE);
4538     else
4539         need_to_commit = B_TRUE;
4540     return;
4541 case RT_IPTYPE:
4542     if (strcmp(prop_id, "shared") == 0) {

```

```

4543         iptype = ZS_SHARED;
4544     } else if (strcmp(prop_id, "exclusive") == 0) {
4545         iptype = ZS_EXCLUSIVE;
4546     } else {
4547         zerr(gettext("%s value must be '%s' or '%s'."),
4548             pt_to_str(PT_IPTYPE), "shared", "exclusive");
4549         saw_error = B_TRUE;
4550         return;
4551     }
4552     if (iptype == ZS_EXCLUSIVE && !allow_exclusive()) {
4553         saw_error = B_TRUE;
4554         return;
4555     }
4556     if ((err = zonecfg_set_iptype(handle, iptype)) != Z_OK)
4557         zone_perror(zone, err, B_TRUE);
4558     else
4559         need_to_commit = B_TRUE;
4560     return;
4561 case RT_MAXLWPS:
4562     set_aliased_rctl(ALIAS_MAXLWPS, prop_type, prop_id);
4563     return;
4564 case RT_MAXPROCS:
4565     set_aliased_rctl(ALIAS_MAXPROCS, prop_type, prop_id);
4566     return;
4567 case RT_MAXSHMMEM:
4568     set_aliased_rctl(ALIAS_MAXSHMMEM, prop_type, prop_id);
4569     return;
4570 case RT_MAXSHMIDS:
4571     set_aliased_rctl(ALIAS_MAXSHMIDS, prop_type, prop_id);
4572     return;
4573 case RT_MAXMSGIDS:
4574     set_aliased_rctl(ALIAS_MAXMSGIDS, prop_type, prop_id);
4575     return;
4576 case RT_MAXSEMIDS:
4577     set_aliased_rctl(ALIAS_MAXSEMIDS, prop_type, prop_id);
4578     return;
4579 case RT_SHARES:
4580     set_aliased_rctl(ALIAS_SHARES, prop_type, prop_id);
4581     return;
4582 case RT_HOSTID:
4583     if ((err = zonecfg_set_hostid(handle, prop_id)) != Z_OK) {
4584         if (err == Z_TOO_BIG) {
4585             zerr(gettext("hostid string is too large: %s"),
4586                 prop_id);
4587             saw_error = B_TRUE;
4588         } else {
4589             zone_perror(pt_to_str(prop_type), err, B_TRUE);
4590         }
4591         return;
4592     }
4593     need_to_commit = B_TRUE;
4594     return;
4595 case RT_FS_ALLOWED:
4596     if ((err = zonecfg_set_fs_allowed(handle, prop_id)) != Z_OK)
4597         zone_perror(zone, err, B_TRUE);
4598     else
4599         need_to_commit = B_TRUE;
4600     return;
4601 case RT_FS:
4602     switch (prop_type) {
4603     case PT_DIR:
4604         (void) strlcpy(in_progress_fstab.zone_fs_dir, prop_id,
4605             sizeof (in_progress_fstab.zone_fs_dir));
4606         return;
4607     case PT_SPECIAL:
4608         (void) strlcpy(in_progress_fstab.zone_fs_special,

```

```

4609         prop_id,
4610         sizeof (in_progress_fstab.zone_fs_special));
4611     return;
4612 case PT_RAW:
4613     (void) strncpy(in_progress_fstab.zone_fs_raw,
4614                  prop_id, sizeof (in_progress_fstab.zone_fs_raw));
4615     return;
4616 case PT_TYPE:
4617     if (!valid_fs_type(prop_id)) {
4618         zerr(gettext("\'%s\' is not a valid %s."),
4619             prop_id, pt_to_str(PT_TYPE));
4620         saw_error = B_TRUE;
4621         return;
4622     }
4623     (void) strncpy(in_progress_fstab.zone_fs_type, prop_id,
4624                  sizeof (in_progress_fstab.zone_fs_type));
4625     return;
4626 case PT_OPTIONS:
4627     if (pp->pv_type != PROP_VAL_SIMPLE &&
4628         pp->pv_type != PROP_VAL_LIST) {
4629         zerr(gettext("A %s or %s value was expected "
4630                    "here."), pvt_to_str(PROP_VAL_SIMPLE),
4631             pvt_to_str(PROP_VAL_LIST));
4632         saw_error = B_TRUE;
4633         return;
4634     }
4635     zoncfg_free_fs_option_list(
4636         in_progress_fstab.zone_fs_options);
4637     in_progress_fstab.zone_fs_options = NULL;
4638     if (!(pp->pv_type == PROP_VAL_LIST &&
4639         pp->pv_list == NULL))
4640         add_property(cmd);
4641     return;
4642 default:
4643     break;
4644 }
4645 zone_perror(pt_to_str(prop_type), Z_NO_PROPERTY_TYPE, B_TRUE);
4646 long_usage(CMD_SET, B_TRUE);
4647 usage(B_FALSE, HELP_PROPS);
4648 return;
4649 case RT_NET:
4650     switch (prop_type) {
4651     case PT_ADDRESS:
4652     case PT_ALLOWED_ADDRESS:
4653         if (validate_net_address_syntax(prop_id, B_FALSE)
4654             != Z_OK) {
4655             saw_error = B_TRUE;
4656             return;
4657         }
4658         set_in_progress_nwifstab_address(prop_id, prop_type);
4659         break;
4660     case PT_PHYSICAL:
4661         if (validate_net_physical_syntax(prop_id) != Z_OK) {
4662             saw_error = B_TRUE;
4663             return;
4664         }
4665         (void) strncpy(in_progress_nwifstab.zone_nwif_physical,
4666                      prop_id,
4667                      sizeof (in_progress_nwifstab.zone_nwif_physical));
4668         break;
4669     case PT_DEFROUTER:
4670         if (validate_net_address_syntax(prop_id, B_TRUE)
4671             != Z_OK) {
4672             saw_error = B_TRUE;
4673             return;
4674         }

```

```

4675         (void) strncpy(in_progress_nwifstab.zone_nwif_defrouter,
4676                      prop_id,
4677                      sizeof (in_progress_nwifstab.zone_nwif_defrouter));
4678         break;
4679     default:
4680         zone_perror(pt_to_str(prop_type), Z_NO_PROPERTY_TYPE,
4681                    B_TRUE);
4682         long_usage(CMD_SET, B_TRUE);
4683         usage(B_FALSE, HELP_PROPS);
4684         return;
4685     }
4686     return;
4687 case RT_DEVICE:
4688     switch (prop_type) {
4689     case PT_MATCH:
4690         (void) strncpy(in_progress_devtab.zone_dev_match,
4691                      prop_id,
4692                      sizeof (in_progress_devtab.zone_dev_match));
4693         break;
4694     default:
4695         zone_perror(pt_to_str(prop_type), Z_NO_PROPERTY_TYPE,
4696                    B_TRUE);
4697         long_usage(CMD_SET, B_TRUE);
4698         usage(B_FALSE, HELP_PROPS);
4699         return;
4700     }
4701     return;
4702 case RT_RCTL:
4703     switch (prop_type) {
4704     case PT_NAME:
4705         if (!zoncfg_valid_rctlname(prop_id)) {
4706             zerr(gettext("\'%s\' is not a valid zone %s "
4707                        "name."), prop_id, rt_to_str(RT_RCTL));
4708             return;
4709         }
4710         (void) strncpy(in_progress_rctltab.zone_rctl_name,
4711                      prop_id,
4712                      sizeof (in_progress_rctltab.zone_rctl_name));
4713         break;
4714     case PT_VALUE:
4715         if (pp->pv_type != PROP_VAL_COMPLEX &&
4716             pp->pv_type != PROP_VAL_LIST) {
4717             zerr(gettext("A %s or %s value was expected "
4718                        "here."), pvt_to_str(PROP_VAL_COMPLEX),
4719                 pvt_to_str(PROP_VAL_LIST));
4720             saw_error = B_TRUE;
4721             return;
4722         }
4723         zoncfg_free_rctl_value_list(
4724             in_progress_rctltab.zone_rctl_valptr);
4725         in_progress_rctltab.zone_rctl_valptr = NULL;
4726         if (!(pp->pv_type == PROP_VAL_LIST &&
4727             pp->pv_list == NULL))
4728             add_property(cmd);
4729         break;
4730     default:
4731         zone_perror(pt_to_str(prop_type), Z_NO_PROPERTY_TYPE,
4732                    B_TRUE);
4733         long_usage(CMD_SET, B_TRUE);
4734         usage(B_FALSE, HELP_PROPS);
4735         return;
4736     }
4737     return;
4738 case RT_ATTR:
4739     switch (prop_type) {
4740     case PT_NAME:

```



```

4741         (void) strcpy(in_progress_attrtab.zone_attr_name,
4742             prop_id,
4743             sizeof (in_progress_attrtab.zone_attr_name));
4744         break;
4745     case PT_TYPE:
4746         (void) strcpy(in_progress_attrtab.zone_attr_type,
4747             prop_id,
4748             sizeof (in_progress_attrtab.zone_attr_type));
4749         break;
4750     case PT_VALUE:
4751         (void) strcpy(in_progress_attrtab.zone_attr_value,
4752             prop_id,
4753             sizeof (in_progress_attrtab.zone_attr_value));
4754         break;
4755     default:
4756         zone_perror(pt_to_str(prop_type), Z_NO_PROPERTY_TYPE,
4757             B_TRUE);
4758         long_usage(CMD_SET, B_TRUE);
4759         usage(B_FALSE, HELP_PROPS);
4760         return;
4761     }
4762     return;
4763 case RT_DATASET:
4764     switch (prop_type) {
4765     case PT_NAME:
4766         (void) strcpy(in_progress_dstab.zone_dataset_name,
4767             prop_id,
4768             sizeof (in_progress_dstab.zone_dataset_name));
4769         return;
4770     default:
4771         break;
4772     }
4773     zone_perror(pt_to_str(prop_type), Z_NO_PROPERTY_TYPE, B_TRUE);
4774     long_usage(CMD_SET, B_TRUE);
4775     usage(B_FALSE, HELP_PROPS);
4776     return;
4777 case RT_DCPU:
4778     switch (prop_type) {
4779     char *lowp, *highp;
4780
4781     case PT_NCPUS:
4782         lowp = prop_id;
4783         if ((highp = strchr(prop_id, '-')) != NULL)
4784             *highp++ = '\0';
4785         else
4786             highp = lowp;
4787
4788         /* Make sure the input makes sense. */
4789         if (!zonecfg_valid_ncpus(lowp, highp)) {
4790             zerr(gettext("%s property is out of range."),
4791                 pt_to_str(PT_NCPUS));
4792             saw_error = B_TRUE;
4793             return;
4794         }
4795
4796         (void) strcpy(
4797             in_progress_psettab.zone_ncpu_min, lowp,
4798             sizeof (in_progress_psettab.zone_ncpu_min));
4799         (void) strcpy(
4800             in_progress_psettab.zone_ncpu_max, highp,
4801             sizeof (in_progress_psettab.zone_ncpu_max));
4802         return;
4803     case PT_IMPORTANCE:
4804         /* Make sure the value makes sense. */
4805         if (!zonecfg_valid_importance(prop_id)) {
4806             zerr(gettext("%s property is out of range."),

```

```

4807             pt_to_str(PT_IMPORTANCE));
4808             saw_error = B_TRUE;
4809             return;
4810         }
4811
4812         (void) strcpy(in_progress_psettab.zone_importance,
4813             prop_id,
4814             sizeof (in_progress_psettab.zone_importance));
4815         return;
4816     default:
4817         break;
4818     }
4819     zone_perror(pt_to_str(prop_type), Z_NO_PROPERTY_TYPE, B_TRUE);
4820     long_usage(CMD_SET, B_TRUE);
4821     usage(B_FALSE, HELP_PROPS);
4822     return;
4823 case RT_PCAP:
4824     if (prop_type != PT_NCPUS) {
4825         zone_perror(pt_to_str(prop_type), Z_NO_PROPERTY_TYPE,
4826             B_TRUE);
4827         long_usage(CMD_SET, B_TRUE);
4828         usage(B_FALSE, HELP_PROPS);
4829         return;
4830     }
4831
4832     /*
4833     * We already checked that an rctl alias is allowed in
4834     * the add_resource() function.
4835     */
4836
4837     if ((cap = strtouprop(prop_id, &unitp)) <= 0 || *unitp != '\0' ||
4838         (int)(cap * 100) < 1) {
4839         zerr(gettext("%s property is out of range."),
4840             pt_to_str(PT_NCPUS));
4841         saw_error = B_TRUE;
4842         return;
4843     }
4844
4845     if ((err = zonecfg_set_aliased_rctl(handle, ALIAS_CPUCAP,
4846         (int)(cap * 100))) != Z_OK)
4847         zone_perror(zone, err, B_TRUE);
4848     else
4849         need_to_commit = B_TRUE;
4850     return;
4851 case RT_MCAP:
4852     switch (prop_type) {
4853     case PT_PHYSICAL:
4854         if (!zonecfg_valid_memlimit(prop_id, &mem_cap)) {
4855             zerr(gettext("A positive number with a "
4856                 "required scale suffix (K, M, G or T) was "
4857                 "expected here.));
4858             saw_error = B_TRUE;
4859         } else if (mem_cap < ONE_MB) {
4860             zerr(gettext("%s value is too small. It must "
4861                 "be at least 1M."), pt_to_str(PT_PHYSICAL));
4862             saw_error = B_TRUE;
4863         } else {
4864             snprintf(in_progress_mcaptab.zone_physmem_cap,
4865                 physmem_size, "%llu", mem_cap);
4866         }
4867         break;
4868     case PT_SWAP:
4869         /*
4870         * We have to check if an rctl is allowed here since
4871         * there might already be a rctl defined that blocks
4872         * the alias.

```

```

4873     */
4874     if (!zonecfg_aliased_rctl_ok(handle, ALIAS_MAXSWAP)) {
4875         zone_perror(pt_to_str(PT_MAXSWAP),
4876             Z_ALIAS_DISALLOW, B_FALSE);
4877         saw_error = B_TRUE;
4878         return;
4879     }
4881     if (global_zone)
4882         mem_limit = ONE_MB * 100;
4883     else
4884         mem_limit = ONE_MB * 50;
4886     if (!zonecfg_valid_memlimit(prop_id, &mem_cap)) {
4887         zerr(gettext("A positive number with a "
4888             "required scale suffix (K, M, G or T) was "
4889             "expected here.));
4890         saw_error = B_TRUE;
4891     } else if (mem_cap < mem_limit) {
4892         char buf[128];
4894         (void) snprintf(buf, sizeof (buf), "%llu",
4895             mem_limit);
4896         bytes_to_units(buf, buf, sizeof (buf));
4897         zerr(gettext("%s value is too small. It must "
4898             "be at least %s."), pt_to_str(PT_SWAP),
4899             buf);
4900         saw_error = B_TRUE;
4901     } else {
4902         if ((err = zonecfg_set_aliased_rctl(handle,
4903             ALIAS_MAXSWAP, mem_cap)) != Z_OK)
4904             zone_perror(zone, err, B_TRUE);
4905         else
4906             need_to_commit = B_TRUE;
4907     }
4908     break;
4909 case PT_LOCKED:
4910     /*
4911     * We have to check if an rctl is allowed here since
4912     * there might already be a rctl defined that blocks
4913     * the alias.
4914     */
4915     if (!zonecfg_aliased_rctl_ok(handle,
4916         ALIAS_MAXLOCKEDMEM)) {
4917         zone_perror(pt_to_str(PT_LOCKED),
4918             Z_ALIAS_DISALLOW, B_FALSE);
4919         saw_error = B_TRUE;
4920         return;
4921     }
4923     if (!zonecfg_valid_memlimit(prop_id, &mem_cap)) {
4924         zerr(gettext("A non-negative number with a "
4925             "required scale suffix (K, M, G or T) was "
4926             "expected\nhere.));
4927         saw_error = B_TRUE;
4928     } else {
4929         if ((err = zonecfg_set_aliased_rctl(handle,
4930             ALIAS_MAXLOCKEDMEM, mem_cap)) != Z_OK)
4931             zone_perror(zone, err, B_TRUE);
4932         else
4933             need_to_commit = B_TRUE;
4934     }
4935     break;
4936 default:
4937     zone_perror(pt_to_str(prop_type), Z_NO_PROPERTY_TYPE,
4938         B_TRUE);

```

```

4939         long_usage(CMD_SET, B_TRUE);
4940         usage(B_FALSE, HELP_PROPS);
4941         return;
4942     }
4943     return;
4944 case RT_ADMIN:
4945     switch (prop_type) {
4946     case PT_USER:
4947         (void) strncpy(in_progress_admintab.zone_admin_user,
4948             prop_id,
4949             sizeof (in_progress_admintab.zone_admin_user));
4950         return;
4951     case PT_AUTHS:
4952         (void) strncpy(in_progress_admintab.zone_admin_auths,
4953             prop_id,
4954             sizeof (in_progress_admintab.zone_admin_auths));
4955         return;
4956     default:
4957         zone_perror(pt_to_str(prop_type), Z_NO_PROPERTY_TYPE,
4958             B_TRUE);
4959         long_usage(CMD_SET, B_TRUE);
4960         usage(B_FALSE, HELP_PROPS);
4961         return;
4962     }
4963 case RT_SECFLAGS: {
4964     char *propstr;
4966     switch (prop_type) {
4967     case PT_DEFAULT:
4968         propstr = in_progress_secflagstab.zone_secflags_default;
4969         break;
4970     case PT_UPPER:
4971         propstr = in_progress_secflagstab.zone_secflags_upper;
4972         break;
4973     case PT_LOWER:
4974         propstr = in_progress_secflagstab.zone_secflags_lower;
4975         break;
4976     default:
4977         zone_perror(pt_to_str(prop_type), Z_NO_PROPERTY_TYPE,
4978             B_TRUE);
4979         long_usage(CMD_SET, B_TRUE);
4980         usage(B_FALSE, HELP_PROPS);
4981         return;
4982     }
4983     (void) strncpy(propstr, prop_id, ZONECFG_SECFLAGS_MAX);
4984     return;
4985 }
4986 #endif /* ! codereview */
4987 default:
4988     zone_perror(rt_to_str(res_type), Z_NO_RESOURCE_TYPE, B_TRUE);
4989     long_usage(CMD_SET, B_TRUE);
4990     usage(B_FALSE, HELP_RESOURCES);
4991     return;
4992 }
4993 }
4995 static void
4996 output_prop(FILE *fp, int pnum, char *pval, boolean_t print_notspec)
4997 {
4998     char *qstr;
5000     if (*pval != '\0') {
5001         qstr = quoteit(pval);
5002         if (pnum == PT_SWAP || pnum == PT_LOCKED)
5003             (void) fprintf(fp, "\t[%s: %s]\n", pt_to_str(pnum),
5004                 qstr);

```

```

5005         else
5006             (void) fprintf(fp, "\t%s: %s\n", pt_to_str(pnum), qstr);
5007             free(qstr);
5008     } else if (print_notspec)
5009         (void) fprintf(fp, gettext("\t%s not specified\n"),
5010             pt_to_str(pnum));
5011 }

5013 static void
5014 info_zonename(zone_dochandle_t handle, FILE *fp)
5015 {
5016     char zonename[ZONENAME_MAX];

5018     if (zoncfg_get_name(handle, zonename, sizeof (zonename)) == Z_OK)
5019         (void) fprintf(fp, "%s: %s\n", pt_to_str(PT_ZONENAME),
5020             zonename);
5021     else
5022         (void) fprintf(fp, gettext("%s not specified\n"),
5023             pt_to_str(PT_ZONENAME));
5024 }

5026 static void
5027 info_zonepath(zone_dochandle_t handle, FILE *fp)
5028 {
5029     char zonepath[MAXPATHLEN];

5031     if (zoncfg_get_zonepath(handle, zonepath, sizeof (zonepath)) == Z_OK)
5032         (void) fprintf(fp, "%s: %s\n", pt_to_str(PT_ZONEPATH),
5033             zonepath);
5034     else {
5035         (void) fprintf(fp, gettext("%s not specified\n"),
5036             pt_to_str(PT_ZONEPATH));
5037     }
5038 }

5040 static void
5041 info_brand(zone_dochandle_t handle, FILE *fp)
5042 {
5043     char brand[MAXNAMELEN];

5045     if (zoncfg_get_brand(handle, brand, sizeof (brand)) == Z_OK)
5046         (void) fprintf(fp, "%s: %s\n", pt_to_str(PT_BRAND),
5047             brand);
5048     else
5049         (void) fprintf(fp, "%s %s\n", pt_to_str(PT_BRAND),
5050             gettext("not specified"));
5051 }

5053 static void
5054 info_autoboot(zone_dochandle_t handle, FILE *fp)
5055 {
5056     boolean_t autoboot;
5057     int err;

5059     if ((err = zoncfg_get_autoboot(handle, &autoboot)) == Z_OK)
5060         (void) fprintf(fp, "%s: %s\n", pt_to_str(PT_AUTOBOOT),
5061             autoboot ? "true" : "false");
5062     else
5063         zone_perror(zone, err, B_TRUE);
5064 }

5066 static void
5067 info_pool(zone_dochandle_t handle, FILE *fp)
5068 {
5069     char pool[MAXNAMELEN];
5070     int err;

```

```

5072     if ((err = zoncfg_get_pool(handle, pool, sizeof (pool))) == Z_OK)
5073         (void) fprintf(fp, "%s: %s\n", pt_to_str(PT_POOL), pool);
5074     else
5075         zone_perror(zone, err, B_TRUE);
5076 }

5078 static void
5079 info_limitpriv(zone_dochandle_t handle, FILE *fp)
5080 {
5081     char *limitpriv;
5082     int err;

5084     if ((err = zoncfg_get_limitpriv(handle, &limitpriv)) == Z_OK) {
5085         (void) fprintf(fp, "%s: %s\n", pt_to_str(PT_LIMITPRIV),
5086             limitpriv);
5087         free(limitpriv);
5088     } else {
5089         zone_perror(zone, err, B_TRUE);
5090     }
5091 }

5093 static void
5094 info_bootargs(zone_dochandle_t handle, FILE *fp)
5095 {
5096     char bootargs[BOOTARGS_MAX];
5097     int err;

5099     if ((err = zoncfg_get_bootargs(handle, bootargs,
5100         sizeof (bootargs))) == Z_OK) {
5101         (void) fprintf(fp, "%s: %s\n", pt_to_str(PT_BOOTARGS),
5102             bootargs);
5103     } else {
5104         zone_perror(zone, err, B_TRUE);
5105     }
5106 }

5108 static void
5109 info_sched(zone_dochandle_t handle, FILE *fp)
5110 {
5111     char sched[MAXNAMELEN];
5112     int err;

5114     if ((err = zoncfg_get_sched_class(handle, sched, sizeof (sched)))
5115         == Z_OK) {
5116         (void) fprintf(fp, "%s: %s\n", pt_to_str(PT_SCHED), sched);
5117     } else {
5118         zone_perror(zone, err, B_TRUE);
5119     }
5120 }

5122 static void
5123 info_iptype(zone_dochandle_t handle, FILE *fp)
5124 {
5125     zone_ipype_t iptype;
5126     int err;

5128     if ((err = zoncfg_get_ipype(handle, &iptype)) == Z_OK) {
5129         switch (iptype) {
5130             case ZS_SHARED:
5131                 (void) fprintf(fp, "%s: %s\n", pt_to_str(PT_IPTYPE),
5132                     "shared");
5133                 break;
5134             case ZS_EXCLUSIVE:
5135                 (void) fprintf(fp, "%s: %s\n", pt_to_str(PT_IPTYPE),
5136                     "exclusive");

```

```

5137         break;
5138     }
5139 } else {
5140     zone_perror(zone, err, B_TRUE);
5141 }
5142 }

5144 static void
5145 info_hostid(zone_dochandle_t handle, FILE *fp)
5146 {
5147     char hostidp[HW_HOSTID_LEN];
5148     int err;

5150     if ((err = zoncfg_get_hostid(handle, hostidp,
5151         sizeof(hostidp))) == Z_OK) {
5152         (void) fprintf(fp, "%s: %s\n", pt_to_str(PT_HOSTID), hostidp);
5153     } else if (err == Z_BAD_PROPERTY) {
5154         (void) fprintf(fp, "%s: \n", pt_to_str(PT_HOSTID));
5155     } else {
5156         zone_perror(zone, err, B_TRUE);
5157     }
5158 }

5160 static void
5161 info_fs_allowed(zone_dochandle_t handle, FILE *fp)
5162 {
5163     char fsallowedp[ZONE_FS_ALLOWED_MAX];
5164     int err;

5166     if ((err = zoncfg_get_fs_allowed(handle, fsallowedp,
5167         sizeof(fsallowedp))) == Z_OK) {
5168         (void) fprintf(fp, "%s: %s\n", pt_to_str(PT_FS_ALLOWED),
5169             fsallowedp);
5170     } else if (err == Z_BAD_PROPERTY) {
5171         (void) fprintf(fp, "%s: \n", pt_to_str(PT_FS_ALLOWED));
5172     } else {
5173         zone_perror(zone, err, B_TRUE);
5174     }
5175 }

5177 static void
5178 output_fs(FILE *fp, struct zone_fstab *fstab)
5179 {
5180     zone_fsopt_t *this;

5182     (void) fprintf(fp, "%s:\n", rt_to_str(RT_FS));
5183     output_prop(fp, PT_DIR, fstab->zone_fs_dir, B_TRUE);
5184     output_prop(fp, PT_SPECIAL, fstab->zone_fs_special, B_TRUE);
5185     output_prop(fp, PT_RAW, fstab->zone_fs_raw, B_TRUE);
5186     output_prop(fp, PT_TYPE, fstab->zone_fs_type, B_TRUE);
5187     (void) fprintf(fp, "\t%s: [", pt_to_str(PT_OPTIONS));
5188     for (this = fstab->zone_fs_options; this != NULL;
5189         this = this->zone_fsopt_next) {
5190         if (strchr(this->zone_fsopt_opt, '='))
5191             (void) fprintf(fp, "%s", this->zone_fsopt_opt);
5192         else
5193             (void) fprintf(fp, "%s", this->zone_fsopt_opt);
5194         if (this->zone_fsopt_next != NULL)
5195             (void) fprintf(fp, ",");
5196     }
5197     (void) fprintf(fp, "]\n");
5198 }

5200 static void
5201 info_net(zone_dochandle_t handle, FILE *fp, cmd_t *cmd)
5202 {

```

```

5203     struct zone_fstab lookup, user;
5204     boolean_t output = B_FALSE;

5206     if (zoncfg_setfsent(handle) != Z_OK)
5207         return;
5208     while (zoncfg_getfsent(handle, &lookup) == Z_OK) {
5209         if (cmd->cmd_prop_nv_pairs == 0) {
5210             output_fs(fp, &lookup);
5211             goto loopend;
5212         }
5213         if (fill_in_fstab(cmd, &user, B_TRUE) != Z_OK)
5214             goto loopend;
5215         if (strlen(user.zone_fs_dir) > 0 &&
5216             strcmp(user.zone_fs_dir, lookup.zone_fs_dir) != 0)
5217             goto loopend; /* no match */
5218         if (strlen(user.zone_fs_special) > 0 &&
5219             strcmp(user.zone_fs_special, lookup.zone_fs_special) != 0)
5220             goto loopend; /* no match */
5221         if (strlen(user.zone_fs_type) > 0 &&
5222             strcmp(user.zone_fs_type, lookup.zone_fs_type) != 0)
5223             goto loopend; /* no match */
5224         output_fs(fp, &lookup);
5225         output = B_TRUE;
5226     loopend:
5227         zoncfg_free_fs_option_list(lookup.zone_fs_options);
5228     }
5229     (void) zoncfg_endfsent(handle);
5230     /*
5231      * If a property n/v pair was specified, warn the user if there was
5232      * nothing to output.
5233      */
5234     if (!output && cmd->cmd_prop_nv_pairs > 0)
5235         (void) printf(gettext("No such %s resource.\n"),
5236             rt_to_str(RT_FS));
5237 }

5239 static void
5240 output_net(FILE *fp, struct zone_nwifstab *nwifstab)
5241 {
5242     (void) fprintf(fp, "%s:\n", rt_to_str(RT_NET));
5243     output_prop(fp, PT_ADDRESS, nwifstab->zone_nwif_address, B_TRUE);
5244     output_prop(fp, PT_ALLOWED_ADDRESS,
5245         nwifstab->zone_nwif_allowed_address, B_TRUE);
5246     output_prop(fp, PT_PHYSICAL, nwifstab->zone_nwif_physical, B_TRUE);
5247     output_prop(fp, PT_DEFROUTER, nwifstab->zone_nwif_defrouter, B_TRUE);
5248 }

5250 static void
5251 info_net(zone_dochandle_t handle, FILE *fp, cmd_t *cmd)
5252 {
5253     struct zone_nwifstab lookup, user;
5254     boolean_t output = B_FALSE;

5256     if (zoncfg_setnwifent(handle) != Z_OK)
5257         return;
5258     while (zoncfg_getnwifent(handle, &lookup) == Z_OK) {
5259         if (cmd->cmd_prop_nv_pairs == 0) {
5260             output_net(fp, &lookup);
5261             continue;
5262         }
5263         if (fill_in_nwifstab(cmd, &user, B_TRUE) != Z_OK)
5264             continue;
5265         if (strlen(user.zone_nwif_physical) > 0 &&
5266             strcmp(user.zone_nwif_physical,
5267                 lookup.zone_nwif_physical) != 0)
5268             continue; /* no match */

```

```

5269     /* If present make sure it matches */
5270     if (strlen(user.zone_nwif_address) > 0 &&
5271         !zonecfg_same_net_address(user.zone_nwif_address,
5272             lookup.zone_nwif_address))
5273         continue; /* no match */
5274     output_net(fp, &lookup);
5275     output = B_TRUE;
5276 }
5277 (void) zonecfg_endnwifent(handle);
5278 /*
5279  * If a property n/v pair was specified, warn the user if there was
5280  * nothing to output.
5281  */
5282 if (!output && cmd->cmd_prop_nv_pairs > 0)
5283     (void) printf(gettext("No such %s resource.\n"),
5284         rt_to_str(RT_NET));
5285 }

5287 static void
5288 output_dev(FILE *fp, struct zone_devtab *devtab)
5289 {
5290     (void) fprintf(fp, "%s:\n", rt_to_str(RT_DEVICE));
5291     output_prop(fp, PT_MATCH, devtab->zone_dev_match, B_TRUE);
5292 }

5294 static void
5295 info_dev(zone_dochandle_t handle, FILE *fp, cmd_t *cmd)
5296 {
5297     struct zone_devtab lookup, user;
5298     boolean_t output = B_FALSE;

5300     if (zonecfg_setdevent(handle) != Z_OK)
5301         return;
5302     while (zonecfg_getdevent(handle, &lookup) == Z_OK) {
5303         if (cmd->cmd_prop_nv_pairs == 0) {
5304             output_dev(fp, &lookup);
5305             continue;
5306         }
5307         if (fill_in_devtab(cmd, &user, B_TRUE) != Z_OK)
5308             continue;
5309         if (strlen(user.zone_dev_match) > 0 &&
5310             strcmp(user.zone_dev_match, lookup.zone_dev_match) != 0)
5311             continue; /* no match */
5312         output_dev(fp, &lookup);
5313         output = B_TRUE;
5314     }
5315     (void) zonecfg_enddevent(handle);
5316     /*
5317      * If a property n/v pair was specified, warn the user if there was
5318      * nothing to output.
5319      */
5320     if (!output && cmd->cmd_prop_nv_pairs > 0)
5321         (void) printf(gettext("No such %s resource.\n"),
5322             rt_to_str(RT_DEVICE));
5323 }

5325 static void
5326 output_rctl(FILE *fp, struct zone_rctltab *rctltab)
5327 {
5328     struct zone_rctlvaltab *valptr;

5330     (void) fprintf(fp, "%s:\n", rt_to_str(RT_RCTL));
5331     output_prop(fp, PT_NAME, rctltab->zone_rctl_name, B_TRUE);
5332     for (valptr = rctltab->zone_rctl_valptr; valptr != NULL;
5333         valptr = valptr->zone_rctlval_next) {
5334         fprintf(fp, "\t%s: (%s=%s,%s=%s,%s=%s)\n",

```

```

5335         pt_to_str(PT_VALUE),
5336         pt_to_str(PT_PRIV), valptr->zone_rctlval_priv,
5337         pt_to_str(PT_LIMIT), valptr->zone_rctlval_limit,
5338         pt_to_str(PT_ACTION), valptr->zone_rctlval_action);
5339     }
5340 }

5342 static void
5343 info_rctl(zone_dochandle_t handle, FILE *fp, cmd_t *cmd)
5344 {
5345     struct zone_rctltab lookup, user;
5346     boolean_t output = B_FALSE;

5348     if (zonecfg_setrctlent(handle) != Z_OK)
5349         return;
5350     while (zonecfg_getrctlent(handle, &lookup) == Z_OK) {
5351         if (cmd->cmd_prop_nv_pairs == 0) {
5352             output_rctl(fp, &lookup);
5353         } else if (fill_in_rctltab(cmd, &user, B_TRUE) == Z_OK &&
5354             (strlen(user.zone_rctl_name) == 0 ||
5355             strcmp(user.zone_rctl_name, lookup.zone_rctl_name) == 0)) {
5356             output_rctl(fp, &lookup);
5357             output = B_TRUE;
5358         }
5359         zonecfg_free_rctl_value_list(lookup.zone_rctl_valptr);
5360     }
5361     (void) zonecfg_endrctlent(handle);
5362     /*
5363      * If a property n/v pair was specified, warn the user if there was
5364      * nothing to output.
5365      */
5366     if (!output && cmd->cmd_prop_nv_pairs > 0)
5367         (void) printf(gettext("No such %s resource.\n"),
5368             rt_to_str(RT_RCTL));
5369 }

5371 static void
5372 output_attr(FILE *fp, struct zone_attrtab *attrtab)
5373 {
5374     (void) fprintf(fp, "%s:\n", rt_to_str(RT_ATTR));
5375     output_prop(fp, PT_NAME, attrtab->zone_attr_name, B_TRUE);
5376     output_prop(fp, PT_TYPE, attrtab->zone_attr_type, B_TRUE);
5377     output_prop(fp, PT_VALUE, attrtab->zone_attr_value, B_TRUE);
5378 }

5380 static void
5381 info_attr(zone_dochandle_t handle, FILE *fp, cmd_t *cmd)
5382 {
5383     struct zone_attrtab lookup, user;
5384     boolean_t output = B_FALSE;

5386     if (zonecfg_setattrent(handle) != Z_OK)
5387         return;
5388     while (zonecfg_getattrent(handle, &lookup) == Z_OK) {
5389         if (cmd->cmd_prop_nv_pairs == 0) {
5390             output_attr(fp, &lookup);
5391             continue;
5392         }
5393         if (fill_in_attrtab(cmd, &user, B_TRUE) != Z_OK)
5394             continue;
5395         if (strlen(user.zone_attr_name) > 0 &&
5396             strcmp(user.zone_attr_name, lookup.zone_attr_name) != 0)
5397             continue; /* no match */
5398         if (strlen(user.zone_attr_type) > 0 &&
5399             strcmp(user.zone_attr_type, lookup.zone_attr_type) != 0)
5400             continue; /* no match */

```

```

5401         if (strlen(user.zone_attr_value) > 0 &&
5402             strcmp(user.zone_attr_value, lookup.zone_attr_value) != 0)
5403             continue; /* no match */
5404         output_attr(fp, &lookup);
5405         output = B_TRUE;
5406     }
5407     (void) zonecfg_endattrent(handle);
5408     /*
5409     * If a property n/v pair was specified, warn the user if there was
5410     * nothing to output.
5411     */
5412     if (!output && cmd->cmd_prop_nv_pairs > 0)
5413         (void) printf(gettext("No such %s resource.\n"),
5414             rt_to_str(RT_ATTR));
5415 }

5417 static void
5418 output_ds(FILE *fp, struct zone_dstab *dstab)
5419 {
5420     (void) fprintf(fp, "%s:\n", rt_to_str(RT_DATASET));
5421     output_prop(fp, PT_NAME, dstab->zone_dataset_name, B_TRUE);
5422 }

5424 static void
5425 info_ds(zone_dochandle_t handle, FILE *fp, cmd_t *cmd)
5426 {
5427     struct zone_dstab lookup;
5428     boolean_t output = B_FALSE;

5430     if (zonecfg_setdsent(handle) != Z_OK)
5431         return;
5432     while (zonecfg_getdsent(handle, &lookup) == Z_OK) {
5433         if (cmd->cmd_prop_nv_pairs == 0) {
5434             output_ds(fp, &lookup);
5435             continue;
5436         }
5437         if (fill_in_dstab(cmd, &user, B_TRUE) != Z_OK)
5438             continue;
5439         if (strlen(user.zone_dataset_name) > 0 &&
5440             strcmp(user.zone_dataset_name,
5441                 lookup.zone_dataset_name) != 0)
5442             continue; /* no match */
5443         output_ds(fp, &lookup);
5444         output = B_TRUE;
5445     }
5446     (void) zonecfg_enddsent(handle);
5447     /*
5448     * If a property n/v pair was specified, warn the user if there was
5449     * nothing to output.
5450     */
5451     if (!output && cmd->cmd_prop_nv_pairs > 0)
5452         (void) printf(gettext("No such %s resource.\n"),
5453             rt_to_str(RT_DATASET));
5454 }

5456 static void
5457 output_pset(FILE *fp, struct zone_psettab *psettab)
5458 {
5459     (void) fprintf(fp, "%s:\n", rt_to_str(RT_DCPU));
5460     if (strcmp(psettab->zone_ncpu_min, psettab->zone_ncpu_max) == 0)
5461         (void) fprintf(fp, "\t%s: %s\n", pt_to_str(PT_NCPUS),
5462             psettab->zone_ncpu_max);
5463     else
5464         (void) fprintf(fp, "\t%s: %s-%s\n", pt_to_str(PT_NCPUS),
5465             psettab->zone_ncpu_min, psettab->zone_ncpu_max);
5466     if (psettab->zone_importance[0] != '\0')

```

```

5467         (void) fprintf(fp, "\t%s: %s\n", pt_to_str(PT_IMPORTANCE),
5468             psettab->zone_importance);
5469     }

5471 static void
5472 info_pset(zone_dochandle_t handle, FILE *fp)
5473 {
5474     struct zone_psettab lookup;

5476     if (zonecfg_getpsetent(handle, &lookup) == Z_OK)
5477         output_pset(fp, &lookup);
5478 }

5480 static void
5481 output_pcap(FILE *fp)
5482 {
5483     uint64_t cap;

5485     if (zonecfg_get_aliased_rctl(handle, ALIAS_CPUCAP, &cap) == Z_OK) {
5486         float scaled = (float)cap / 100;
5487         (void) fprintf(fp, "%s:\n", rt_to_str(RT_PCAP));
5488         (void) fprintf(fp, "\t[%s: %.2f]\n", pt_to_str(PT_NCPUS),
5489             scaled);
5490     }
5491 }

5493 static void
5494 info_pcap(FILE *fp)
5495 {
5496     output_pcap(fp);
5497 }

5500 static void
5501 info_aliased_rctl(zone_dochandle_t handle, FILE *fp, char *alias)
5502 {
5503     uint64_t limit;

5505     if (zonecfg_get_aliased_rctl(handle, alias, &limit) == Z_OK) {
5506         /* convert memory based properties */
5507         if (strcmp(alias, ALIAS_MAXSHMMEM) == 0) {
5508             char buf[128];

5510             (void) snprintf(buf, sizeof (buf), "%llu", limit);
5511             bytes_to_units(buf, buf, sizeof (buf));
5512             (void) fprintf(fp, "[%s: %s]\n", alias, buf);
5513             return;
5514         }

5516         (void) fprintf(fp, "[%s: %llu]\n", alias, limit);
5517     }
5518 }

5520 static void
5521 bytes_to_units(char *str, char *buf, int bufsize)
5522 {
5523     unsigned long long num;
5524     unsigned long long save = 0;
5525     char *units = "BKMG";
5526     char *up = units;

5528     num = strtoll(str, NULL, 10);

5530     if (num < 1024) {
5531         (void) snprintf(buf, bufsize, "%llu", num);
5532         return;

```

```

5533     }
5534
5535     while ((num >= 1024) && (*up != 'T')) {
5536         up++; /* next unit of measurement */
5537         save = num;
5538         num = (num + 512) >> 10;
5539     }
5540
5541     /* check if we should output a fraction.  snprintf will round for us */
5542     if (save % 1024 != 0 && ((save >> 10) < 10))
5543         (void) snprintf(buf, bufsize, "%.1f%c", ((float)save / 1024),
5544             *up);
5545     else
5546         (void) snprintf(buf, bufsize, "%llu%c", num, *up);
5547 }
5548
5549 static void
5550 output_mcap(FILE *fp, struct zone_mcaptab *mcaptab, int showswap,
5551     uint64_t maxswap, int showlocked, uint64_t maxlocked)
5552 {
5553     char buf[128];
5554
5555     (void) fprintf(fp, "%s:\n", rt_to_str(RT_MCAP));
5556     if (mcaptab->zone_physmem_cap[0] != '\0') {
5557         bytes_to_units(mcaptab->zone_physmem_cap, buf, sizeof (buf));
5558         output_prop(fp, PT_PHYSICAL, buf, B_TRUE);
5559     }
5560
5561     if (showswap == Z_OK) {
5562         (void) snprintf(buf, sizeof (buf), "%llu", maxswap);
5563         bytes_to_units(buf, buf, sizeof (buf));
5564         output_prop(fp, PT_SWAP, buf, B_TRUE);
5565     }
5566
5567     if (showlocked == Z_OK) {
5568         (void) snprintf(buf, sizeof (buf), "%llu", maxlocked);
5569         bytes_to_units(buf, buf, sizeof (buf));
5570         output_prop(fp, PT_LOCKED, buf, B_TRUE);
5571     }
5572 }
5573
5574 static void
5575 info_mcap(zone_dochandle_t handle, FILE *fp)
5576 {
5577     int res1, res2, res3;
5578     uint64_t swap_limit;
5579     uint64_t locked_limit;
5580     struct zone_mcaptab lookup;
5581
5582     bzero(&lookup, sizeof (lookup));
5583     res1 = zoncfg_getmcapent(handle, &lookup);
5584     res2 = zoncfg_get_aliased_rctl(handle, ALIAS_MAXSWAP, &swap_limit);
5585     res3 = zoncfg_get_aliased_rctl(handle, ALIAS_MAXLOCKEDMEM,
5586         &locked_limit);
5587
5588     if (res1 == Z_OK || res2 == Z_OK || res3 == Z_OK)
5589         output_mcap(fp, &lookup, res2, swap_limit, res3, locked_limit);
5590 }
5591
5592 static void
5593 output_auth(FILE *fp, struct zone_admintab *admintab)
5594 {
5595     (void) fprintf(fp, "%s:\n", rt_to_str(RT_ADMIN));
5596     output_prop(fp, PT_USER, admintab->zone_admin_user, B_TRUE);
5597     output_prop(fp, PT_AUTHS, admintab->zone_admin_auths, B_TRUE);
5598 }

```

```

5600 static void
5601 output_secflags(FILE *fp, struct zone_secflagstab *sftab)
5602 {
5603     (void) fprintf(fp, "%s:\n", rt_to_str(RT_SECFLAGS));
5604     output_prop(fp, PT_DEFAULT, sftab->zone_secflags_default, B_TRUE);
5605     output_prop(fp, PT_LOWER, sftab->zone_secflags_lower, B_TRUE);
5606     output_prop(fp, PT_UPPER, sftab->zone_secflags_upper, B_TRUE);
5607 }
5608
5609 static void
5610 #endif /* ! codereview */
5611 info_auth(zone_dochandle_t handle, FILE *fp, cmd_t *cmd)
5612 {
5613     struct zone_admintab lookup, user;
5614     boolean_t output = B_FALSE;
5615     int err;
5616
5617     if ((err = zoncfg_setadminent(handle)) != Z_OK) {
5618         zone_perror(zone, err, B_TRUE);
5619         return;
5620     }
5621     while (zoncfg_getadminent(handle, &lookup) == Z_OK) {
5622         if (cmd->cmd_prop_nv_pairs == 0) {
5623             output_auth(fp, &lookup);
5624             continue;
5625         }
5626         if (fill_in_admintab(cmd, &user, B_TRUE) != Z_OK)
5627             continue;
5628         if (strlen(user.zone_admin_user) > 0 &&
5629             strcmp(user.zone_admin_user, lookup.zone_admin_user) != 0)
5630             continue; /* no match */
5631         output_auth(fp, &lookup);
5632         output = B_TRUE;
5633     }
5634     (void) zoncfg_endadminent(handle);
5635     /*
5636      * If a property n/v pair was specified, warn the user if there was
5637      * nothing to output.
5638      */
5639     if (!output && cmd->cmd_prop_nv_pairs > 0)
5640         (void) printf(gettext("No such %s resource.\n"),
5641             rt_to_str(RT_ADMIN));
5642 }
5643
5644 static void
5645 info_secflags(zone_dochandle_t handle, FILE *fp)
5646 {
5647     struct zone_secflagstab sftab;
5648
5649     if (zoncfg_lookup_secflags(handle, &sftab) == Z_OK) {
5650         output_secflags(fp, &sftab);
5651     }
5652 }
5653
5654 #endif /* ! codereview */
5655 void
5656 info_func(cmd_t *cmd)
5657 {
5658     FILE *fp = stdout;
5659     boolean_t need_to_close = B_FALSE;
5660     int type;
5661     int res1, res2;
5662     uint64_t swap_limit;
5663     uint64_t locked_limit;

```

```

5665     assert(cmd != NULL);
5666
5667     if (initialize(B_TRUE) != Z_OK)
5668         return;
5670
5671     /* don't page error output */
5672     if (interactive_mode) {
5673         if ((fp = pager_open()) != NULL)
5674             need_to_close = B_TRUE;
5675         else
5676             fp = stdout;
5677
5678         setbuf(fp, NULL);
5679     }
5680
5681     if (!global_scope) {
5682         switch (resource_scope) {
5683             case RT_FS:
5684                 output_fs(fp, &in_progress_fstab);
5685                 break;
5686             case RT_NET:
5687                 output_net(fp, &in_progress_nwifstab);
5688                 break;
5689             case RT_DEVICE:
5690                 output_dev(fp, &in_progress_devtab);
5691                 break;
5692             case RT_RCTL:
5693                 output_rctl(fp, &in_progress_rctltab);
5694                 break;
5695             case RT_ATTR:
5696                 output_attr(fp, &in_progress_attrtab);
5697                 break;
5698             case RT_DATASET:
5699                 output_ds(fp, &in_progress_dstab);
5700                 break;
5701             case RT_DCPU:
5702                 output_pset(fp, &in_progress_psettab);
5703                 break;
5704             case RT_PCAP:
5705                 output_pcap(fp);
5706                 break;
5707             case RT_MCAP:
5708                 res1 = zoncfg_get_aliased_rctl(handle, ALIAS_MAXSWAP,
5709                 &swap_limit);
5710                 res2 = zoncfg_get_aliased_rctl(handle,
5711                 ALIAS_MAXLOCKEDMEM, &locked_limit);
5712                 output_mcap(fp, &in_progress_mcaptab, res1, swap_limit,
5713                 res2, locked_limit);
5714                 break;
5715             case RT_ADMIN:
5716                 output_auth(fp, &in_progress_admintab);
5717                 break;
5718             case RT_SECFLAGS:
5719                 output_secflags(fp, &in_progress_secflagstab);
5720                 break;
5721         }
5722         goto cleanup;
5723     }
5725
5726     type = cmd->cmd_res_type;
5727
5728     if (gz_invalid_rt_property(type)) {
5729         zerr(gettext("%s is not a valid property for the global zone."),
5730         rt_to_str(type));
5731         goto cleanup;

```

```

5732     }
5733
5734     if (gz_invalid_resource(type)) {
5735         zerr(gettext("%s is not a valid resource for the global zone."),
5736         rt_to_str(type));
5737         goto cleanup;
5738     }
5739
5740     switch (cmd->cmd_res_type) {
5741     case RT_UNKNOWN:
5742         info_zonename(handle, fp);
5743         if (!global_zone) {
5744             info_zonepath(handle, fp);
5745             info_brand(handle, fp);
5746             info_autoboot(handle, fp);
5747             info_bootargs(handle, fp);
5748         }
5749         info_pool(handle, fp);
5750         if (!global_zone) {
5751             info_limitpriv(handle, fp);
5752             info_sched(handle, fp);
5753             info_iptype(handle, fp);
5754             info_hostid(handle, fp);
5755             info_fs_allowed(handle, fp);
5756         }
5757         info_aliased_rctl(handle, fp, ALIAS_MAXLWPS);
5758         info_aliased_rctl(handle, fp, ALIAS_MAXPROCS);
5759         info_aliased_rctl(handle, fp, ALIAS_MAXSHMMEM);
5760         info_aliased_rctl(handle, fp, ALIAS_MAXSHMIDS);
5761         info_aliased_rctl(handle, fp, ALIAS_MAXMSGIDS);
5762         info_aliased_rctl(handle, fp, ALIAS_MAXSEMIDS);
5763         info_aliased_rctl(handle, fp, ALIAS_SHARES);
5764         if (!global_zone) {
5765             info_fs(handle, fp, cmd);
5766             info_net(handle, fp, cmd);
5767             info_dev(handle, fp, cmd);
5768         }
5769         info_pset(handle, fp);
5770         info_pcap(fp);
5771         info_mcap(handle, fp);
5772         if (!global_zone) {
5773             info_attr(handle, fp, cmd);
5774             info_ds(handle, fp, cmd);
5775             info_auth(handle, fp, cmd);
5776         }
5777         info_rctl(handle, fp, cmd);
5778         info_secflags(handle, fp);
5779     #endif /* ! codereview */
5780     case RT_ZONENAME:
5781         info_zonename(handle, fp);
5782         break;
5783     case RT_ZONEPATH:
5784         info_zonepath(handle, fp);
5785         break;
5786     case RT_BRAND:
5787         info_brand(handle, fp);
5788         break;
5789     case RT_AUTOBOOT:
5790         info_autoboot(handle, fp);
5791         break;
5792     case RT_POOL:
5793         info_pool(handle, fp);
5794         break;
5795     case RT_LIMITPRIV:
5796         info_limitpriv(handle, fp);

```



```

5797         break;
5798     case RT_BOOTARGS:
5799         info_bootargs(handle, fp);
5800         break;
5801     case RT_SCHED:
5802         info_sched(handle, fp);
5803         break;
5804     case RT_IPTYPE:
5805         info_ipctype(handle, fp);
5806         break;
5807     case RT_MAXLWPS:
5808         info_aliased_rctl(handle, fp, ALIAS_MAXLWPS);
5809         break;
5810     case RT_MAXPROCS:
5811         info_aliased_rctl(handle, fp, ALIAS_MAXPROCS);
5812         break;
5813     case RT_MAXSHMEM:
5814         info_aliased_rctl(handle, fp, ALIAS_MAXSHMEM);
5815         break;
5816     case RT_MAXSHMIDS:
5817         info_aliased_rctl(handle, fp, ALIAS_MAXSHMIDS);
5818         break;
5819     case RT_MAXMSGIDS:
5820         info_aliased_rctl(handle, fp, ALIAS_MAXMSGIDS);
5821         break;
5822     case RT_MAXSEMIDS:
5823         info_aliased_rctl(handle, fp, ALIAS_MAXSEMIDS);
5824         break;
5825     case RT_SHARES:
5826         info_aliased_rctl(handle, fp, ALIAS_SHARES);
5827         break;
5828     case RT_FS:
5829         info_fs(handle, fp, cmd);
5830         break;
5831     case RT_NET:
5832         info_net(handle, fp, cmd);
5833         break;
5834     case RT_DEVICE:
5835         info_dev(handle, fp, cmd);
5836         break;
5837     case RT_RCTL:
5838         info_rctl(handle, fp, cmd);
5839         break;
5840     case RT_ATTR:
5841         info_attr(handle, fp, cmd);
5842         break;
5843     case RT_DATASET:
5844         info_ds(handle, fp, cmd);
5845         break;
5846     case RT_DCPU:
5847         info_pset(handle, fp);
5848         break;
5849     case RT_PCAP:
5850         info_pcap(fp);
5851         break;
5852     case RT_MCAP:
5853         info_mcap(handle, fp);
5854         break;
5855     case RT_HOSTID:
5856         info_hostid(handle, fp);
5857         break;
5858     case RT_ADMIN:
5859         info_auth(handle, fp, cmd);
5860         break;
5861     case RT_FS_ALLOWED:
5862         info_fs_allowed(handle, fp);

```

```

5863         break;
5864     case RT_SECFLAGS:
5865         info_secflags(handle, fp);
5866         break;
5867 #endif /* ! codereview */
5868     default:
5869         zone_perror(rt_to_str(cmd->cmd_res_type), Z_NO_RESOURCE_TYPE,
5870                     B_TRUE);
5871     }
5872
5873 cleanup:
5874     if (need_to_close)
5875         (void) pager_close(fp);
5876 }
5877
5878 /*
5879  * Helper function for verify-- checks that a required string property
5880  * exists.
5881  */
5882 static void
5883 check_reqd_prop(char *attr, int rt, int pt, int *ret_val)
5884 {
5885     if (strlen(attr) == 0) {
5886         zerr(gettext("%s: %s not specified"), rt_to_str(rt),
5887              pt_to_str(pt));
5888         saw_error = B_TRUE;
5889         if (*ret_val == Z_OK)
5890             *ret_val = Z_REQD_PROPERTY_MISSING;
5891     }
5892 }
5893
5894 static int
5895 do_subproc(char *cmdbuf)
5896 {
5897     char inbuf[MAX_CMD_LEN];
5898     FILE *file;
5899     int status;
5900
5901     file = popen(cmdbuf, "r");
5902     if (file == NULL) {
5903         zerr(gettext("Could not launch: %s"), cmdbuf);
5904         return (-1);
5905     }
5906
5907     while (fgets(inbuf, sizeof (inbuf), file) != NULL)
5908         fprintf(stderr, "%s", inbuf);
5909     status = pclose(file);
5910
5911     if (WIFSIGNALED(status) {
5912         zerr(gettext("%s unexpectedly terminated due to signal %d"),
5913              cmdbuf, WTERMSIG(status));
5914         return (-1);
5915     }
5916     assert(WIFEXITED(status));
5917     return (WEXITSTATUS(status));
5918 }
5919
5920 static int
5921 brand_verify(zone_dochandle_t handle)
5922 {
5923     char xml_file[32];
5924     char cmdbuf[MAX_CMD_LEN];
5925     brand_handle_t bh;
5926     char brand[MAXNAMELEN];
5927     int err;

```

```

5929     if (zonedcfg_get_brand(handle, brand, sizeof (brand)) != Z_OK) {
5930         zerr("%s: %s\n", zone, gettext("could not get zone brand"));
5931         return (Z_INVALID_DOCUMENT);
5932     }
5933     if ((bh = brand_open(brand)) == NULL) {
5934         zerr("%s: %s\n", zone, gettext("unknown brand."));
5935         return (Z_INVALID_DOCUMENT);
5936     }
5937
5938     /*
5939     * Fetch the verify command, if any, from the brand configuration
5940     * and build the command line to execute it.
5941     */
5942     strcpy(cmdbuf, EXEC_PREFIX);
5943     err = brand_get_verify_cfg(bh, cmdbuf + EXEC_LEN,
5944         sizeof (cmdbuf) - (EXEC_LEN + (strlen(xml_file) + 1)));
5945     brand_close(bh);
5946     if (err != Z_OK) {
5947         zerr("%s: %s\n", zone,
5948             gettext("could not get brand verification command"));
5949         return (Z_INVALID_DOCUMENT);
5950     }
5951
5952     /*
5953     * If the brand doesn't provide a verification routine, we just
5954     * return success.
5955     */
5956     if (strlen(cmdbuf) == EXEC_LEN)
5957         return (Z_OK);
5958
5959     /*
5960     * Dump the current config information for this zone to a file.
5961     */
5962     strcpy(xml_file, "/tmp/zonedcfg_verify.XXXXXX");
5963     if (mkstemp(xml_file) == NULL)
5964         return (Z_TEMP_FILE);
5965     if ((err = zonedcfg_verify_save(handle, xml_file)) != Z_OK) {
5966         (void) unlink(xml_file);
5967         return (err);
5968     }
5969
5970     /*
5971     * Execute the verification command.
5972     */
5973     if ((strlen(cmdbuf, " ", MAX_CMD_LEN) >= MAX_CMD_LEN) ||
5974         (strlen(cmdbuf, xml_file, MAX_CMD_LEN) >= MAX_CMD_LEN)) {
5975         err = Z_BRAND_ERROR;
5976     } else {
5977         err = do_subproc(cmdbuf);
5978     }
5979
5980     (void) unlink(xml_file);
5981     return ((err == Z_OK) ? Z_OK : Z_BRAND_ERROR);
5982 }
5983
5984 /*
5985 * Track the network interfaces listed in zonedcfg(1m) in a linked list
5986 * so that we can later check that defrouter is specified for an exclusive IP
5987 * zone if and only if at least one allowed-address has been specified.
5988 */
5989 static boolean_t
5990 add_nwif(struct zone_nwifstab *nwif)
5991 {
5992     struct xif *tmp;
5993
5994     for (tmp = xif; tmp != NULL; tmp = tmp->xif_next) {

```

```

5995         if (strcmp(tmp->xif_name, nwif->zone_nwif_physical) == 0) {
5996             if (strlen(nwif->zone_nwif_allowed_address) > 0)
5997                 tmp->xif_has_address = B_TRUE;
5998             if (strlen(nwif->zone_nwif_defrouter) > 0)
5999                 tmp->xif_has_defrouter = B_TRUE;
6000             return (B_TRUE);
6001         }
6002     }
6003
6004     tmp = malloc(sizeof (*tmp));
6005     if (tmp == NULL) {
6006         zerr(gettext("memory allocation failed for %s"),
6007             nwif->zone_nwif_physical);
6008         return (B_FALSE);
6009     }
6010     strcpy(tmp->xif_name, nwif->zone_nwif_physical,
6011         sizeof (tmp->xif_name));
6012     tmp->xif_has_defrouter = (strlen(nwif->zone_nwif_defrouter) > 0);
6013     tmp->xif_has_address = (strlen(nwif->zone_nwif_allowed_address) > 0);
6014     tmp->xif_next = xif;
6015     xif = tmp;
6016     return (B_TRUE);
6017 }
6018
6019 boolean_t
6020 verify_secflags(struct zone_secflagstab *tab)
6021 {
6022     secflagdelta_t def = {0};
6023     secflagdelta_t upper = {0};
6024     secflagdelta_t lower = {0};
6025     boolean_t def_set = B_FALSE;
6026     boolean_t upper_set = B_FALSE;
6027     boolean_t lower_set = B_FALSE;
6028     boolean_t ret = B_TRUE;
6029
6030     if (strlen(tab->zone_secflags_default) > 0) {
6031         def_set = B_TRUE;
6032         if (secflags_parse(NULL, tab->zone_secflags_default,
6033             &def) == -1) {
6034             zerr(gettext("default security flags '%s' are invalid"),
6035                 tab->zone_secflags_default);
6036             ret = B_FALSE;
6037         }
6038     } else {
6039         secflags_zero(&def.psd_assign);
6040         def.psd_ass_active = B_TRUE;
6041     }
6042
6043     if (strlen(tab->zone_secflags_upper) > 0) {
6044         upper_set = B_TRUE;
6045         if (secflags_parse(NULL, tab->zone_secflags_upper,
6046             &upper) == -1) {
6047             zerr(gettext("upper security flags '%s' are invalid"),
6048                 tab->zone_secflags_upper);
6049             ret = B_FALSE;
6050         }
6051     } else {
6052         secflags_fullset(&upper.psd_assign);
6053         upper.psd_ass_active = B_TRUE;
6054     }
6055
6056     if (strlen(tab->zone_secflags_lower) > 0) {
6057         lower_set = B_TRUE;
6058         if (secflags_parse(NULL, tab->zone_secflags_lower,
6059             &lower) == -1) {
6060             zerr(gettext("lower security flags '%s' are invalid"),

```

```

6061         tab->zone_secflags_lower);
6062         ret = B_FALSE;
6063     }
6064     } else {
6065         secflags_zero(&lower.psd_assign);
6066         lower.psd_ass_active = B_TRUE;
6067     }
6069     if (def_set && !def.psd_ass_active) {
6070         zerr(gettext("only assignment of security flags is "
6071             "allowed (default: %s)", tab->zone_secflags_default);
6072     }
6074     if (lower_set && !lower.psd_ass_active) {
6075         zerr(gettext("only assignment of security flags is "
6076             "allowed (lower: %s)", tab->zone_secflags_lower);
6077     }
6079     if (upper_set && !upper.psd_ass_active) {
6080         zerr(gettext("only assignment of security flags is "
6081             "allowed (upper: %s)", tab->zone_secflags_upper);
6082     }
6084     if (def.psd_assign & ~upper.psd_assign) { /* In default but not upper */
6085         zerr(gettext("default secflags must be within the "
6086             "upper limit"));
6087         ret = B_FALSE;
6088     }
6089     if (lower.psd_assign & ~def.psd_assign) { /* In lower but not default */
6090         zerr(gettext("default secflags must be above the lower limit"));
6091         ret = B_FALSE;
6092     }
6093     if (lower.psd_assign & ~upper.psd_assign) { /* In lower but not upper */
6094         zerr(gettext("lower secflags must be within the upper limit"));
6095         ret = B_FALSE;
6096     }
6098     return (ret);
6099 }
6101 #endif /* ! codereview */
6102 /*
6103  * See the DTD for which attributes are required for which resources.
6104  *
6105  * This function can be called by commit_func(), which needs to save things,
6106  * in addition to the general call from parse_and_run(), which doesn't need
6107  * things saved. Since the parameters are standardized, we distinguish by
6108  * having commit_func() call here with cmd->cmd_arg set to "save" to indicate
6109  * that a save is needed.
6110  */
6111 void
6112 verify_func(cmd_t *cmd)
6113 {
6114     struct zone_nwiftab nwiftab;
6115     struct zone_fstab fstab;
6116     struct zone_attrtab attrtab;
6117     struct zone_rctltab rctltab;
6118     struct zone_dstab dstab;
6119     struct zone_psettab psettab;
6120     struct zone_adminstab adminstab;
6121     struct zone_secflagstab secflagstab;
6122 #endif /* ! codereview */
6123     char zonepath[MAXPATHLEN];
6124     char sched[MAXNAMELEN];
6125     char brand[MAXNAMELEN];
6126     char hostidp[HW_HOSTID_LEN];

```

```

6127     char fsallowedp[ZONE_FS_ALLOWED_MAX];
6128     priv_set_t *privs;
6129     char *privname = NULL;
6130     int err, ret_val = Z_OK, arg;
6131     int pset_res;
6132     boolean_t save = B_FALSE;
6133     boolean_t arg_err = B_FALSE;
6134     zone_iptype_t iptype;
6135     boolean_t has_cpu_shares = B_FALSE;
6136     boolean_t has_cpu_cap = B_FALSE;
6137     struct xif *tmp;
6139     optind = 0;
6140     while ((arg = getopt(cmd->cmd_argc, cmd->cmd_argv, "?")) != EOF) {
6141         switch (arg) {
6142             case '?':
6143                 longer_usage(CMD_VERIFY);
6144                 arg_err = B_TRUE;
6145                 break;
6146             default:
6147                 short_usage(CMD_VERIFY);
6148                 arg_err = B_TRUE;
6149                 break;
6150         }
6151     }
6152     if (arg_err)
6153         return;
6155     if (optind > cmd->cmd_argc) {
6156         short_usage(CMD_VERIFY);
6157         return;
6158     }
6160     if (zone_is_read_only(CMD_VERIFY))
6161         return;
6163     assert(cmd != NULL);
6165     if (cmd->cmd_argc > 0 && (strcmp(cmd->cmd_argv[0], "save") == 0))
6166         save = B_TRUE;
6167     if (initialize(B_TRUE) != Z_OK)
6168         return;
6170     if (zonecfg_get_zonepath(handle, zonepath, sizeof (zonepath)) != Z_OK &&
6171         !global_zone) {
6172         zerr(gettext("%s not specified", pt_to_str(PT_ZONEPATH)));
6173         ret_val = Z_REQD_RESOURCE_MISSING;
6174         saw_error = B_TRUE;
6175     }
6176     if (strlen(zonepath) == 0 && !global_zone) {
6177         zerr(gettext("%s cannot be empty.", pt_to_str(PT_ZONEPATH)));
6178         ret_val = Z_REQD_RESOURCE_MISSING;
6179         saw_error = B_TRUE;
6180     }
6182     if ((err = zonecfg_get_brand(handle, brand, sizeof (brand))) != Z_OK) {
6183         zone_perror(zone, err, B_TRUE);
6184         return;
6185     }
6186     if ((err = brand_verify(handle)) != Z_OK) {
6187         zone_perror(zone, err, B_TRUE);
6188         return;
6189     }
6191     if (zonecfg_get_iptype(handle, &iptype) != Z_OK) {
6192         zerr("%s %s", gettext("cannot get"), pt_to_str(PT_IPTYPE));

```

```

6193         ret_val = Z_REQD_RESOURCE_MISSING;
6194         saw_error = B_TRUE;
6195     }
6197     if ((privs = priv_allocset()) == NULL) {
6198         zerr(gettext("%s: priv_allocset failed"), zone);
6199         return;
6200     }
6201     if (zonecfg_get_privset(handle, privs, &privname) != Z_OK) {
6202         zerr(gettext("%s: invalid privilege: %s"), zone, privname);
6203         priv_freerset(privs);
6204         free(privname);
6205         return;
6206     }
6207     priv_freerset(privs);
6209     if (zonecfg_get_hostid(handle, hostidp,
6210         sizeof(hostidp)) == Z_INVALID_PROPERTY) {
6211         zerr(gettext("%s: invalid hostid: %s"),
6212             zone, hostidp);
6213         return;
6214     }
6216     if (zonecfg_get_fs_allowed(handle, fsallowedp,
6217         sizeof(fsallowedp)) == Z_INVALID_PROPERTY) {
6218         zerr(gettext("%s: invalid fs-allowed: %s"),
6219             zone, fsallowedp);
6220         return;
6221     }
6223     if ((err = zonecfg_setfsent(handle)) != Z_OK) {
6224         zone_perror(zone, err, B_TRUE);
6225         return;
6226     }
6227     while (zonecfg_getfsent(handle, &fstab) == Z_OK) {
6228         check_reqd_prop(fstab.zone_fs_dir, RT_FS, PT_DIR, &ret_val);
6229         check_reqd_prop(fstab.zone_fs_special, RT_FS, PT_SPECIAL,
6230             &ret_val);
6231         check_reqd_prop(fstab.zone_fs_type, RT_FS, PT_TYPE, &ret_val);
6233         zonecfg_free_fs_option_list(fstab.zone_fs_options);
6234     }
6235     (void) zonecfg_endfsent(handle);
6237     if ((err = zonecfg_setnwifent(handle)) != Z_OK) {
6238         zone_perror(zone, err, B_TRUE);
6239         return;
6240     }
6241     while (zonecfg_getnwifent(handle, &nwiftab) == Z_OK) {
6242         /*
6243          * physical is required in all cases.
6244          * A shared IP requires an address,
6245          * and may include a default router, while
6246          * an exclusive IP must have neither an address
6247          * nor a default router.
6248          * The physical interface name must be valid in all cases.
6249          */
6250         check_reqd_prop(nwiftab.zone_nwif_physical, RT_NET,
6251             PT_PHYSICAL, &ret_val);
6252         if (validate_net_physical_syntax(nwiftab.zone_nwif_physical) !=
6253             Z_OK) {
6254             saw_error = B_TRUE;
6255             if (ret_val == Z_OK)
6256                 ret_val = Z_INVALID;
6257         }

```

```

6259         switch (iptype) {
6260             case ZS_SHARED:
6261                 check_reqd_prop(nwiftab.zone_nwif_address, RT_NET,
6262                     PT_ADDRESS, &ret_val);
6263                 if (strlen(nwiftab.zone_nwif_allowed_address) > 0) {
6264                     zerr(gettext("%s: %s cannot be specified "
6265                         "for a shared IP type"),
6266                         rt_to_str(RT_NET),
6267                         pt_to_str(PT_ALLOWED_ADDRESS));
6268                     saw_error = B_TRUE;
6269                     if (ret_val == Z_OK)
6270                         ret_val = Z_INVALID;
6271                 }
6272                 break;
6273             case ZS_EXCLUSIVE:
6274                 if (strlen(nwiftab.zone_nwif_address) > 0) {
6275                     zerr(gettext("%s: %s cannot be specified "
6276                         "for an exclusive IP type"),
6277                         rt_to_str(RT_NET), pt_to_str(PT_ADDRESS));
6278                     saw_error = B_TRUE;
6279                     if (ret_val == Z_OK)
6280                         ret_val = Z_INVALID;
6281                 } else {
6282                     if (!add_nwif(&nwiftab)) {
6283                         saw_error = B_TRUE;
6284                         if (ret_val == Z_OK)
6285                             ret_val = Z_INVALID;
6286                     }
6287                 }
6288                 break;
6289             }
6290         }
6291         for (tmp = xif; tmp != NULL; tmp = tmp->xif_next) {
6292             if (!tmp->xif_has_address && tmp->xif_has_defrouter) {
6293                 zerr(gettext("%s: %s for %s cannot be specified "
6294                     "without %s for an exclusive IP type"),
6295                     rt_to_str(RT_NET), pt_to_str(PT_DEFROUTER),
6296                     tmp->xif_name, pt_to_str(PT_ALLOWED_ADDRESS));
6297                 saw_error = B_TRUE;
6298                 ret_val = Z_INVALID;
6299             }
6300         }
6301         free(xif);
6302         xif = NULL;
6303         (void) zonecfg_endnwifent(handle);
6305     if ((err = zonecfg_setrctlent(handle)) != Z_OK) {
6306         zone_perror(zone, err, B_TRUE);
6307         return;
6308     }
6309     while (zonecfg_getrctlent(handle, &rctltab) == Z_OK) {
6310         check_reqd_prop(rctltab.zone_rctl_name, RT_RCTL, PT_NAME,
6311             &ret_val);
6313         if (strcmp(rctltab.zone_rctl_name, "zone.cpu-shares") == 0)
6314             has_cpu_shares = B_TRUE;
6316         if (strcmp(rctltab.zone_rctl_name, "zone.cpu-cap") == 0)
6317             has_cpu_cap = B_TRUE;
6319         if (rctltab.zone_rctl_valptr == NULL) {
6320             zerr(gettext("%s: no %s specified"),
6321                 rt_to_str(RT_RCTL), pt_to_str(PT_VALUE));
6322             saw_error = B_TRUE;
6323             if (ret_val == Z_OK)
6324                 ret_val = Z_REQD_PROPERTY_MISSING;

```

```

6325     } else {
6326         zonecfg_free_rctl_value_list(rctltab.zone_rctl_valptr);
6327     }
6328 }
6329 (void) zonecfg_endrctlent(handle);

6331 if ((pset_res = zonecfg_lookup_pset(handle, &psettab)) == Z_OK &&
6332     has_cpu_shares) {
6333     zerr(gettext("%s zone.cpu-shares and %s are incompatible."),
6334         rt_to_str(RT_RCTL), rt_to_str(RT_DCPU));
6335     saw_error = B_TRUE;
6336     if (ret_val == Z_OK)
6337         ret_val = Z_INCOMPATIBLE;
6338 }

6340 if (has_cpu_shares && zonecfg_get_sched_class(handle, sched,
6341     sizeof(sched)) == Z_OK && strlen(sched) > 0 &&
6342     strcmp(sched, "FSS" != 0) {
6343     zerr(gettext("WARNING: %s zone.cpu-shares and %s=%s are "
6344         "incompatible"),
6345         rt_to_str(RT_RCTL), rt_to_str(RT_SCHEM), sched);
6346     saw_error = B_TRUE;
6347     if (ret_val == Z_OK)
6348         ret_val = Z_INCOMPATIBLE;
6349 }

6351 if (pset_res == Z_OK && has_cpu_cap) {
6352     zerr(gettext("%s zone.cpu-cap and the %s are incompatible."),
6353         rt_to_str(RT_RCTL), rt_to_str(RT_DCPU));
6354     saw_error = B_TRUE;
6355     if (ret_val == Z_OK)
6356         ret_val = Z_INCOMPATIBLE;
6357 }

6359 if ((err = zonecfg_setattrent(handle)) != Z_OK) {
6360     zone_perror(zone, err, B_TRUE);
6361     return;
6362 }
6363 while (zonecfg_getattrent(handle, &attrtab) == Z_OK) {
6364     check_reqd_prop(attrtab.zone_attr_name, RT_ATTR, PT_NAME,
6365         &ret_val);
6366     check_reqd_prop(attrtab.zone_attr_type, RT_ATTR, PT_TYPE,
6367         &ret_val);
6368     check_reqd_prop(attrtab.zone_attr_value, RT_ATTR, PT_VALUE,
6369         &ret_val);
6370 }
6371 (void) zonecfg_endattrent(handle);

6373 if ((err = zonecfg_setdsent(handle)) != Z_OK) {
6374     zone_perror(zone, err, B_TRUE);
6375     return;
6376 }
6377 while (zonecfg_getdsent(handle, &dstab) == Z_OK) {
6378     if (strlen(dstab.zone_dataset_name) == 0) {
6379         zerr("%s: %s %s", rt_to_str(RT_DATASET),
6380             pt_to_str(PT_NAME), gettext("not specified"));
6381         saw_error = B_TRUE;
6382         if (ret_val == Z_OK)
6383             ret_val = Z_REQD_PROPERTY_MISSING;
6384     } else if (!zfs_name_valid(dstab.zone_dataset_name,
6385         ZFS_TYPE_FILESYSTEM)) {
6386         zerr("%s: %s %s", rt_to_str(RT_DATASET),
6387             pt_to_str(PT_NAME), gettext("invalid"));
6388         saw_error = B_TRUE;
6389         if (ret_val == Z_OK)
6390             ret_val = Z_BAD_PROPERTY;

```

```

6391     }
6393 }
6394 (void) zonecfg_enddsent(handle);

6396 if ((err = zonecfg_setadminent(handle)) != Z_OK) {
6397     zone_perror(zone, err, B_TRUE);
6398     return;
6399 }
6400 while (zonecfg_getadminent(handle, &admintab) == Z_OK) {
6401     check_reqd_prop(admintab.zone_admin_user, RT_ADMIN,
6402         PT_USER, &ret_val);
6403     check_reqd_prop(admintab.zone_admin_auths, RT_ADMIN,
6404         PT_AUTHS, &ret_val);
6405     if ((ret_val == Z_OK) && (getpwnam(admintab.zone_admin_user)
6406         == NULL)) {
6407         zerr(gettext("%s %s is not a valid username"),
6408             pt_to_str(PT_USER),
6409             admintab.zone_admin_user);
6410         ret_val = Z_BAD_PROPERTY;
6411     }
6412     if ((ret_val == Z_OK) && (!zonecfg_valid_auths(
6413         admintab.zone_admin_auths, zone))) {
6414         ret_val = Z_BAD_PROPERTY;
6415     }
6416 }
6417 (void) zonecfg_endadminent(handle);

6419 if (zonecfg_getsecflagsent(handle, &secflagstab) == Z_OK) {
6420     /*
6421     * No properties are required, but any specified should be
6422     * valid
6423     */
6424     if (verify_secflags(&secflagstab) != B_TRUE) {
6425         /* Error is reported from verify_secflags */
6426         ret_val = Z_BAD_PROPERTY;
6427     }
6428 }

6430 #endif /* ! codereview */
6431 if (!global_scope) {
6432     zerr(gettext("resource specification incomplete"));
6433     saw_error = B_TRUE;
6434     if (ret_val == Z_OK)
6435         ret_val = Z_INSUFFICIENT_SPEC;
6436 }

6438 if (save) {
6439     if (ret_val == Z_OK) {
6440         if ((ret_val = zonecfg_save(handle)) == Z_OK) {
6441             need_to_commit = B_FALSE;
6442             (void) strlcpy(revert_zone, zone,
6443                 sizeof(revert_zone));
6444         } else {
6445             zerr(gettext("Zone %s failed to verify"), zone);
6446         }
6447     }
6448 }
6449 if (ret_val != Z_OK)
6450     zone_perror(zone, ret_val, B_TRUE);
6451 }

6453 void
6454 cancel_func(cmd_t *cmd)
6455 {
6456     int arg;

```

```

6457     boolean_t arg_err = B_FALSE;
6459     assert(cmd != NULL);

6461     optind = 0;
6462     while ((arg = getopt(cmd->cmd_argc, cmd->cmd_argv, "?")) != EOF) {
6463         switch (arg) {
6464             case '?':
6465                 longer_usage(CMD_CANCEL);
6466                 arg_err = B_TRUE;
6467                 break;
6468             default:
6469                 short_usage(CMD_CANCEL);
6470                 arg_err = B_TRUE;
6471                 break;
6472         }
6473     }
6474     if (arg_err)
6475         return;

6477     if (optind != cmd->cmd_argc) {
6478         short_usage(CMD_CANCEL);
6479         return;
6480     }

6482     if (global_scope)
6483         scope_usage(CMD_CANCEL);
6484     global_scope = B_TRUE;
6485     zoncfg_free_fs_option_list(in_progress_fstab.zone_fs_options);
6486     bzero(&in_progress_fstab, sizeof (in_progress_fstab));
6487     bzero(&in_progress_nwifstab, sizeof (in_progress_nwifstab));
6488     bzero(&in_progress_devtab, sizeof (in_progress_devtab));
6489     zoncfg_free_rctl_value_list(in_progress_rctltab.zone_rctl_valptr);
6490     bzero(&in_progress_rctltab, sizeof (in_progress_rctltab));
6491     bzero(&in_progress_attrtab, sizeof (in_progress_attrtab));
6492     bzero(&in_progress_dstab, sizeof (in_progress_dstab));
6493 }

6495 static int
6496 validate_attr_name(char *name)
6497 {
6498     int i;

6500     if (!isalnum(name[0])) {
6501         zerr(gettext("Invalid %s %s: must start with an alpha-
6502             \"numeric character.\""), rt_to_str(RT_ATTR),
6503             pt_to_str(PT_NAME), name);
6504         return (Z_INVALID);
6505     }
6506     for (i = 1; name[i]; i++)
6507         if (!isalnum(name[i]) && name[i] != '-' && name[i] != '.') {
6508             zerr(gettext("Invalid %s %s %s: can only contain \"
6509                 alpha-numeric characters, plus '-' and '.'.\""),
6510                 rt_to_str(RT_ATTR), pt_to_str(PT_NAME), name);
6511             return (Z_INVALID);
6512         }
6513     return (Z_OK);
6514 }

6516 static int
6517 validate_attr_type_val(struct zone_attrtab *attrtab)
6518 {
6519     boolean_t boolval;
6520     int64_t intval;
6521     char strval[MAXNAMELEN];
6522     uint64_t uintval;

```

```

6524     if (strcmp(attrtab->zone_attr_type, "boolean") == 0) {
6525         if (zoncfg_get_attr_boolean(attrtab, &boolval) == Z_OK)
6526             return (Z_OK);
6527         zerr(gettext("invalid %s value for %s=%s"),
6528             rt_to_str(RT_ATTR), pt_to_str(PT_TYPE), "boolean");
6529         return (Z_ERR);
6530     }

6532     if (strcmp(attrtab->zone_attr_type, "int") == 0) {
6533         if (zoncfg_get_attr_int(attrtab, &intval) == Z_OK)
6534             return (Z_OK);
6535         zerr(gettext("invalid %s value for %s=%s"),
6536             rt_to_str(RT_ATTR), pt_to_str(PT_TYPE), "int");
6537         return (Z_ERR);
6538     }

6540     if (strcmp(attrtab->zone_attr_type, "string") == 0) {
6541         if (zoncfg_get_attr_string(attrtab, strval,
6542             sizeof (strval)) == Z_OK)
6543             return (Z_OK);
6544         zerr(gettext("invalid %s value for %s=%s"),
6545             rt_to_str(RT_ATTR), pt_to_str(PT_TYPE), "string");
6546         return (Z_ERR);
6547     }

6549     if (strcmp(attrtab->zone_attr_type, "uint") == 0) {
6550         if (zoncfg_get_attr_uint(attrtab, &uintval) == Z_OK)
6551             return (Z_OK);
6552         zerr(gettext("invalid %s value for %s=%s"),
6553             rt_to_str(RT_ATTR), pt_to_str(PT_TYPE), "uint");
6554         return (Z_ERR);
6555     }

6557     zerr(gettext("invalid %s %s '%s'"), rt_to_str(RT_ATTR),
6558         pt_to_str(PT_TYPE), attrtab->zone_attr_type);
6559     return (Z_ERR);
6560 }

6562 /*
6563  * Helper function for end_func-- checks the existence of a given property
6564  * and emits a message if not specified.
6565  */
6566 static int
6567 end_check_reqd(char *attr, int pt, boolean_t *validation_failed)
6568 {
6569     if (strlen(attr) == 0) {
6570         *validation_failed = B_TRUE;
6571         zerr(gettext("%s not specified"), pt_to_str(pt));
6572         return (Z_ERR);
6573     }
6574     return (Z_OK);
6575 }

6577 static void
6578 net_exists_error(struct zone_nwifstab nwif)
6579 {
6580     if (strlen(nwif.zone_nwif_address) > 0) {
6581         zerr(gettext("A %s resource with the %s '%s', \"
6582             \"and %s '%s' already exists.\""),
6583             rt_to_str(RT_NET),
6584             pt_to_str(PT_PHYSICAL),
6585             nwif.zone_nwif_physical,
6586             pt_to_str(PT_ADDRESS),
6587             in_progress_nwifstab.zone_nwif_address);
6588     } else {

```

```

6589         zerr(gettext("A %s resource with the %s '%s', "
6590             "and %s '%s' already exists."),
6591             rt_to_str(RT_NET),
6592             pt_to_str(PT_PHYSICAL),
6593             nwif.zone_nwif_physical,
6594             pt_to_str(PT_ALLOWED_ADDRESS),
6595             nwif.zone_nwif_allowed_address);
6596     }
6597 }

6599 void
6600 end_func(cmd_t *cmd)
6601 {
6602     boolean_t validation_failed = B_FALSE;
6603     boolean_t arg_err = B_FALSE;
6604     struct zone_fstab tmp_fstab;
6605     struct zone_nwifstab tmp_nwifstab;
6606     struct zone_devtab tmp_devtab;
6607     struct zone_rctltab tmp_rctltab;
6608     struct zone_attrtab tmp_attrtab;
6609     struct zone_dstab tmp_dstab;
6610     struct zone_admintab tmp_admintab;
6611     int err, arg, res1, res2, res3;
6612     uint64_t swap_limit;
6613     uint64_t locked_limit;
6614     uint64_t proc_cap;

6616     assert(cmd != NULL);

6618     optind = 0;
6619     while ((arg = getopt(cmd->cmd_argc, cmd->cmd_argv, "?")) != EOF) {
6620         switch (arg) {
6621             case '?':
6622                 longer_usage(CMD_END);
6623                 arg_err = B_TRUE;
6624                 break;
6625             default:
6626                 short_usage(CMD_END);
6627                 arg_err = B_TRUE;
6628                 break;
6629         }
6630     }
6631     if (arg_err)
6632         return;

6634     if (optind != cmd->cmd_argc) {
6635         short_usage(CMD_END);
6636         return;
6637     }

6639     if (global_scope) {
6640         scope_usage(CMD_END);
6641         return;
6642     }

6644     assert(end_op == CMD_ADD || end_op == CMD_SELECT);

6646     switch (resource_scope) {
6647     case RT_FS:
6648         /* First make sure everything was filled in. */
6649         if (end_check_reqd(in_progress_fstab.zone_fs_dir,
6650             PT_DIR, &validation_failed) == Z_OK) {
6651             if (in_progress_fstab.zone_fs_dir[0] != '/') {
6652                 zerr(gettext("%s %s is not an absolute path."),
6653                     pt_to_str(PT_DIR),
6654                     in_progress_fstab.zone_fs_dir);

```

```

6655         validation_failed = B_TRUE;
6656     }
6657 }

6659     (void) end_check_reqd(in_progress_fstab.zone_fs_special,
6660         PT_SPECIAL, &validation_failed);

6662     if (in_progress_fstab.zone_fs_raw[0] != '\0' &&
6663         in_progress_fstab.zone_fs_raw[0] != '/') {
6664         zerr(gettext("%s %s is not an absolute path."),
6665             pt_to_str(PT_RAW),
6666             in_progress_fstab.zone_fs_raw);
6667         validation_failed = B_TRUE;
6668     }

6670     (void) end_check_reqd(in_progress_fstab.zone_fs_type, PT_TYPE,
6671         &validation_failed);

6673     if (validation_failed) {
6674         saw_error = B_TRUE;
6675         return;
6676     }

6678     if (end_op == CMD_ADD) {
6679         /* Make sure there isn't already one like this. */
6680         bzero(&tmp_fstab, sizeof (tmp_fstab));
6681         (void) strncpy(tmp_fstab.zone_fs_dir,
6682             in_progress_fstab.zone_fs_dir,
6683             sizeof (tmp_fstab.zone_fs_dir));
6684         err = zoncfg_lookup_filesystem(handle, &tmp_fstab);
6685         zoncfg_free_fs_option_list(tmp_fstab.zone_fs_options);
6686         if (err == Z_OK) {
6687             zerr(gettext("A %s resource "
6688                 "with the %s '%s' already exists."),
6689                 rt_to_str(RT_FS), pt_to_str(PT_DIR),
6690                 in_progress_fstab.zone_fs_dir);
6691             saw_error = B_TRUE;
6692             return;
6693         }
6694         err = zoncfg_add_filesystem(handle,
6695             &in_progress_fstab);
6696     } else {
6697         err = zoncfg_modify_filesystem(handle, &old_fstab,
6698             &in_progress_fstab);
6699     }
6700     zoncfg_free_fs_option_list(in_progress_fstab.zone_fs_options);
6701     in_progress_fstab.zone_fs_options = NULL;
6702     break;

6704     case RT_NET:
6705         /*
6706          * First make sure everything was filled in.
6707          * Since we don't know whether IP will be shared
6708          * or exclusive here, some checks are deferred until
6709          * the verify command.
6710          */
6711         (void) end_check_reqd(in_progress_nwifstab.zone_nwif_physical,
6712             PT_PHYSICAL, &validation_failed);

6714         if (validation_failed) {
6715             saw_error = B_TRUE;
6716             return;
6717         }
6718         if (end_op == CMD_ADD) {
6719             /* Make sure there isn't already one like this. */
6720             bzero(&tmp_nwifstab, sizeof (tmp_nwifstab));

```

```

6721         (void) strcpy(tmp_nwiftab.zone_nwif_physical,
6722             in_progress_nwiftab.zone_nwif_physical,
6723             sizeof (tmp_nwiftab.zone_nwif_physical));
6724         (void) strcpy(tmp_nwiftab.zone_nwif_address,
6725             in_progress_nwiftab.zone_nwif_address,
6726             sizeof (tmp_nwiftab.zone_nwif_address));
6727         (void) strcpy(tmp_nwiftab.zone_nwif_allowed_address,
6728             in_progress_nwiftab.zone_nwif_allowed_address,
6729             sizeof (tmp_nwiftab.zone_nwif_allowed_address));
6730         (void) strcpy(tmp_nwiftab.zone_nwif_defrouter,
6731             in_progress_nwiftab.zone_nwif_defrouter,
6732             sizeof (tmp_nwiftab.zone_nwif_defrouter));
6733         if (zoncfg_lookup_nwif(handle, &tmp_nwiftab) == Z_OK) {
6734             net_exists_error(in_progress_nwiftab);
6735             saw_error = B_TRUE;
6736             return;
6737         }
6738         err = zoncfg_add_nwif(handle, &in_progress_nwiftab);
6739     } else {
6740         err = zoncfg_modify_nwif(handle, &old_nwiftab,
6741             &in_progress_nwiftab);
6742     }
6743     break;

6745 case RT_DEVICE:
6746     /* First make sure everything was filled in. */
6747     (void) end_check_reqd(in_progress_devtab.zone_dev_match,
6748         PT_MATCH, &validation_failed);

6750     if (validation_failed) {
6751         saw_error = B_TRUE;
6752         return;
6753     }

6755     if (end_op == CMD_ADD) {
6756         /* Make sure there isn't already one like this. */
6757         (void) strcpy(tmp_devtab.zone_dev_match,
6758             in_progress_devtab.zone_dev_match,
6759             sizeof (tmp_devtab.zone_dev_match));
6760         if (zoncfg_lookup_dev(handle, &tmp_devtab) == Z_OK) {
6761             zerr(gettext("A %s resource with the %s '%s' "
6762                 "already exists."), rt_to_str(RT_DEVICE),
6763                 pt_to_str(PT_MATCH),
6764                 in_progress_devtab.zone_dev_match);
6765             saw_error = B_TRUE;
6766             return;
6767         }
6768         err = zoncfg_add_dev(handle, &in_progress_devtab);
6769     } else {
6770         err = zoncfg_modify_dev(handle, &old_devtab,
6771             &in_progress_devtab);
6772     }
6773     break;

6775 case RT_RCTL:
6776     /* First make sure everything was filled in. */
6777     (void) end_check_reqd(in_progress_rctltab.zone_rctl_name,
6778         PT_NAME, &validation_failed);

6780     if (in_progress_rctltab.zone_rctl_valptr == NULL) {
6781         zerr(gettext("no %s specified"), pt_to_str(PT_VALUE));
6782         validation_failed = B_TRUE;
6783     }

6785     if (validation_failed) {
6786         saw_error = B_TRUE;

```

```

6787         return;
6788     }

6790     if (end_op == CMD_ADD) {
6791         /* Make sure there isn't already one like this. */
6792         (void) strcpy(tmp_rctltab.zone_rctl_name,
6793             in_progress_rctltab.zone_rctl_name,
6794             sizeof (tmp_rctltab.zone_rctl_name));
6795         tmp_rctltab.zone_rctl_valptr = NULL;
6796         err = zoncfg_lookup_rctl(handle, &tmp_rctltab);
6797         zoncfg_free_rctl_value_list(
6798             tmp_rctltab.zone_rctl_valptr);
6799         if (err == Z_OK) {
6800             zerr(gettext("A %s resource "
6801                 "with the %s '%s' already exists."),
6802                 rt_to_str(RT_RCTL), pt_to_str(PT_NAME),
6803                 in_progress_rctltab.zone_rctl_name);
6804             saw_error = B_TRUE;
6805             return;
6806         }
6807         err = zoncfg_add_rctl(handle, &in_progress_rctltab);
6808     } else {
6809         err = zoncfg_modify_rctl(handle, &old_rctltab,
6810             &in_progress_rctltab);
6811     }
6812     if (err == Z_OK) {
6813         zoncfg_free_rctl_value_list(
6814             in_progress_rctltab.zone_rctl_valptr);
6815         in_progress_rctltab.zone_rctl_valptr = NULL;
6816     }
6817     break;

6819 case RT_ATTR:
6820     /* First make sure everything was filled in. */
6821     (void) end_check_reqd(in_progress_attrtab.zone_attr_name,
6822         PT_NAME, &validation_failed);
6823     (void) end_check_reqd(in_progress_attrtab.zone_attr_type,
6824         PT_TYPE, &validation_failed);
6825     (void) end_check_reqd(in_progress_attrtab.zone_attr_value,
6826         PT_VALUE, &validation_failed);

6828     if (validate_attr_name(in_progress_attrtab.zone_attr_name) !=
6829         Z_OK)
6830         validation_failed = B_TRUE;

6832     if (validate_attr_type_val(&in_progress_attrtab) != Z_OK)
6833         validation_failed = B_TRUE;

6835     if (validation_failed) {
6836         saw_error = B_TRUE;
6837         return;
6838     }
6839     if (end_op == CMD_ADD) {
6840         /* Make sure there isn't already one like this. */
6841         bzero(&tmp_attrtab, sizeof (tmp_attrtab));
6842         (void) strcpy(tmp_attrtab.zone_attr_name,
6843             in_progress_attrtab.zone_attr_name,
6844             sizeof (tmp_attrtab.zone_attr_name));
6845         if (zoncfg_lookup_attr(handle, &tmp_attrtab) == Z_OK) {
6846             zerr(gettext("An %s resource "
6847                 "with the %s '%s' already exists."),
6848                 rt_to_str(RT_ATTR), pt_to_str(PT_NAME),
6849                 in_progress_attrtab.zone_attr_name);
6850             saw_error = B_TRUE;
6851             return;
6852         }

```



```

6853         err = zoncfg_add_attr(handle, &in_progress_attrtab);
6854     } else {
6855         err = zoncfg_modify_attr(handle, &old_attrtab,
6856             &in_progress_attrtab);
6857     }
6858     break;
6859 case RT_DATASET:
6860     /* First make sure everything was filled in. */
6861     if (strlen(in_progress_dstab.zone_dataset_name) == 0) {
6862         zerr("%s %s", pt_to_str(PT_NAME),
6863             gettext("not specified"));
6864         saw_error = B_TRUE;
6865         validation_failed = B_TRUE;
6866     }
6867     if (validation_failed)
6868         return;
6869     if (end_op == CMD_ADD) {
6870         /* Make sure there isn't already one like this. */
6871         bzero(&tmp_dstab, sizeof(tmp_dstab));
6872         (void) strcpy(tmp_dstab.zone_dataset_name,
6873             in_progress_dstab.zone_dataset_name,
6874             sizeof(tmp_dstab.zone_dataset_name));
6875         err = zoncfg_lookup_ds(handle, &tmp_dstab);
6876         if (err == Z_OK) {
6877             zerr(gettext("A %s resource "
6878                 "with the %s '%s' already exists."),
6879                 rt_to_str(RT_DATASET), pt_to_str(PT_NAME),
6880                 in_progress_dstab.zone_dataset_name);
6881             saw_error = B_TRUE;
6882             return;
6883         }
6884         err = zoncfg_add_ds(handle, &in_progress_dstab);
6885     } else {
6886         err = zoncfg_modify_ds(handle, &old_dstab,
6887             &in_progress_dstab);
6888     }
6889     break;
6890 case RT_DCPU:
6891     /* Make sure everything was filled in. */
6892     if (end_check_reqd(in_progress_psettab.zone_ncpu_min,
6893         PT_NCPUS, &validation_failed) != Z_OK) {
6894         saw_error = B_TRUE;
6895         return;
6896     }
6897
6898     if (end_op == CMD_ADD) {
6899         err = zoncfg_add_pset(handle, &in_progress_psettab);
6900     } else {
6901         err = zoncfg_modify_pset(handle, &in_progress_psettab);
6902     }
6903     break;
6904 case RT_PCAP:
6905     /* Make sure everything was filled in. */
6906     if (zoncfg_get_aliased_rctl(handle, ALIAS_CPUCAP, &proc_cap)
6907         != Z_OK) {
6908         zerr(gettext("%s not specified"), pt_to_str(PT_NCPUS));
6909         saw_error = B_TRUE;
6910         validation_failed = B_TRUE;
6911         return;
6912     }
6913     err = Z_OK;
6914     break;
6915 case RT_MCAP:
6916     /* Make sure everything was filled in. */
6917     res1 = strlen(in_progress_mcaptab.zone_physmem_cap) == 0 ?
6918         Z_ERR : Z_OK;

```

```

6919         res2 = zoncfg_get_aliased_rctl(handle, ALIAS_MAXSWAP,
6920             &swap_limit);
6921         res3 = zoncfg_get_aliased_rctl(handle, ALIAS_MAXLOCKEDMEM,
6922             &locked_limit);
6923
6924         if (res1 != Z_OK && res2 != Z_OK && res3 != Z_OK) {
6925             zerr(gettext("No property was specified. One of %s, "
6926                 "%s or %s is required."), pt_to_str(PT_PHYSICAL),
6927                 pt_to_str(PT_SWAP), pt_to_str(PT_LOCKED));
6928             saw_error = B_TRUE;
6929             return;
6930         }
6931
6932         /* if phys & locked are both set, verify locked <= phys */
6933         if (res1 == Z_OK && res3 == Z_OK) {
6934             uint64_t phys_limit;
6935             char *endp;
6936
6937             phys_limit = strtoull(
6938                 in_progress_mcaptab.zone_physmem_cap, &endp, 10);
6939             if (phys_limit < locked_limit) {
6940                 zerr(gettext("The %s cap must be less than or "
6941                     "equal to the %s cap."),
6942                     pt_to_str(PT_LOCKED),
6943                     pt_to_str(PT_PHYSICAL));
6944                 saw_error = B_TRUE;
6945                 return;
6946             }
6947         }
6948
6949         err = Z_OK;
6950         if (res1 == Z_OK) {
6951             /*
6952              * We could be ending from either an add operation
6953              * or a select operation. Since all of the properties
6954              * within this resource are optional, we always use
6955              * modify on the mcap entry. zoncfg_modify_mcap()
6956              * will handle both adding and modifying a memory cap.
6957              */
6958             err = zoncfg_modify_mcap(handle, &in_progress_mcaptab);
6959         } else if (end_op == CMD_SELECT) {
6960             /*
6961              * If we're ending from a select and the physical
6962              * memory cap is empty then the user could have cleared
6963              * the physical cap value, so try to delete the entry.
6964              */
6965             (void) zoncfg_delete_mcap(handle);
6966         }
6967         break;
6968 case RT_ADMIN:
6969     /* First make sure everything was filled in. */
6970     if (end_check_reqd(in_progress_admintab.zone_admin_user,
6971         PT_USER, &validation_failed) == Z_OK) {
6972         if (getpwnam(in_progress_admintab.zone_admin_user)
6973             == NULL) {
6974             zerr(gettext("%s %s is not a valid username"),
6975                 pt_to_str(PT_USER),
6976                 in_progress_admintab.zone_admin_user);
6977             validation_failed = B_TRUE;
6978         }
6979     }
6980
6981     if (end_check_reqd(in_progress_admintab.zone_admin_auths,
6982         PT_AUTHS, &validation_failed) == Z_OK) {
6983         if (!zoncfg_valid_auths(
6984             in_progress_admintab.zone_admin_auths,

```

```

6985     zone) {
6986         validation_failed = B_TRUE;
6987     }
6988 }
6990 if (validation_failed) {
6991     saw_error = B_TRUE;
6992     return;
6993 }
6995 if (end_op == CMD_ADD) {
6996     /* Make sure there isn't already one like this. */
6997     bzero(&tmp_admintab, sizeof (tmp_admintab));
6998     (void) strncpy(tmp_admintab.zone_admin_user,
6999                 in_progress_admintab.zone_admin_user,
7000                 sizeof (tmp_admintab.zone_admin_user));
7001     err = zoncfg_lookup_admin(
7002         handle, &tmp_admintab);
7003     if (err == Z_OK) {
7004         zerr(gettext("A %s resource "
7005                    "with the %s '%s' already exists."),
7006             rt_to_str(RT_ADMIN),
7007             pt_to_str(PT_USER),
7008             in_progress_admintab.zone_admin_user);
7009         saw_error = B_TRUE;
7010         return;
7011     }
7012     err = zoncfg_add_admin(handle,
7013                          &in_progress_admintab, zone);
7014 } else {
7015     err = zoncfg_modify_admin(handle,
7016                             &old_admintab, &in_progress_admintab,
7017                             zone);
7018 }
7019 break;
7020 case RT_SECFLAGS:
7021     if (verify_secflags(&in_progress_secflagstab) != B_TRUE) {
7022         saw_error = B_TRUE;
7023         return;
7024     }
7026     if (end_op == CMD_ADD) {
7027         err = zoncfg_add_secflags(handle,
7028                                 &in_progress_secflagstab);
7029     } else {
7030         err = zoncfg_modify_secflags(handle,
7031                                     &old_secflagstab, &in_progress_secflagstab);
7032     }
7033     break;
7034 #endif /* ! codereview */
7035 default:
7036     zone_perror(rt_to_str(resource_scope), Z_NO_RESOURCE_TYPE,
7037                B_TRUE);
7038     saw_error = B_TRUE;
7039     return;
7040 }
7042 if (err != Z_OK) {
7043     zone_perror(zone, err, B_TRUE);
7044 } else {
7045     need_to_commit = B_TRUE;
7046     global_scope = B_TRUE;
7047     end_op = -1;
7048 }
7049 }

```

```

7051 void
7052 commit_func(cmd_t *cmd)
7053 {
7054     int arg;
7055     boolean_t arg_err = B_FALSE;
7057     optind = 0;
7058     while ((arg = getopt(cmd->cmd_argc, cmd->cmd_argv, "?")) != EOF) {
7059         switch (arg) {
7060             case '?':
7061                 longer_usage(CMD_COMMIT);
7062                 arg_err = B_TRUE;
7063                 break;
7064             default:
7065                 short_usage(CMD_COMMIT);
7066                 arg_err = B_TRUE;
7067                 break;
7068         }
7069     }
7070     if (arg_err)
7071         return;
7073     if (optind != cmd->cmd_argc) {
7074         short_usage(CMD_COMMIT);
7075         return;
7076     }
7078     if (zone_is_read_only(CMD_COMMIT))
7079         return;
7081     assert(cmd != NULL);
7083     cmd->cmd_argc = 1;
7084     /*
7085      * cmd arg normally comes from a strdup() in the lexer, and the
7086      * whole cmd structure and its (char *) attributes are freed at
7087      * the completion of each command, so the strdup() below is needed
7088      * to match this and prevent a core dump from trying to free()
7089      * something that can't be.
7090      */
7091     if ((cmd->cmd_argv[0] = strdup("save")) == NULL) {
7092         zone_perror(zone, Z_NOMEM, B_TRUE);
7093         exit(Z_ERR);
7094     }
7095     cmd->cmd_argv[1] = NULL;
7096     verify_func(cmd);
7097 }
7099 void
7100 revert_func(cmd_t *cmd)
7101 {
7102     char line[128]; /* enough to ask a question */
7103     boolean_t force = B_FALSE;
7104     boolean_t arg_err = B_FALSE;
7105     int err, arg, answer;
7107     optind = 0;
7108     while ((arg = getopt(cmd->cmd_argc, cmd->cmd_argv, "?F")) != EOF) {
7109         switch (arg) {
7110             case '?':
7111                 longer_usage(CMD_REVERT);
7112                 arg_err = B_TRUE;
7113                 break;
7114             case 'F':
7115                 force = B_TRUE;
7116                 break;

```

```

7117         default:
7118             short_usage(CMD_REVERT);
7119             arg_err = B_TRUE;
7120             break;
7121     }
7122 }
7123 if (arg_err)
7124     return;
7125
7126 if (optind != cmd->cmd_argc) {
7127     short_usage(CMD_REVERT);
7128     return;
7129 }
7130
7131 if (zone_is_read_only(CMD_REVERT))
7132     return;
7133
7134 if (!global_scope) {
7135     zerr(gettext("You can only use %s in the global scope.\nUse"
7136                "'%s' to cancel changes to a resource specification."),
7137          cmd_to_str(CMD_REVERT), cmd_to_str(CMD_CANCEL));
7138     saw_error = B_TRUE;
7139     return;
7140 }
7141
7142 if (zonecfg_check_handle(handle) != Z_OK) {
7143     zerr(gettext("No changes to revert."));
7144     saw_error = B_TRUE;
7145     return;
7146 }
7147
7148 if (!force) {
7149     (void) snprintf(line, sizeof (line),
7150                    gettext("Are you sure you want to revert"));
7151     if ((answer = ask_yesno(B_FALSE, line)) == -1) {
7152         zerr(gettext("Input not from terminal and -F not "
7153                    "specified:\n%s command ignored, exiting."),
7154             cmd_to_str(CMD_REVERT));
7155         exit(Z_ERR);
7156     }
7157     if (answer != 1)
7158         return;
7159 }
7160
7161 /*
7162  * Reset any pending admins that were
7163  * removed from the previous zone
7164  */
7165 zonecfg_remove_userauths(handle, "", zone, B_FALSE);
7166
7167 /*
7168  * Time for a new handle: finish the old one off first
7169  * then get a new one properly to avoid leaks.
7170  */
7171 zonecfg_fini_handle(handle);
7172 if ((handle = zonecfg_init_handle()) == NULL) {
7173     zone_perror(execname, Z_NOMEM, B_TRUE);
7174     exit(Z_ERR);
7175 }
7176
7177 if ((err = zonecfg_get_handle(revert_zone, handle)) != Z_OK) {
7178     saw_error = B_TRUE;
7179     got_handle = B_FALSE;
7180     if (err == Z_NO_ZONE)
7181         zerr(gettext("%s: no such saved zone to revert to."),
7182             revert_zone);

```

```

7183     else
7184         zone_perror(zone, err, B_TRUE);
7185 }
7186 (void) strncpy(zone, revert_zone, sizeof (zone));
7187 }
7188
7189 void
7190 help_func(cmd_t *cmd)
7191 {
7192     int i;
7193
7194     assert(cmd != NULL);
7195
7196     if (cmd->cmd_argc == 0) {
7197         usage(B_TRUE, global_scope ? HELP_SUBCMDS : HELP_RES_SCOPE);
7198         return;
7199     }
7200     if (strcmp(cmd->cmd_argv[0], "usage") == 0) {
7201         usage(B_TRUE, HELP_USAGE);
7202         return;
7203     }
7204     if (strcmp(cmd->cmd_argv[0], "commands") == 0) {
7205         usage(B_TRUE, HELP_SUBCMDS);
7206         return;
7207     }
7208     if (strcmp(cmd->cmd_argv[0], "syntax") == 0) {
7209         usage(B_TRUE, HELP_SYNTAX | HELP_RES_PROPS);
7210         return;
7211     }
7212     if (strcmp(cmd->cmd_argv[0], "-?") == 0) {
7213         longer_usage(CMD_HELP);
7214         return;
7215     }
7216
7217     for (i = 0; i <= CMD_MAX; i++) {
7218         if (strcmp(cmd->cmd_argv[0], cmd_to_str(i)) == 0) {
7219             longer_usage(i);
7220             return;
7221         }
7222     }
7223     /* We do not use zerr() here because we do not want its extra \n. */
7224     (void) fprintf(stderr, gettext("Unknown help subject %s. "),
7225                  cmd->cmd_argv[0]);
7226     usage(B_FALSE, HELP_META);
7227 }
7228
7229 static int
7230 string_to_yyin(char *string)
7231 {
7232     if ((yyin = tmpfile()) == NULL) {
7233         zone_perror(execname, Z_TEMP_FILE, B_TRUE);
7234         return (Z_ERR);
7235     }
7236     if (fwrite(string, strlen(string), 1, yyin) != 1) {
7237         zone_perror(execname, Z_TEMP_FILE, B_TRUE);
7238         return (Z_ERR);
7239     }
7240     if (fseek(yyin, 0, SEEK_SET) != 0) {
7241         zone_perror(execname, Z_TEMP_FILE, B_TRUE);
7242         return (Z_ERR);
7243     }
7244     return (Z_OK);
7245 }
7246
7247 /* This is the back-end helper function for read_input() below. */

```

```

7249 static int
7250 cleanup()
7251 {
7252     int answer;
7253     cmd_t *cmd;

7255     if (!interactive_mode && !cmd_file_mode) {
7256         /*
7257          * If we're not in interactive mode, and we're not in command
7258          * file mode, then we must be in commands-from-the-command-line
7259          * mode. As such, we can't loop back and ask for more input.
7260          * It was OK to prompt for such things as whether or not to
7261          * really delete a zone in the command handler called from
7262          * yyparse() above, but "really quit?" makes no sense in this
7263          * context. So disable prompting.
7264          */
7265         ok_to_prompt = B_FALSE;
7266     }
7267     if (!global_scope) {
7268         if (!time_to_exit) {
7269             /*
7270              * Just print a simple error message in the -1 case,
7271              * since exit_func() already handles that case, and
7272              * EOF means we are finished anyway.
7273              */
7274             answer = ask_yn(B_FALSE,
7275                 gettext("Resource incomplete; really quit"));
7276             if (answer == -1) {
7277                 zerr(gettext("Resource incomplete.));
7278                 return (Z_ERR);
7279             }
7280             if (answer != 1) {
7281                 yyin = stdin;
7282                 return (Z_REPEAT);
7283             }
7284         } else {
7285             saw_error = B_TRUE;
7286         }
7287     }
7288     /*
7289     * Make sure we tried something and that the handle checks
7290     * out, or we would get a false error trying to commit.
7291     */
7292     if (need_to_commit && zoncfg_check_handle(handle) == Z_OK) {
7293         if ((cmd = alloc_cmd()) == NULL) {
7294             zone_perror(zone, Z_NOMEM, B_TRUE);
7295             return (Z_ERR);
7296         }
7297         cmd->cmd_argc = 0;
7298         cmd->cmd_argv[0] = NULL;
7299         commit_func(cmd);
7300         free_cmd(cmd);
7301         /*
7302          * need_to_commit will get set back to FALSE if the
7303          * configuration is saved successfully.
7304          */
7305         if (need_to_commit) {
7306             if (force_exit) {
7307                 zerr(gettext("Configuration not saved.));
7308                 return (Z_ERR);
7309             }
7310             answer = ask_yn(B_FALSE,
7311                 gettext("Configuration not saved; really quit"));
7312             if (answer == -1) {
7313                 zerr(gettext("Configuration not saved.));
7314                 return (Z_ERR);

```

```

7315     }
7316     if (answer != 1) {
7317         time_to_exit = B_FALSE;
7318         yyin = stdin;
7319         return (Z_REPEAT);
7320     }
7321 }
7322 }
7323 return ((need_to_commit || saw_error) ? Z_ERR : Z_OK);
7324 }

7326 /*
7327 * read_input() is the driver of this program. It is a wrapper around
7328 * yyparse(), printing appropriate prompts when needed, checking for
7329 * exit conditions and reacting appropriately [the latter in its cleanup()
7330 * helper function].
7331 *
7332 * Like most zoncfg functions, it returns Z_OK or Z_ERR, *or* Z_REPEAT
7333 * so do_interactive() knows that we are not really done (i.e, we asked
7334 * the user if we should really quit and the user said no).
7335 */
7336 static int
7337 read_input()
7338 {
7339     boolean_t yyin_is_a_tty = isatty(fileno(yyin));
7340     /*
7341      * The prompt is "e:z> " or "e:z:r> " where e is execname, z is zone
7342      * and r is resource_scope: 5 is for the two ":s + "> " + terminator.
7343      */
7344     char prompt[MAXPATHLEN + ZONENAME_MAX + MAX_RT_STRLEN + 5], *line;

7346     /* yyin should have been set to the appropriate (FILE *) if not stdin */
7347     newline_terminated = B_TRUE;
7348     for (;;) {
7349         if (yyin_is_a_tty) {
7350             if (newline_terminated) {
7351                 if (global_scope)
7352                     (void) snprintf(prompt, sizeof (prompt),
7353                         "%s:%s> ", execname, zone);
7354                 else
7355                     (void) snprintf(prompt, sizeof (prompt),
7356                         "%s:%s:%s> ", execname, zone,
7357                         rt_to_str(resource_scope));
7358             }
7359             /*
7360              * If the user hits ^C then we want to catch it and
7361              * start over. If the user hits EOF then we want to
7362              * bail out.
7363              */
7364             line = gl_get_line(gl, prompt, NULL, -1);
7365             if (gl_return_status(gl) == GLR_SIGNAL) {
7366                 gl_abandon_line(gl);
7367                 continue;
7368             }
7369             if (line == NULL)
7370                 break;
7371             (void) string_to_yyin(line);
7372             while (!feof(yyin))
7373                 yyparse();
7374         } else {
7375             yyparse();
7376         }
7377         /* Bail out on an error in command file mode. */
7378         if (saw_error && cmd_file_mode && !interactive_mode)
7379             time_to_exit = B_TRUE;
7380         if (time_to_exit || (!yyin_is_a_tty && feof(yyin)))

```

```

7381         break;
7382     }
7383     return (cleanup());
7384 }

7386 /*
7387  * This function is used in the zonecfg-interactive-mode scenario: it just
7388  * calls read_input() until we are done.
7389  */

7391 static int
7392 do_interactive(void)
7393 {
7394     int err;

7396     interactive_mode = B_TRUE;
7397     if (!read_only_mode) {
7398         /*
7399          * Try to set things up proactively in interactive mode, so
7400          * that if the zone in question does not exist yet, we can
7401          * provide the user with a clue.
7402          */
7403         (void) initialize(B_FALSE);
7404     }
7405     do {
7406         err = read_input();
7407     } while (err == Z_REPEAT);
7408     return (err);
7409 }

7411 /*
7412  * cmd_file is slightly more complicated, as it has to open the command file
7413  * and set yyin appropriately. Once that is done, though, it just calls
7414  * read_input(), and only once, since prompting is not possible.
7415  */

7417 static int
7418 cmd_file(char *file)
7419 {
7420     FILE *infile;
7421     int err;
7422     struct stat statbuf;
7423     boolean_t using_real_file = (strcmp(file, "-") != 0);

7425     if (using_real_file) {
7426         /*
7427          * zerr() prints a line number in cmd_file_mode, which we do
7428          * not want here, so temporarily unset it.
7429          */
7430         cmd_file_mode = B_FALSE;
7431         if ((infile = fopen(file, "r")) == NULL) {
7432             zerr(gettext("could not open file %s: %s"),
7433                 file, strerror(errno));
7434             return (Z_ERR);
7435         }
7436         if ((err = fstat(fileno(infile), &statbuf)) != 0) {
7437             zerr(gettext("could not stat file %s: %s"),
7438                 file, strerror(errno));
7439             err = Z_ERR;
7440             goto done;
7441         }
7442         if (!S_ISREG(statbuf.st_mode)) {
7443             zerr(gettext("%s is not a regular file."), file);
7444             err = Z_ERR;
7445             goto done;
7446         }

```

```

7447         yyin = infile;
7448         cmd_file_mode = B_TRUE;
7449         ok_to_prompt = B_FALSE;
7450     } else {
7451         /*
7452          * "-f -" is essentially the same as interactive mode,
7453          * so treat it that way.
7454          */
7455         interactive_mode = B_TRUE;
7456     }
7457     /* Z_REPEAT is for interactive mode; treat it like Z_ERR here. */
7458     if ((err = read_input()) == Z_REPEAT)
7459         err = Z_ERR;
7460 done:
7461     if (using_real_file)
7462         (void) fclose(infile);
7463     return (err);
7464 }

7466 /*
7467  * Since yacc is based on reading from a (FILE *) whereas what we get from
7468  * the command line is in argv format, we need to convert when the user
7469  * gives us commands directly from the command line. That is done here by
7470  * concatenating the argv list into a space-separated string, writing it
7471  * to a temp file, and rewinding the file so yyin can be set to it. Then
7472  * we call read_input(), and only once, since prompting about whether to
7473  * continue or quit would make no sense in this context.
7474  */

7476 static int
7477 one_command_at_a_time(int argc, char *argv[])
7478 {
7479     char *command;
7480     size_t len = 2; /* terminal \n\0 */
7481     int i, err;

7483     for (i = 0; i < argc; i++)
7484         len += strlen(argv[i]) + 1;
7485     if ((command = malloc(len)) == NULL) {
7486         zone_perror(execname, Z_NOMEM, B_TRUE);
7487         return (Z_ERR);
7488     }
7489     (void) strcpy(command, argv[0], len);
7490     for (i = 1; i < argc; i++) {
7491         (void) strcat(command, " ", len);
7492         (void) strcat(command, argv[i], len);
7493     }
7494     (void) strcat(command, "\n", len);
7495     err = string_to_yyin(command);
7496     free(command);
7497     if (err != Z_OK)
7498         return (err);
7499     while (!feof(yyin))
7500         yyparse();
7501     return (cleanup());
7502 }

7504 static char *
7505 get_execbasename(char *execfullname)
7506 {
7507     char *last_slash, *execbasename;

7509     /* guard against '/' at end of command invocation */
7510     for (;;) {
7511         last_slash = strrchr(execfullname, '/');
7512         if (last_slash == NULL) {

```

```

7513         execbasename = execfullname;
7514         break;
7515     } else {
7516         execbasename = last_slash + 1;
7517         if (*execbasename == '\0') {
7518             *last_slash = '\0';
7519             continue;
7520         }
7521         break;
7522     }
7523 }
7524 return (execbasename);
7525 }

7527 int
7528 main(int argc, char *argv[])
7529 {
7530     int err, arg;
7531     struct stat st;

7533     /* This must be before anything goes to stdout. */
7534     setbuf(stdout, NULL);

7536     saw_error = B_FALSE;
7537     cmd_file_mode = B_FALSE;
7538     execname = get_execbasename(argv[0]);

7540     (void) setlocale(LC_ALL, "");
7541     (void) textdomain(TEXT_DOMAIN);

7543     if (getzoneid() != GLOBAL_ZONEID) {
7544         zerr(gettext("%s can only be run from the global zone."),
7545             execname);
7546         exit(Z_ERR);
7547     }

7549     if (argc < 2) {
7550         usage(B_FALSE, HELP_USAGE | HELP_SUBCMDS);
7551         exit(Z_USAGE);
7552     }
7553     if (strcmp(argv[1], cmd_to_str(CMD_HELP)) == 0) {
7554         (void) one_command_at_a_time(argc - 1, &(argv[1]));
7555         exit(Z_OK);
7556     }

7558     while ((arg = getopt(argc, argv, "?f:R:z:")) != EOF) {
7559         switch (arg) {
7560             case '?':
7561                 if (optopt == '?')
7562                     usage(B_TRUE, HELP_USAGE | HELP_SUBCMDS);
7563                 else
7564                     usage(B_FALSE, HELP_USAGE);
7565                 exit(Z_USAGE);
7566                 /* NOTREACHED */
7567             case 'f':
7568                 cmd_file_name = optarg;
7569                 cmd_file_mode = B_TRUE;
7570                 break;
7571             case 'R':
7572                 if (*optarg != '/') {
7573                     zerr(gettext("root path must be absolute: %s"),
7574                         optarg);
7575                     exit(Z_USAGE);
7576                 }
7577                 if (stat(optarg, &st) == -1 || !S_ISDIR(st.st_mode)) {
7578                     zerr(gettext(

```

```

7579                 "root path must be a directory: %s"),
7580                 optarg);
7581                 exit(Z_USAGE);
7582             }
7583             zoncfg_set_root(optarg);
7584             break;
7585         case 'z':
7586             if (strcmp(optarg, GLOBAL_ZONENAME) == 0) {
7587                 global_zone = B_TRUE;
7588             } else if (zoncfg_validate_zonename(optarg) != Z_OK) {
7589                 zone_perror(optarg, Z_BOGUS_ZONE_NAME, B_TRUE);
7590                 usage(B_FALSE, HELP_SYNTAX);
7591                 exit(Z_USAGE);
7592             }
7593             (void) strcpy(zone, optarg, sizeof (zone));
7594             (void) strcpy(revert_zone, optarg, sizeof (zone));
7595             break;
7596         default:
7597             usage(B_FALSE, HELP_USAGE);
7598             exit(Z_USAGE);
7599     }
7600 }

7602     if (optind > argc || strcmp(zone, "") == 0) {
7603         usage(B_FALSE, HELP_USAGE);
7604         exit(Z_USAGE);
7605     }

7607     if ((err = zoncfg_access(zone, W_OK)) == Z_OK) {
7608         read_only_mode = B_FALSE;
7609     } else if (err == Z_ACCES) {
7610         read_only_mode = B_TRUE;
7611         /* skip this message in one-off from command line mode */
7612         if (optind == argc)
7613             (void) fprintf(stderr, gettext("WARNING: you do not "
7614                 "have write access to this zone's configuration "
7615                 "file;\ngoing into read-only mode.\n"));
7616     } else {
7617         fprintf(stderr, "%s: Could not access zone configuration "
7618             "store: %s\n", execname, zoncfg_strerror(err));
7619         exit(Z_ERR);
7620     }

7622     if ((handle = zoncfg_init_handle()) == NULL) {
7623         zone_perror(execname, Z_NOMEM, B_TRUE);
7624         exit(Z_ERR);
7625     }

7627     /*
7628     * This may get set back to FALSE again in cmd_file() if cmd_file_name
7629     * is a "real" file as opposed to "-" (i.e. meaning use stdin).
7630     */
7631     if (isatty(STDIN_FILENO))
7632         ok_to_prompt = B_TRUE;
7633     if ((gl = new_GetLine(MAX_LINE_LEN, MAX_CMD_HIST)) == NULL)
7634         exit(Z_ERR);
7635     if (gl_customize_completion(gl, NULL, cmd_cpl_fn) != 0)
7636         exit(Z_ERR);
7637     (void) sigset(SIGINT, SIG_IGN);
7638     if (optind == argc) {
7639         if (!cmd_file_mode)
7640             err = do_interactive();
7641         else
7642             err = cmd_file(cmd_file_name);
7643     } else {
7644         err = one_command_at_a_time(argc - optind, &(argv[optind]));

```

```
7645     }  
7646     zonecfg_fini_handle(handle);  
7647     if (brand != NULL)  
7648         brand_close(brand);  
7649     (void) del_GetLine(gl);  
7650     return (err);  
7651 }
```

new/usr/src/cmd/zoncfg/zoncfg.h

1

```
*****
6708 Thu Jun 30 22:01:52 2016
new/usr/src/cmd/zoncfg/zoncfg.h
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
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 /*
23  * Copyright (c) 2003, 2010, Oracle and/or its affiliates. All rights reserved.
24  */
25
26 #ifndef _ZONECFG_H
27 #define _ZONECFG_H
28
29 /*
30  * header file for zoncfg command
31  */
32
33 #ifdef __cplusplus
34 extern "C" {
35 #endif
36
37 #include <unistd.h>
38
39 #define Z_ERR 1
40 #define Z_USAGE 2
41 #define Z_REPEAT 3
42
43 #define CMD_ADD 0
44 #define CMD_CANCEL 1
45 #define CMD_CLEAR 2
46 #define CMD_COMMIT 3
47 #define CMD_CREATE 4
48 #define CMD_DELETE 5
49 #define CMD_END 6
50 #define CMD_EXIT 7
51 #define CMD_EXPORT 8
52 #define CMD_HELP 9
53 #define CMD_INFO 10
54 #define CMD_REMOVE 11
55 #define CMD_REVERT 12
56 #define CMD_SELECT 13
57 #define CMD_SET 14
58 #define CMD_VERIFY 15
```

new/usr/src/cmd/zoncfg/zoncfg.h

2

```
60 #define CMD_MIN CMD_ADD
61 #define CMD_MAX CMD_VERIFY
62
63 /* resource types: increment RT_MAX when expanding this list */
64 #define RT_UNKNOWN 0
65 #define RT_ZONENAME 1 /* really a property, but for info ... */
66 #define RT_ZONEPATH 2 /* really a property, but for info ... */
67 #define RT_AUTOBOOT 3 /* really a property, but for info ... */
68 #define RT_POOL 4 /* really a property, but for info ... */
69 #define RT_FS 5
70 #define RT_NET 6
71 #define RT_DEVICE 7
72 #define RT_RCTL 8
73 #define RT_ATTR 9
74 #define RT_DATASET 10
75 #define RT_LIMITPRIV 11 /* really a property, but for info ... */
76 #define RT_BOOTARGS 12 /* really a property, but for info ... */
77 #define RT_BRAND 13 /* really a property, but for info ... */
78 #define RT_DCPU 14
79 #define RT_MCAP 15
80 #define RT_MAXLWPS 16 /* really a rctl alias property, but for info */
81 #define RT_MAXSHMEM 17 /* really a rctl alias property, but for info */
82 #define RT_MAXSHMIDS 18 /* really a rctl alias property, but for info */
83 #define RT_MAXMSGIDS 19 /* really a rctl alias property, but for info */
84 #define RT_MAXSEMIDS 20 /* really a rctl alias property, but for info */
85 #define RT_SHARES 21 /* really a rctl alias property, but for info */
86 #define RT_SCHED 22 /* really a property, but for info ... */
87 #define RT_IPTYPE 23 /* really a property, but for info ... */
88 #define RT_PCAP 24
89 #define RT_HOSTID 25 /* really a property, but for info ... */
90 #define RT_ADMIN 26
91 #define RT_FS_ALLOWED 27
92 #define RT_MAXPROCS 28 /* really a rctl alias property, but for info */
93 #define RT_SECFLAGS 29
94 #endif /* ! codereview */
95
96 #define RT_MIN RT_UNKNOWN
97 #define RT_MAX RT_SECFLAGS
98 #define RT_MAX RT_MAXPROCS
99
100 /* property types: increment PT_MAX when expanding this list */
101 #define PT_UNKNOWN 0
102 #define PT_ZONENAME 1
103 #define PT_ZONEPATH 2
104 #define PT_AUTOBOOT 3
105 #define PT_POOL 4
106 #define PT_DIR 5
107 #define PT_SPECIAL 6
108 #define PT_TYPE 7
109 #define PT_OPTIONS 8
110 #define PT_ADDRESS 9
111 #define PT_PHYSICAL 10
112 #define PT_NAME 11
113 #define PT_VALUE 12
114 #define PT_MATCH 13
115 #define PT_PRIV 14
116 #define PT_LIMIT 15
117 #define PT_ACTION 16
118 #define PT_RAW 17
119 #define PT_LIMITPRIV 18
120 #define PT_BOOTARGS 19
121 #define PT_BRAND 20
122 #define PT_NCPUS 21
123 #define PT_IMPORTANCE 22
124 #define PT_SWAP 23
```



```
124 #define PT_LOCKED      24
125 #define PT_SHARES     25
126 #define PT_MAXLWPS    26
127 #define PT_MAXSHMMEM  27
128 #define PT_MAXSHMIDS  28
129 #define PT_MAXMSGIDS  29
130 #define PT_MAXSEMIDS  30
131 #define PT_MAXLOCKEDMEM 31
132 #define PT_MAXSWAP    32
133 #define PT_SCHED      33
134 #define PT_IPTYPE     34
135 #define PT_DEFROUTER  35
136 #define PT_HOSTID     36
137 #define PT_USER       37
138 #define PT_AUTHS      38
139 #define PT_FS_ALLOWED  39
140 #define PT_MAXPROCS   40
141 #define PT_ALLOWED_ADDRESS 41
142 #define PT_DEFAULT    42
143 #define PT_LOWER      43
144 #define PT_UPPER      44
145 #endif /* ! codereview */

147 #define PT_MIN        PT_UNKNOWN
148 #define PT_MAX        PT_UPPER
138 #define PT_MAX        PT_ALLOWED_ADDRESS

150 #define MAX_EQ_PROP_PAIRS      3

152 #define PROP_VAL_SIMPLE      0
153 #define PROP_VAL_COMPLEX    1
154 #define PROP_VAL_LIST       2

156 #define PROP_VAL_MIN        PROP_VAL_SIMPLE
157 #define PROP_VAL_MAX        PROP_VAL_LIST

159 /*
160  * If any subcommand is ever modified to take more than three arguments,
161  * this will need to be incremented.
162  */
163 #define MAX_SUBCMD_ARGS      3

165 typedef struct complex_property {
166     int    cp_type; /* from the PT_* list above */
167     char   *cp_value;
168     struct complex_property *cp_next;
169 } complex_property_t, *complex_property_ptr_t;
unchanged_portion_omitted
```

```

*****
27387 Thu Jun 30 22:01:53 2016
new/usr/src/cmd/zoncfg/zoncfg_grammar.y
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
_____unchanged_portion_omitted_____

129 %start commands

131 %token HELP CREATE EXPORT ADD DELETE REMOVE SELECT SET INFO CANCEL END VERIFY
132 %token COMMIT REVERT EXIT SEMICOLON TOKEN ZONENAME ZONEPATH AUTOBOOT POOL NET
133 %token FS ATTR DEVICE RCTL SPECIAL RAW DIR OPTIONS TYPE ADDRESS PHYSICAL
134 %token IPTYPE HOSTID FS_ALLOWED ALLOWED_ADDRESS
135 %token NAME MATCH PRIV LIMIT ACTION VALUE EQUAL OPEN_SQ_BRACKET CLOSE_SQ_BRACKET
136 %token OPEN_PAREN CLOSE_PAREN COMMA DATASET LIMITPRIV BOOTARGS BRAND PSET PCAP
137 %token MCAP NCPUS IMPORTANCE SHARES MAXLWPS MAXSHMEM MAXSHMIDS MAXMSGIDS
138 %token MAXSEMIDS LOCKED SWAP SCHED CLEAR DEFROUTER ADMIN SECFLAGS USER AUTHS MAX
139 %token DEFAULT UPPER LOWER
138 %token MAXSEMIDS LOCKED SWAP SCHED CLEAR DEFROUTER ADMIN USER AUTHS MAXPROCS

141 %type <strval> TOKEN EQUAL OPEN_SQ_BRACKET CLOSE_SQ_BRACKET
142     property_value OPEN_PAREN CLOSE_PAREN COMMA simple_prop_val
143 %type <complex> complex_piece complex_prop_val
144 %type <ival> resource_type NET FS DEVICE RCTL ATTR DATASET PSET PCAP MCAP
145     ADMIN SECFLAGS
146 %type <ival> property_name SPECIAL RAW DIR OPTIONS TYPE ADDRESS PHYSICAL NAME
147     MATCH ZONENAME ZONEPATH AUTOBOOT POOL LIMITPRIV BOOTARGS VALUE PRIV LIMIT
148     ACTION BRAND SCHED IPTYPE DEFROUTER HOSTID USER AUTHS FS_ALLOWED
149     ALLOWED_ADDRESS DEFAULT UPPER LOWER
150 %type <cmd> command
151 %type <cmd> add_command ADD
152 %type <cmd> cancel_command CANCEL
153 %type <cmd> commit_command COMMIT
154 %type <cmd> create_command CREATE
155 %type <cmd> delete_command DELETE
156 %type <cmd> end_command END
157 %type <cmd> exit_command EXIT
158 %type <cmd> export_command EXPORT
159 %type <cmd> help_command HELP
160 %type <cmd> info_command INFO
161 %type <cmd> remove_command REMOVE
162 %type <cmd> revert_command REVERT
163 %type <cmd> select_command SELECT
164 %type <cmd> set_command SET
165 %type <cmd> clear_command CLEAR
166 %type <cmd> verify_command VERIFY
167 %type <cmd> terminator

169 %%

171 /*
172  * NOTE: Each commands reduction rule must invoke assert_no_unclaimed_tokens()
173  * before it completes if it isn't processing an error. This ensures that
174  * reduction rules properly consume TOKENS.
175  */
176 commands: command terminator
177     {
178         if ($1 != NULL) {
179             if ($1->cmd_handler != NULL)
180                 $1->cmd_handler($1);
181             free_cmd($1);

```

```

182         bzero(list, sizeof (list_property_t));
183         num_prop_vals = 0;
184     }
185     assert_no_unclaimed_tokens();
186     return (0);
187 }
188     command error terminator
189 }
190     if ($1 != NULL) {
191         free_cmd($1);
192         bzero(list, sizeof (list_property_t));
193         num_prop_vals = 0;
194     }
195     if (YYRECOVERING())
196         YYABORT;
197     yyclearin;
198     yyerrok;
199 }
200     error terminator
201 }
202     if (YYRECOVERING())
203         YYABORT;
204     yyclearin;
205     yyerrok;
206 }
207     terminator
208 }
209     assert_no_unclaimed_tokens();
210     return (0);
211 }

213 command: add_command
214     cancel_command
215     clear_command
216     create_command
217     commit_command
218     delete_command
219     end_command
220     exit_command
221     export_command
222     help_command
223     info_command
224     remove_command
225     revert_command
226     select_command
227     set_command
228     verify_command

230 terminator: '\n' { newline_terminated = B_TRUE; }
231             ';'  { newline_terminated = B_FALSE; }

233 add_command: ADD
234     {
235         short_usage(CMD_ADD);
236         (void) fputs("\n", stderr);
237         usage(B_FALSE, HELP_RES_PROPS);
238         YYERROR;
239     }
240     ADD TOKEN
241 }
242     if (($$ = alloc_cmd()) == NULL)
243         YYERROR;
244     cmd = $$;
245     $$->cmd_handler = &add_func;
246     $$->cmd_argc = 1;
247     $$->cmd_argv[0] = claim_token($2);

```

```

248     $$->cmd_argv[1] = NULL;
249
250     {
251     ADD resource_type
252         if (($$ = alloc_cmd()) == NULL)
253             YYERROR;
254         cmd = $$;
255         $$->cmd_handler = &add_func;
256         $$->cmd_argc = 0;
257         $$->cmd_res_type = $2;
258         $$->cmd_prop_nv_pairs = 0;
259     }
260     {
261     ADD property_name property_value
262         if (($$ = alloc_cmd()) == NULL)
263             YYERROR;
264         cmd = $$;
265         $$->cmd_handler = &add_func;
266         $$->cmd_argc = 0;
267         $$->cmd_prop_nv_pairs = 1;
268         $$->cmd_prop_name[0] = $2;
269         $$->cmd_property_ptr[0] = &property[0];
270     }
271
272 cancel_command: CANCEL
273 {
274     if (($$ = alloc_cmd()) == NULL)
275         YYERROR;
276     cmd = $$;
277     $$->cmd_handler = &cancel_func;
278     $$->cmd_argc = 0;
279     $$->cmd_argv[0] = NULL;
280 }
281 {
282 CANCEL TOKEN
283     if (($$ = alloc_cmd()) == NULL)
284         YYERROR;
285     cmd = $$;
286     $$->cmd_handler = &cancel_func;
287     $$->cmd_argc = 1;
288     $$->cmd_argv[0] = claim_token($2);
289     $$->cmd_argv[1] = NULL;
290 }
291
292 create_command: CREATE
293 {
294     if (($$ = alloc_cmd()) == NULL)
295         YYERROR;
296     cmd = $$;
297     $$->cmd_handler = &create_func;
298     $$->cmd_argc = 0;
299     $$->cmd_argv[0] = NULL;
300 }
301 {
302 CREATE TOKEN
303     if (($$ = alloc_cmd()) == NULL)
304         YYERROR;
305     cmd = $$;
306     $$->cmd_handler = &create_func;
307     $$->cmd_argc = 1;
308     $$->cmd_argv[0] = claim_token($2);
309     $$->cmd_argv[1] = NULL;
310 }
311 {
312 CREATE TOKEN TOKEN
313     if (($$ = alloc_cmd()) == NULL)

```

```

314         YYERROR;
315         cmd = $$;
316         $$->cmd_handler = &create_func;
317         $$->cmd_argc = 2;
318         $$->cmd_argv[0] = claim_token($2);
319         $$->cmd_argv[1] = claim_token($3);
320         $$->cmd_argv[2] = NULL;
321     }
322     {
323 CREATE TOKEN TOKEN TOKEN
324         if (($$ = alloc_cmd()) == NULL)
325             YYERROR;
326         cmd = $$;
327         $$->cmd_handler = &create_func;
328         $$->cmd_argc = 3;
329         $$->cmd_argv[0] = claim_token($2);
330         $$->cmd_argv[1] = claim_token($3);
331         $$->cmd_argv[2] = claim_token($4);
332         $$->cmd_argv[3] = NULL;
333     }
334
335 commit_command: COMMIT
336 {
337     if (($$ = alloc_cmd()) == NULL)
338         YYERROR;
339     cmd = $$;
340     $$->cmd_handler = &commit_func;
341     $$->cmd_argc = 0;
342     $$->cmd_argv[0] = NULL;
343 }
344 {
345 COMMIT TOKEN
346     if (($$ = alloc_cmd()) == NULL)
347         YYERROR;
348     cmd = $$;
349     $$->cmd_handler = &commit_func;
350     $$->cmd_argc = 1;
351     $$->cmd_argv[0] = claim_token($2);
352     $$->cmd_argv[1] = NULL;
353 }
354
355 delete_command: DELETE
356 {
357     if (($$ = alloc_cmd()) == NULL)
358         YYERROR;
359     cmd = $$;
360     $$->cmd_handler = &delete_func;
361     $$->cmd_argc = 0;
362     $$->cmd_argv[0] = NULL;
363 }
364 {
365 DELETE TOKEN
366     if (($$ = alloc_cmd()) == NULL)
367         YYERROR;
368     cmd = $$;
369     $$->cmd_handler = &delete_func;
370     $$->cmd_argc = 1;
371     $$->cmd_argv[0] = claim_token($2);
372     $$->cmd_argv[1] = NULL;
373 }
374
375 end_command: END
376 {
377     if (($$ = alloc_cmd()) == NULL)
378         YYERROR;
379     cmd = $$;

```

```

380     $$->cmd_handler = &end_func;
381     $$->cmd_argc = 0;
382     $$->cmd_argv[0] = NULL;
383 }
384 { END TOKEN
385 }
386     if (($$ = alloc_cmd()) == NULL)
387         YYERROR;
388     cmd = $$;
389     $$->cmd_handler = &end_func;
390     $$->cmd_argc = 1;
391     $$->cmd_argv[0] = claim_token($2);
392     $$->cmd_argv[1] = NULL;
393 }

395 exit_command: EXIT
396 {
397     if (($$ = alloc_cmd()) == NULL)
398         YYERROR;
399     cmd = $$;
400     $$->cmd_handler = &exit_func;
401     $$->cmd_argc = 0;
402     $$->cmd_argv[0] = NULL;
403 }
404 { EXIT TOKEN
405 }
406     if (($$ = alloc_cmd()) == NULL)
407         YYERROR;
408     cmd = $$;
409     $$->cmd_handler = &exit_func;
410     $$->cmd_argc = 1;
411     $$->cmd_argv[0] = claim_token($2);
412     $$->cmd_argv[1] = NULL;
413 }

415 export_command: EXPORT
416 {
417     if (($$ = alloc_cmd()) == NULL)
418         YYERROR;
419     cmd = $$;
420     $$->cmd_handler = &export_func;
421     $$->cmd_argc = 0;
422     $$->cmd_argv[0] = NULL;
423 }
424 { EXPORT TOKEN
425 }
426     if (($$ = alloc_cmd()) == NULL)
427         YYERROR;
428     cmd = $$;
429     $$->cmd_handler = &export_func;
430     $$->cmd_argc = 1;
431     $$->cmd_argv[0] = claim_token($2);
432     $$->cmd_argv[1] = NULL;
433 }
434 { EXPORT TOKEN TOKEN
435 }
436     if (($$ = alloc_cmd()) == NULL)
437         YYERROR;
438     cmd = $$;
439     $$->cmd_handler = &export_func;
440     $$->cmd_argc = 2;
441     $$->cmd_argv[0] = claim_token($2);
442     $$->cmd_argv[1] = claim_token($3);
443     $$->cmd_argv[2] = NULL;
444 }

```

```

446 help_command: HELP
447 {
448     if (($$ = alloc_cmd()) == NULL)
449         YYERROR;
450     cmd = $$;
451     $$->cmd_handler = &help_func;
452     $$->cmd_argc = 0;
453     $$->cmd_argv[0] = NULL;
454 }
455 { HELP TOKEN
456 }
457     if (($$ = alloc_cmd()) == NULL)
458         YYERROR;
459     cmd = $$;
460     $$->cmd_handler = &help_func;
461     $$->cmd_argc = 1;
462     $$->cmd_argv[0] = claim_token($2);
463     $$->cmd_argv[1] = NULL;
464 }

466 info_command: INFO
467 {
468     if (($$ = alloc_cmd()) == NULL)
469         YYERROR;
470     cmd = $$;
471     $$->cmd_handler = &info_func;
472     $$->cmd_res_type = RT_UNKNOWN;
473     $$->cmd_prop_nv_pairs = 0;
474 }
475 { INFO TOKEN
476 }
477     short_usage(CMD_INFO);
478     (void) fputs("\n", stderr);
479     usage(B_FALSE, HELP_RES_PROPS);
480     free(claim_token($2));
481     YYERROR;
482 }
483 { INFO resource_type
484 }
485     if (($$ = alloc_cmd()) == NULL)
486         YYERROR;
487     cmd = $$;
488     $$->cmd_handler = &info_func;
489     $$->cmd_res_type = $2;
490     $$->cmd_prop_nv_pairs = 0;
491 }
492 { INFO ZONENAME
493 }
494     if (($$ = alloc_cmd()) == NULL)
495         YYERROR;
496     cmd = $$;
497     $$->cmd_handler = &info_func;
498     $$->cmd_res_type = RT_ZONENAME;
499     $$->cmd_prop_nv_pairs = 0;
500 }
501 { INFO ZONEPATH
502 }
503     if (($$ = alloc_cmd()) == NULL)
504         YYERROR;
505     cmd = $$;
506     $$->cmd_handler = &info_func;
507     $$->cmd_res_type = RT_ZONEPATH;
508     $$->cmd_prop_nv_pairs = 0;
509 }
510 { INFO BRAND
511 }

```

```

512     if (($$ = alloc_cmd()) == NULL)
513         YYERROR;
514     cmd = $$;
515     $$->cmd_handler = &info_func;
516     $$->cmd_res_type = RT_BRAND;
517     $$->cmd_prop_nv_pairs = 0;
518 }
519     INFO AUTOBOOT
520 }
521     if (($$ = alloc_cmd()) == NULL)
522         YYERROR;
523     cmd = $$;
524     $$->cmd_handler = &info_func;
525     $$->cmd_res_type = RT_AUTOBOOT;
526     $$->cmd_prop_nv_pairs = 0;
527 }
528     INFO IPTYPE
529 }
530     if (($$ = alloc_cmd()) == NULL)
531         YYERROR;
532     cmd = $$;
533     $$->cmd_handler = &info_func;
534     $$->cmd_res_type = RT_IPTYPE;
535     $$->cmd_prop_nv_pairs = 0;
536 }
537     INFO POOL
538 }
539     if (($$ = alloc_cmd()) == NULL)
540         YYERROR;
541     cmd = $$;
542     $$->cmd_handler = &info_func;
543     $$->cmd_res_type = RT_POOL;
544     $$->cmd_prop_nv_pairs = 0;
545 }
546     INFO LIMITPRIV
547 }
548     if (($$ = alloc_cmd()) == NULL)
549         YYERROR;
550     cmd = $$;
551     $$->cmd_handler = &info_func;
552     $$->cmd_res_type = RT_LIMITPRIV;
553     $$->cmd_prop_nv_pairs = 0;
554 }
555     INFO BOOTARGS
556 }
557     if (($$ = alloc_cmd()) == NULL)
558         YYERROR;
559     cmd = $$;
560     $$->cmd_handler = &info_func;
561     $$->cmd_res_type = RT_BOOTARGS;
562     $$->cmd_prop_nv_pairs = 0;
563 }
564     INFO SCHED
565 }
566     if (($$ = alloc_cmd()) == NULL)
567         YYERROR;
568     cmd = $$;
569     $$->cmd_handler = &info_func;
570     $$->cmd_res_type = RT_SCHED;
571     $$->cmd_prop_nv_pairs = 0;
572 }
573     INFO SHARES
574 }
575     if (($$ = alloc_cmd()) == NULL)
576         YYERROR;
577     cmd = $$;

```

```

578     $$->cmd_handler = &info_func;
579     $$->cmd_res_type = RT_SHARES;
580     $$->cmd_prop_nv_pairs = 0;
581 }
582     INFO MAXLWPS
583 }
584     if (($$ = alloc_cmd()) == NULL)
585         YYERROR;
586     cmd = $$;
587     $$->cmd_handler = &info_func;
588     $$->cmd_res_type = RT_MAXLWPS;
589     $$->cmd_prop_nv_pairs = 0;
590 }
591     INFO MAXPROCS
592 }
593     if (($$ = alloc_cmd()) == NULL)
594         YYERROR;
595     cmd = $$;
596     $$->cmd_handler = &info_func;
597     $$->cmd_res_type = RT_MAXPROCS;
598     $$->cmd_prop_nv_pairs = 0;
599 }
600     INFO MAXSHMMEM
601 }
602     if (($$ = alloc_cmd()) == NULL)
603         YYERROR;
604     cmd = $$;
605     $$->cmd_handler = &info_func;
606     $$->cmd_res_type = RT_MAXSHMMEM;
607     $$->cmd_prop_nv_pairs = 0;
608 }
609     INFO MAXSHMIDS
610 }
611     if (($$ = alloc_cmd()) == NULL)
612         YYERROR;
613     cmd = $$;
614     $$->cmd_handler = &info_func;
615     $$->cmd_res_type = RT_MAXSHMIDS;
616     $$->cmd_prop_nv_pairs = 0;
617 }
618     INFO MAXMSGIDS
619 }
620     if (($$ = alloc_cmd()) == NULL)
621         YYERROR;
622     cmd = $$;
623     $$->cmd_handler = &info_func;
624     $$->cmd_res_type = RT_MAXMSGIDS;
625     $$->cmd_prop_nv_pairs = 0;
626 }
627     INFO MAXSEMIDS
628 }
629     if (($$ = alloc_cmd()) == NULL)
630         YYERROR;
631     cmd = $$;
632     $$->cmd_handler = &info_func;
633     $$->cmd_res_type = RT_MAXSEMIDS;
634     $$->cmd_prop_nv_pairs = 0;
635 }
636     INFO HOSTID
637 }
638     if (($$ = alloc_cmd()) == NULL)
639         YYERROR;
640     cmd = $$;
641     $$->cmd_handler = &info_func;
642     $$->cmd_res_type = RT_HOSTID;
643     $$->cmd_prop_nv_pairs = 0;

```

```

644 }
645     INFO FS_ALLOWED
646 }
647     if (($$ = alloc_cmd()) == NULL)
648         YYERROR;
649     cmd = $$;
650     $$->cmd_handler = &info_func;
651     $$->cmd_res_type = RT_FS_ALLOWED;
652     $$->cmd_prop_nv_pairs = 0;
653 }
654     INFO resource_type property_name EQUAL property_value
655 }
656     if (($$ = alloc_cmd()) == NULL)
657         YYERROR;
658     cmd = $$;
659     $$->cmd_handler = &info_func;
660     $$->cmd_res_type = $2;
661     $$->cmd_prop_nv_pairs = 1;
662     $$->cmd_prop_name[0] = $3;
663     $$->cmd_property_ptr[0] = &property[0];
664 }
665     INFO resource_type property_name EQUAL property_value property_n
666 }
667     if (($$ = alloc_cmd()) == NULL)
668         YYERROR;
669     cmd = $$;
670     $$->cmd_handler = &info_func;
671     $$->cmd_res_type = $2;
672     $$->cmd_prop_nv_pairs = 2;
673     $$->cmd_prop_name[0] = $3;
674     $$->cmd_property_ptr[0] = &property[0];
675     $$->cmd_prop_name[1] = $6;
676     $$->cmd_property_ptr[1] = &property[1];
677 }
678     INFO resource_type property_name EQUAL property_value property_n
679 }
680     if (($$ = alloc_cmd()) == NULL)
681         YYERROR;
682     cmd = $$;
683     $$->cmd_handler = &info_func;
684     $$->cmd_res_type = $2;
685     $$->cmd_prop_nv_pairs = 3;
686     $$->cmd_prop_name[0] = $3;
687     $$->cmd_property_ptr[0] = &property[0];
688     $$->cmd_prop_name[1] = $6;
689     $$->cmd_property_ptr[1] = &property[1];
690     $$->cmd_prop_name[2] = $9;
691     $$->cmd_property_ptr[2] = &property[2];
692 }
693
694 remove_command: REMOVE
695 {
696     short_usage(CMD_REMOVE);
697     (void) fputs("\n", stderr);
698     usage(B_FALSE, HELP_RES_PROPS);
699     YYERROR;
700 }
701 REMOVE TOKEN
702 {
703     short_usage(CMD_REMOVE);
704     (void) fputs("\n", stderr);
705     usage(B_FALSE, HELP_RES_PROPS);
706     free(claim_token($2));
707     YYERROR;
708 }
709 REMOVE resource_type

```

```

710 {
711     if (($$ = alloc_cmd()) == NULL)
712         YYERROR;
713     cmd = $$;
714     $$->cmd_handler = &remove_func;
715     $$->cmd_res_type = $2;
716 }
717 REMOVE TOKEN resource_type
718 {
719     if (($$ = alloc_cmd()) == NULL)
720         YYERROR;
721     cmd = $$;
722     $$->cmd_handler = &remove_func;
723     $$->cmd_res_type = $3;
724     $$->cmd_argc = 1;
725     $$->cmd_argv[0] = claim_token($2);
726     $$->cmd_argv[1] = NULL;
727 }
728 REMOVE property_name property_value
729 {
730     if (($$ = alloc_cmd()) == NULL)
731         YYERROR;
732     cmd = $$;
733     $$->cmd_handler = &remove_func;
734     $$->cmd_prop_nv_pairs = 1;
735     $$->cmd_prop_name[0] = $2;
736     $$->cmd_property_ptr[0] = &property[0];
737 }
738 REMOVE resource_type property_name EQUAL property_value
739 {
740     if (($$ = alloc_cmd()) == NULL)
741         YYERROR;
742     cmd = $$;
743     $$->cmd_handler = &remove_func;
744     $$->cmd_res_type = $2;
745     $$->cmd_prop_nv_pairs = 1;
746     $$->cmd_prop_name[0] = $3;
747     $$->cmd_property_ptr[0] = &property[0];
748 }
749 REMOVE resource_type property_name EQUAL property_value property_name
750 {
751     if (($$ = alloc_cmd()) == NULL)
752         YYERROR;
753     cmd = $$;
754     $$->cmd_handler = &remove_func;
755     $$->cmd_res_type = $2;
756     $$->cmd_prop_nv_pairs = 2;
757     $$->cmd_prop_name[0] = $3;
758     $$->cmd_property_ptr[0] = &property[0];
759     $$->cmd_prop_name[1] = $6;
760     $$->cmd_property_ptr[1] = &property[1];
761 }
762 REMOVE resource_type property_name EQUAL property_value property_name
763 {
764     if (($$ = alloc_cmd()) == NULL)
765         YYERROR;
766     cmd = $$;
767     $$->cmd_handler = &remove_func;
768     $$->cmd_res_type = $2;
769     $$->cmd_prop_nv_pairs = 3;
770     $$->cmd_prop_name[0] = $3;
771     $$->cmd_property_ptr[0] = &property[0];
772     $$->cmd_prop_name[1] = $6;
773     $$->cmd_property_ptr[1] = &property[1];
774     $$->cmd_prop_name[2] = $9;
775     $$->cmd_property_ptr[2] = &property[2];

```

```

776     }
777
778 revert_command: REVERT
779     {
780         if (($$ = alloc_cmd()) == NULL)
781             YYERROR;
782         cmd = $$;
783         $$->cmd_handler = &revert_func;
784         $$->cmd_argc = 0;
785         $$->cmd_argv[0] = NULL;
786     }
787     REVERT TOKEN
788     {
789         if (($$ = alloc_cmd()) == NULL)
790             YYERROR;
791         cmd = $$;
792         $$->cmd_handler = &revert_func;
793         $$->cmd_argc = 1;
794         $$->cmd_argv[0] = claim_token($2);
795         $$->cmd_argv[1] = NULL;
796     }
797
798 select_command: SELECT
799     {
800         short_usage(CMD_SELECT);
801         (void) fputs("\n", stderr);
802         usage(B_FALSE, HELP_RES_PROPS);
803         YYERROR;
804     }
805     SELECT PSET
806     {
807         if (($$ = alloc_cmd()) == NULL)
808             YYERROR;
809         cmd = $$;
810         $$->cmd_handler = &select_func;
811         $$->cmd_res_type = RT_DCPU;
812     }
813     SELECT PCAP
814     {
815         if (($$ = alloc_cmd()) == NULL)
816             YYERROR;
817         cmd = $$;
818         $$->cmd_handler = &select_func;
819         $$->cmd_res_type = RT_PCAP;
820     }
821     SELECT MCAP
822     {
823         if (($$ = alloc_cmd()) == NULL)
824             YYERROR;
825         cmd = $$;
826         $$->cmd_handler = &select_func;
827         $$->cmd_res_type = RT_MCAP;
828     }
829     SELECT resource_type
830     {
831         short_usage(CMD_SELECT);
832         YYERROR;
833     }
834     SELECT resource_type property_name EQUAL property_value
835     {
836         if (($$ = alloc_cmd()) == NULL)
837             YYERROR;
838         cmd = $$;
839         $$->cmd_handler = &select_func;
840         $$->cmd_res_type = $2;
841         $$->cmd_prop_nv_pairs = 1;

```

```

842         $$->cmd_prop_name[0] = $3;
843         $$->cmd_property_ptr[0] = &property[0];
844     }
845     SELECT resource_type property_name EQUAL property_value property_name
846     {
847         if (($$ = alloc_cmd()) == NULL)
848             YYERROR;
849         cmd = $$;
850         $$->cmd_handler = &select_func;
851         $$->cmd_res_type = $2;
852         $$->cmd_prop_nv_pairs = 2;
853         $$->cmd_prop_name[0] = $3;
854         $$->cmd_property_ptr[0] = &property[0];
855         $$->cmd_prop_name[1] = $6;
856         $$->cmd_property_ptr[1] = &property[1];
857     }
858     SELECT resource_type property_name EQUAL property_value property_name
859     {
860         if (($$ = alloc_cmd()) == NULL)
861             YYERROR;
862         cmd = $$;
863         $$->cmd_handler = &select_func;
864         $$->cmd_res_type = $2;
865         $$->cmd_prop_nv_pairs = 3;
866         $$->cmd_prop_name[0] = $3;
867         $$->cmd_property_ptr[0] = &property[0];
868         $$->cmd_prop_name[1] = $6;
869         $$->cmd_property_ptr[1] = &property[1];
870         $$->cmd_prop_name[2] = $9;
871         $$->cmd_property_ptr[2] = &property[2];
872     }
873
874 set_command: SET
875     {
876         short_usage(CMD_SET);
877         (void) fputs("\n", stderr);
878         usage(B_FALSE, HELP_PROPS);
879         YYERROR;
880     }
881     SET property_name EQUAL OPEN_SQ_BRACKET CLOSE_SQ_BRACKET
882     {
883         if (($$ = alloc_cmd()) == NULL)
884             YYERROR;
885         cmd = $$;
886         $$->cmd_handler = &set_func;
887         $$->cmd_prop_nv_pairs = 0;
888         $$->cmd_prop_name[0] = $2;
889         property[0].pv_type = PROP_VAL_LIST;
890         property[0].pv_list = NULL;
891         $$->cmd_property_ptr[0] = &property[0];
892     }
893     SET property_name EQUAL property_value
894     {
895         if (($$ = alloc_cmd()) == NULL)
896             YYERROR;
897         cmd = $$;
898         $$->cmd_handler = &set_func;
899         $$->cmd_prop_nv_pairs = 1;
900         $$->cmd_prop_name[0] = $2;
901         $$->cmd_property_ptr[0] = &property[0];
902     }
903     SET TOKEN ZONEPATH EQUAL property_value
904     {
905         if (($$ = alloc_cmd()) == NULL)
906             YYERROR;
907         cmd = $$;

```

```

908     $$->cmd_argc = 1;
909     $$->cmd_argv[0] = claim_token($2);
910     $$->cmd_argv[1] = NULL;
911     $$->cmd_handler = &set_func;
912     $$->cmd_prop_nv_pairs = 1;
913     $$->cmd_prop_name[0] = PT_ZONEPATH;
914     $$->cmd_property_ptr[0] = &property[0];
915 }

917 clear_command: CLEAR
918 {
919     short_usage(CMD_CLEAR);
920     (void) fputs("\n", stderr);
921     usage(B_FALSE, HELP_PROPS);
922     YYERROR;
923 }
924 CLEAR property_name
925 {
926     if (($$ = alloc_cmd()) == NULL)
927         YYERROR;
928     cmd = $$;
929     $$->cmd_handler = &clear_func;
930     $$->cmd_res_type = $2;
931 }

933 verify_command: VERIFY
934 {
935     if (($$ = alloc_cmd()) == NULL)
936         YYERROR;
937     cmd = $$;
938     $$->cmd_handler = &verify_func;
939     $$->cmd_argc = 0;
940     $$->cmd_argv[0] = NULL;
941 }
942 VERIFY TOKEN
943 {
944     if (($$ = alloc_cmd()) == NULL)
945         YYERROR;
946     cmd = $$;
947     $$->cmd_handler = &verify_func;
948     $$->cmd_argc = 1;
949     $$->cmd_argv[0] = claim_token($2);
950     $$->cmd_argv[1] = NULL;
951 }

953 resource_type: NET      { $$ = RT_NET; }
954                 FS      { $$ = RT_FS; }
955                 DEVICE  { $$ = RT_DEVICE; }
956                 RCTL    { $$ = RT_RCTL; }
957                 ATTR    { $$ = RT_ATTR; }
958                 DATASET { $$ = RT_DATASET; }
959                 PSET    { $$ = RT_DCPU; }
960                 PCAP    { $$ = RT_PCAP; }
961                 MCAP    { $$ = RT_MCAP; }
962                 ADMIN   { $$ = RT_ADMIN; }
963                 SECFLAGS { $$ = RT_SECFLAGS; }
964 #endif /* ! codereview */

966 property_name: SPECIAL { $$ = PT_SPECIAL; }
967                 RAW     { $$ = PT_RAW; }
968                 DIR     { $$ = PT_DIR; }
969                 TYPE    { $$ = PT_TYPE; }
970                 OPTIONS { $$ = PT_OPTIONS; }
971                 ZONENAME { $$ = PT_ZONENAME; }
972                 ZONEPATH { $$ = PT_ZONEPATH; }
973                 AUTOBOOT { $$ = PT_AUTOBOOT; }

```

```

974 IPTYPE { $$ = PT_IPTYPE; }
975 POOL   { $$ = PT_POOL; }
976 LIMITPRIV { $$ = PT_LIMITPRIV; }
977 BOOTARGS { $$ = PT_BOOTARGS; }
978 ADDRESS  { $$ = PT_ADDRESS; }
979 ALLOWED_ADDRESS { $$ = PT_ALLOWED_ADDRESS; }
980 PHYSICAL { $$ = PT_PHYSICAL; }
981 DEFROUTER { $$ = PT_DEFROUTER; }
982 NAME      { $$ = PT_NAME; }
983 VALUE    { $$ = PT_VALUE; }
984 MATCH    { $$ = PT_MATCH; }
985 PRIV     { $$ = PT_PRIV; }
986 LIMIT    { $$ = PT_LIMIT; }
987 ACTION   { $$ = PT_ACTION; }
988 BRAND    { $$ = PT_BRAND; }
989 NCPUS    { $$ = PT_NCPUS; }
990 LOCKED   { $$ = PT_LOCKED; }
991 SWAP     { $$ = PT_SWAP; }
992 IMPORTANCE { $$ = PT_IMPORTANCE; }
993 SHARES   { $$ = PT_SHARES; }
994 MAXLWPS  { $$ = PT_MAXLWPS; }
995 MAXPROCS { $$ = PT_MAXPROCS; }
996 MAXSHMEM { $$ = PT_MAXSHMEM; }
997 MAXSHMIDS { $$ = PT_MAXSHMIDS; }
998 MAXMSGIDS { $$ = PT_MAXMSGIDS; }
999 MAXSEMIDS { $$ = PT_MAXSEMIDS; }
1000 SCHED    { $$ = PT_SCHED; }
1001 HOSTID   { $$ = PT_HOSTID; }
1002 USER     { $$ = PT_USER; }
1003 AUTHS    { $$ = PT_AUTHS; }
1004 FS_ALLOWED { $$ = PT_FS_ALLOWED; }
1005 DEFAULT  { $$ = PT_DEFAULT; }
1006 UPPER    { $$ = PT_UPPER; }
1007 LOWER    { $$ = PT_LOWER; }
1008 #endif /* ! codereview */

1010 /*
1011  * The grammar builds data structures from the bottom up. Thus various
1012  * strings are lexed into TOKENS or commands or resource or property values.
1013  * Below is where the resource and property values are built up into more
1014  * complex data structures.
1015  *
1016  * There are three kinds of properties: simple (single valued), complex
1017  * (one or more name=value pairs) and list (concatenation of one or more
1018  * simple or complex properties).
1019  *
1020  * So the property structure has a type which is one of these, and the
1021  * corresponding _simple, _complex or _list is set to the corresponding
1022  * lower-level data structure.
1023  */

1025 property_value: simple_prop_val
1026 {
1027     property[num_prop_vals].pv_type = PROP_VAL_SIMPLE;
1028     property[num_prop_vals].pv_simple = $1;
1029     if (list[num_prop_vals] != NULL) {
1030         free_outer_list(list[num_prop_vals]);
1031         list[num_prop_vals] = NULL;
1032     }
1033     num_prop_vals++;
1034 }
1035 {
1036     complex_prop_val
1037 {
1038     property[num_prop_vals].pv_type = PROP_VAL_COMPLEX;
1039     property[num_prop_vals].pv_complex = complex;
1040     if (list[num_prop_vals] != NULL) {

```



```

1040         free_outer_list(list[num_prop_vals]);
1041         list[num_prop_vals] = NULL;
1042     }
1043     num_prop_vals++;
1044 }
1045     list_prop_val
1046     {
1047         property[num_prop_vals].pv_type = PROP_VAL_LIST;
1048         property[num_prop_vals].pv_list = list[num_prop_vals];
1049         num_prop_vals++;
1050     }
1051
1052 /*
1053  * One level lower, lists are made up of simple or complex values, so
1054  * simple_prop_val and complex_prop_val fill in a list structure and
1055  * insert it into the linked list which is built up. And because
1056  * complex properties can have multiple name=value pairs, we keep
1057  * track of them in another linked list.
1058  *
1059  * The complex and list structures for the linked lists are allocated
1060  * below, and freed by recursive functions which are ultimately called
1061  * by free_cmd(), which is called from the top-most "commands" part of
1062  * the grammar.
1063  *
1064  * NOTE: simple_prop_val and complex_piece need reduction rules for
1065  * property_name and resource_type so that the parser will accept property names
1066  * and resource type names as property values.
1067  */
1068
1069 simple_prop_val: TOKEN
1070 {
1071     $$ = simple_prop_val_func($1);
1072     free(claim_token($1));
1073     if ($$ == NULL)
1074         YYERROR;
1075 }
1076     resource_type
1077     {
1078         if (($$ = simple_prop_val_func(res_types[$1])) == NULL)
1079             YYERROR;
1080 }
1081     property_name
1082     {
1083         if (($$ = simple_prop_val_func(prop_types[$1])) == NULL)
1084             YYERROR;
1085     }
1086
1087 complex_prop_val: OPEN_PAREN complex_piece CLOSE_PAREN
1088 {
1089     if ((new_list = alloc_list()) == NULL)
1090         YYERROR;
1091     new_list->lp_simple = NULL;
1092     new_list->lp_complex = complex;
1093     new_list->lp_next = NULL;
1094     if (list[num_prop_vals] == NULL) {
1095         list[num_prop_vals] = new_list;
1096     } else {
1097         for (tmp_list = list[num_prop_vals]; tmp_list != NULL;
1098             tmp_list = tmp_list->lp_next)
1099             last = tmp_list;
1100         last->lp_next = new_list;
1101     }
1102 }
1103
1104 complex_piece: property_name EQUAL TOKEN
1105 {

```

```

1106     $$ = complex_piece_func($1, $3, NULL);
1107     free(claim_token($3));
1108     if ($$ == NULL)
1109         YYERROR;
1110 }
1111     property_name EQUAL resource_type
1112     {
1113         if (($$ = complex_piece_func($1, res_types[$3], NULL)) == NULL)
1114             YYERROR;
1115     }
1116     property_name EQUAL property_name
1117     {
1118         if (($$ = complex_piece_func($1, prop_types[$3], NULL)) == NULL)
1119             YYERROR;
1120     }
1121     property_name EQUAL TOKEN COMMA complex_piece
1122     {
1123         $$ = complex_piece_func($1, $3, complex);
1124         free(claim_token($3));
1125         if ($$ == NULL)
1126             YYERROR;
1127     }
1128     property_name EQUAL resource_type COMMA complex_piece
1129     {
1130         if (($$ = complex_piece_func($1, res_types[$3], complex)) ==
1131             NULL)
1132             YYERROR;
1133     }
1134     property_name EQUAL property_name COMMA complex_piece
1135     {
1136         if (($$ = complex_piece_func($1, prop_types[$3], complex)) ==
1137             NULL)
1138             YYERROR;
1139     }
1140
1141 list_piece: simple_prop_val
1142           | complex_prop_val
1143           | simple_prop_val COMMA list_piece
1144           | complex_prop_val COMMA list_piece
1145
1146 list_prop_val: OPEN_SQ_BRACKET list_piece CLOSE_SQ_BRACKET
1147 %%

```

```

*****
11450 Thu Jun 30 22:01:54 2016
new/usr/src/cmd/zoncfg/zoncfg_lex.1
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
1  %{
2  /*
3  * CDDL HEADER START
4  *
5  * The contents of this file are subject to the terms of the
6  * Common Development and Distribution License (the "License").
7  * You may not use this file except in compliance with the License.
8  *
9  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
10 * or http://www.opensolaris.org/os/licensing.
11 * See the License for the specific language governing permissions
12 * and limitations under the License.
13 *
14 * When distributing Covered Code, include this CDDL HEADER in each
15 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
16 * If applicable, add the following below this CDDL HEADER, with the
17 * fields enclosed by brackets "[]" replaced with your own identifying
18 * information: Portions Copyright [yyyy] [name of copyright owner]
19 *
20 * CDDL HEADER END
21 */

23 /*
24 * Copyright (c) 2003, 2010, Oracle and/or its affiliates. All rights reserved.
25 */

27 #include <assert.h>
28 #include <string.h>
29 #include <libintl.h>
30 #include "zoncfg.h"
31 #include "zoncfg_grammar.tab.h"

33 /*
34 * This constant defines the number of entries added to unclaimed_tokens[]
35 * when it runs out of space.
36 */
37 #define UNCLAIMED_TOKENS_BUFFER_GROWTH 4

39 /*
40 * Invariants:
41 *
42 * unclaimed_tokens == NULL IFF unclaimed_tokens_size == 0
43 * unclaimed_tokens_size == 0 IFF num_unclaimed_tokens == 0
44 */
45 static char **unclaimed_tokens; /* TOKENs produced by Lex (see below) */
46 /* but not claimed by YACC reduction */
47 /* rules */
48 static unsigned int unclaimed_tokens_size; /* size of unclaimed_tokens */
49 static unsigned int num_unclaimed_tokens; /* number of unclaimed TOKENs */

51 int lex_lineno = 1; /* line number for error reporting */
52 static int state = INITIAL;
53 extern boolean_t cmd_file_mode;
54 extern boolean_t saw_error;
55 extern void yyerror(char *s);

57 static char *create_token(char *s);
58 %}

```

```

60 %a 7000
61 %p 5000
62 %e 2000
63 %n 1000

65 %{
66 /*
67 * The three states below are for tokens, lists and complex property values.
68 * Note that simple property values are a subset of tokens.
69 */
70 %}
71 %s TSTATE
72 %s LSTATE
73 %s CSTATE
74 %%

76 <INITIAL>"#[^\n]* { }

78 <INITIAL>add {
79 BEGIN TSTATE;
80 state = TSTATE;
81 return ADD;
82 }

84 <INITIAL>cancel {
85 BEGIN TSTATE;
86 state = TSTATE;
87 return CANCEL;
88 }

90 <INITIAL>commit {
91 BEGIN TSTATE;
92 state = TSTATE;
93 return COMMIT;
94 }

96 <INITIAL>create {
97 BEGIN TSTATE;
98 state = TSTATE;
99 return CREATE;
100 }

102 <INITIAL>delete {
103 BEGIN TSTATE;
104 state = TSTATE;
105 return DELETE;
106 }

108 <INITIAL>end {
109 BEGIN TSTATE;
110 state = TSTATE;
111 return END;
112 }

114 <INITIAL>exit {
115 BEGIN TSTATE;
116 state = TSTATE;
117 return EXIT;
118 }

120 <INITIAL>export {
121 BEGIN TSTATE;
122 state = TSTATE;
123 return EXPORT;
124 }

```

```

126 <INITIAL>"?"|help {
127     BEGIN TSTATE;
128     state = TSTATE;
129     return HELP;
130 }
132 <INITIAL>info {
133     BEGIN TSTATE;
134     state = TSTATE;
135     return INFO;
136 }
138 <INITIAL>remove {
139     BEGIN TSTATE;
140     state = TSTATE;
141     return REMOVE;
142 }
144 <INITIAL>revert {
145     BEGIN TSTATE;
146     state = TSTATE;
147     return REVERT;
148 }
150 <INITIAL>select {
151     BEGIN TSTATE;
152     state = TSTATE;
153     return SELECT;
154 }
156 <INITIAL>set {
157     BEGIN TSTATE;
158     state = TSTATE;
159     return SET;
160 }
162 <INITIAL>clear {
163     BEGIN TSTATE;
164     state = TSTATE;
165     return CLEAR;
166 }
168 <INITIAL>verify {
169     BEGIN TSTATE;
170     state = TSTATE;
171     return VERIFY;
172 }
174 <TSTATE>net { return NET; }
176 <TSTATE>fs { return FS; }
178 <TSTATE>device { return DEVICE; }
180 <TSTATE>rctl { return RCTL; }
182 <TSTATE>attr { return ATTR; }
184 <TSTATE>admin { return ADMIN; }
186 <TSTATE>security-flags { return SECFLAGS; }
188 #endif /* ! codereview */
189 <TSTATE>zonename { return ZONENAME; }
190 <CSTATE>zonename { return ZONENAME; }

```

```

192 <TSTATE>dataset { return DATASET; }
194 <TSTATE>dedicated-cpu { return PSET; }
196 <TSTATE>capped-cpu { return PCAP; }
198 <TSTATE>capped-memory { return MCAP; }
200 <TSTATE>zonepath { return ZONEPATH; }
201 <CSTATE>zonepath { return ZONEPATH; }
203 <TSTATE>brand { return BRAND; }
204 <CSTATE>brand { return BRAND; }
206 <TSTATE>autoboot { return AUTOBOOT; }
207 <CSTATE>autoboot { return AUTOBOOT; }
209 <TSTATE>ip-type { return IPTYPE; }
210 <CSTATE>ip-type { return IPTYPE; }
212 <TSTATE>pool { return POOL; }
213 <CSTATE>pool { return POOL; }
215 <TSTATE>limitpriv { return LIMITPRIV; }
216 <CSTATE>limitpriv { return LIMITPRIV; }
218 <TSTATE>bootargs { return BOOTARGS; }
219 <CSTATE>bootargs { return BOOTARGS; }
221 <TSTATE>type { return TYPE; }
222 <CSTATE>type { return TYPE; }
224 <TSTATE>value { return VALUE; }
225 <CSTATE>value { return VALUE; }
227 <TSTATE>options { return OPTIONS; }
228 <CSTATE>options { return OPTIONS; }
230 <TSTATE>allowed-address { return ALLOWED_ADDRESS; }
231 <CSTATE>allowed-address { return ALLOWED_ADDRESS; }
233 <TSTATE>address { return ADDRESS; }
234 <CSTATE>address { return ADDRESS; }
236 <TSTATE>physical { return PHYSICAL; }
237 <CSTATE>physical { return PHYSICAL; }
239 <TSTATE>defrouter { return DEFROUTER; }
240 <CSTATE>defrouter { return DEFROUTER; }
242 <TSTATE>dir { return DIR; }
243 <CSTATE>dir { return DIR; }
245 <TSTATE>special { return SPECIAL; }
246 <CSTATE>special { return SPECIAL; }
248 <TSTATE>raw { return RAW; }
249 <CSTATE>raw { return RAW; }
251 <TSTATE>name { return NAME; }
252 <CSTATE>name { return NAME; }
254 <TSTATE>match { return MATCH; }
255 <CSTATE>match { return MATCH; }

```

```

257 <TSTATE>priv { return PRIV; }
258 <CSTATE>priv { return PRIV; }

260 <TSTATE>limit { return LIMIT; }
261 <CSTATE>limit { return LIMIT; }

263 <TSTATE>action { return ACTION; }
264 <CSTATE>action { return ACTION; }

266 <TSTATE>ncpus { return NCPUS; }
267 <CSTATE>ncpus { return NCPUS; }

269 <TSTATE>locked { return LOCKED; }
270 <CSTATE>locked { return LOCKED; }

272 <TSTATE>swap { return SWAP; }
273 <CSTATE>swap { return SWAP; }

275 <TSTATE>importance { return IMPORTANCE; }
276 <CSTATE>importance { return IMPORTANCE; }

278 <TSTATE>cpu-shares { return SHARES; }
279 <CSTATE>cpu-shares { return SHARES; }

281 <TSTATE>max-lwps { return MAXLWPS; }
282 <CSTATE>max-lwps { return MAXLWPS; }

284 <TSTATE>max-processes { return MAXPROCS; }
285 <CSTATE>max-processes { return MAXPROCS; }

287 <TSTATE>max-shm-memory { return MAXSHMMEM; }
288 <CSTATE>max-shm-memory { return MAXSHMMEM; }

290 <TSTATE>max-shm-ids { return MAXSHMIDS; }
291 <CSTATE>max-shm-ids { return MAXSHMIDS; }

293 <TSTATE>max-msg-ids { return MAXMSGIDS; }
294 <CSTATE>max-msg-ids { return MAXMSGIDS; }

296 <TSTATE>max-sem-ids { return MAXSEMIDS; }
297 <CSTATE>max-sem-ids { return MAXSEMIDS; }

299 <TSTATE>scheduling-class { return SCHED; }
300 <CSTATE>scheduling-class { return SCHED; }

302 <TSTATE>hostid { return HOSTID; }
303 <CSTATE>hostid { return HOSTID; }

305 <TSTATE>user { return USER; }
306 <CSTATE>user { return USER; }

308 <TSTATE>auths { return AUTHS; }
309 <CSTATE>auths { return AUTHS; }

311 <TSTATE>fs-allowed { return FS_ALLOWED; }
312 <CSTATE>fs-allowed { return FS_ALLOWED; }

314 <TSTATE>default { return DEFAULT; }
315 <CSTATE>default { return DEFAULT; }

317 <TSTATE>lower { return LOWER; }
318 <CSTATE>lower { return LOWER; }

320 <TSTATE>upper { return UPPER; }
321 <CSTATE>upper { return UPPER; }

```

```

323 #endif /* ! codereview */
324 <TSTATE>= { return EQUAL; }
325 <LSTATE>= { return EQUAL; }
326 <CSTATE>= { return EQUAL; }

328 <TSTATE> "[" {
329             BEGIN LSTATE;
330             state = LSTATE;
331             return OPEN_SQ_BRACKET;
332 }

334 <LSTATE> "]" {
335             BEGIN TSTATE;
336             state = TSTATE;
337             return CLOSE_SQ_BRACKET;
338 }

340 <TSTATE> "(" {
341             BEGIN CSTATE;
342             return OPEN_PAREN;
343 }

345 <LSTATE> "(" {
346             BEGIN CSTATE;
347             return OPEN_PAREN;
348 }

350 <CSTATE> ")" {
351             BEGIN state;
352             return CLOSE_PAREN;
353 }

355 <LSTATE> "," { return COMMA; }
356 <CSTATE> "," { return COMMA; }

358 <TSTATE> [^ \t\n";=\\[\(\)]+ {
359             yyival.strval = create_token(yytext);
360             return TOKEN;
361 }

363 <LSTATE> [^ \t\n";=\\[\(\)]+ {
364             yyival.strval = create_token(yytext);
365             return TOKEN;
366 }

368 <CSTATE> [^ \t\n";=\\[\(\)]+ {
369             yyival.strval = create_token(yytext);
370             return TOKEN;
371 }

373 <TSTATE> \"[^\n]*[\\n] {
374             yyival.strval = create_token(yytext + 1);
375             if (yyival.strval[yy leng - 2] == '\"')
376                 yyival.strval[yy leng - 2] = 0;
377             return TOKEN;
378 }

380 <LSTATE> \"[^\n]*[\\n] {
381             yyival.strval = create_token(yytext + 1);
382             if (yyival.strval[yy leng - 2] == '\"')
383                 yyival.strval[yy leng - 2] = 0;
384             return TOKEN;
385 }

387 ";" {
388             BEGIN INITIAL;

```

```

389         return (yytext[0]);
390     }
392 \n     {
393         lex_lineno++;
394         BEGIN INITIAL;
395         return (yytext[0]);
396     }
398 [ \t] ; /* Ignore whitespace */
400 .     {
401         return (yytext[0]);
402     }
404 %%
406 /*
407  * Assert that there are no unclaimed tokens. This function enforces the
408  * invariants mentioned at the top of this file.
409  */
410 void
411 assert_no_unclaimed_tokens(void)
412 {
413     assert(num_unclaimed_tokens == 0);
414     assert(unclaimed_tokens == NULL);
415     assert(unclaimed_tokens_size == 0);
416 }
418 /*
419  * Claim the specified unclaimed TOKEN. YACC reduction rules that
420  * use TOKENS should invoke this function immediately before freeing the TOKENS
421  * or adding them to data structures that will be cleaned up when the YACC
422  * parser finishes or encounters errors. Reduction rules should only claim the
423  * TOKENS that they use.
424  *
425  * This function returns its argument but does not free its memory.
426  */
427 char *
428 claim_token(char *token)
429 {
430     unsigned int index;
432     /*
433      * Find the token in the list of unclaimed tokens.
434      */
435     assert(num_unclaimed_tokens > 0);
436     for (index = 0; index < num_unclaimed_tokens; index++) {
437         if (unclaimed_tokens[index] == token)
438             break;
439     }
441     /*
442      * Abort if we didn't find the token.
443      */
444     assert(index != num_unclaimed_tokens);
446     /*
447      * Replace the token with the last unclaimed token.
448      */
449     num_unclaimed_tokens--;
450     unclaimed_tokens[index] = unclaimed_tokens[num_unclaimed_tokens];
452     /*
453      * Delete the list of unclaimed tokens if it's empty.
454      */

```

```

455     if (num_unclaimed_tokens == 0) {
456         free(unclaimed_tokens);
457         unclaimed_tokens = NULL;
458         unclaimed_tokens_size = 0;
459     }
461     return (token);
462 }
464 /*
465  * Free all unclaimed TOKENS. This should only be invoked when the YACC
466  * parser encounters errors.
467  */
468 static void
469 free_tokens(void)
470 {
471     if (unclaimed_tokens != NULL) {
472         while (num_unclaimed_tokens > 0)
473             free(unclaimed_tokens[--num_unclaimed_tokens]);
474         free(unclaimed_tokens);
475         unclaimed_tokens = NULL;
476         unclaimed_tokens_size = 0;
477     }
478     assert_no_unclaimed_tokens();
479 }
481 /*
482  * Create a TOKEN from the specified string. The TOKEN is merely a duplicate
483  * of the specified string. TOKENS must be claimed by the YACC reduction rules
484  * that use them; see claim_token() above.
485  */
486 char *
487 create_token(char *s)
488 {
489     char *result;
491     if ((result = strdup(s)) == NULL) {
492         yyerror("Out of memory");
493         exit(Z_ERR);
494     }
496     /*
497      * Add the new TOKEN to the list of unclaimed TOKENS. The list might
498      * have to be resized.
499      *
500      * Reduction rules should claim TOKENS via claim_token() (see above).
501      */
502     if (num_unclaimed_tokens == unclaimed_tokens_size) {
503         char **new_unclaimed_tokens;
505         unclaimed_tokens_size += UNCLAIMED_TOKENS_BUFFER_GROWTH;
506         new_unclaimed_tokens = (char **)realloc(unclaimed_tokens,
507         unclaimed_tokens_size * sizeof(char *));
508         if (new_unclaimed_tokens == NULL) {
509             yyerror("Out of memory");
510             free(result);
511             exit(Z_ERR);
512         }
513         unclaimed_tokens = new_unclaimed_tokens;
514     }
515     unclaimed_tokens[num_unclaimed_tokens] = result;
516     num_unclaimed_tokens++;
517     return (result);
518 }
520 void

```

```
521 yyerror(char *s)
522 {
523     /*
524      * Ensure that we won't leak unclaimed tokens.
525      */
526     free_tokens();

528     /* feof(yyin) is not an error; anything else is, so we set saw_error */
529     if (yytext[0] == '\0') {
530         if (!feof(yyin)) {
531             saw_error = B_TRUE;
532             (void) fprintf(stderr, gettext("%s, token expected\n"),
533                          s);
534         }
535         return;
536     }

538     saw_error = B_TRUE;
539     if (cmd_file_mode)
540         (void) fprintf(stderr, gettext("%s on line %d at '%s'\n"), s,
541                      lex_lineno, (yytext[0] == '\n') ? "\\n" : yytext);
542     else
543         (void) fprintf(stderr, gettext("%s at '%s'\n"), s,
544                      (yytext[0] == '\n') ? "\\n" : yytext);
545     usage(B_FALSE, HELP_SUBCMDS);
546 }
```

```

*****
4812 Thu Jun 30 22:01:54 2016
new/usr/src/common/secflags/secflags.c
Code review comments from jeffpc
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
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 /* Copyright 2015, Richard Lowe. */

14 #include <sys/secflags.h>
15 #include <sys/types.h>

17 #if defined(_KERNEL)
18 #include <sys/kmem.h>
19 #include <sys/sunddi.h>
20 #else
21 #include "lint.h"          /* libc header */
22 #include <stdio.h>
23 #include <stdlib.h>
24 #include <strings.h>
25 #endif

27 secflagset_t
28 secflag_to_bit(secflag_t secflag)
29 {
30     return (1 << secflag);
31 }

33 boolean_t
34 secflag_isset(secflagset_t flags, secflag_t sf)
35 {
36     return ((flags & secflag_to_bit(sf)) != 0);
37 }

39 void
40 secflag_clear(secflagset_t *flags, secflag_t sf)
41 {
42     *flags &= ~secflag_to_bit(sf);
43 }

45 void
46 secflag_set(secflagset_t *flags, secflag_t sf)
47 {
48     *flags |= secflag_to_bit(sf);
49 }

51 boolean_t
52 secflags_isempty(secflagset_t flags)
53 {
54     return (flags == 0);
55 }

57 void

```

```

58 secflags_zero(secflagset_t *flags)
59 {
60     *flags = 0;
61 }

63 void
64 secflags_fullset(secflagset_t *flags)
65 {
66     *flags = PROC_SEC_MASK;
67 }

69 void
70 secflags_copy(secflagset_t *dst, const secflagset_t *src)
71 {
72     *dst = *src;
73 }

75 boolean_t
76 secflags_issubset(secflagset_t a, secflagset_t b)
77 {
78     return (!(a & ~b));
79 }

81 boolean_t
82 secflags_issuperset(secflagset_t a, secflagset_t b)
83 {
84     return (secflags_issubset(b, a));
85 }

87 boolean_t
88 secflags_intersection(secflagset_t a, secflagset_t b)
89 {
90     return (a & b);
91 }

93 void
94 secflags_union(secflagset_t *a, const secflagset_t *b)
95 {
96     *a |= *b;
97 }

99 void
100 secflags_difference(secflagset_t *a, const secflagset_t *b)
101 {
102     *a &= ~*b;
103 }

105 boolean_t
106 psecflags_validate_delta(const psecflags_t *sf, const secflagdelta_t *delta)
107 {
108     if (delta->psd_ass_active) {
109         /*
110          * If there's a bit in lower not in args, or a bit args not in
111          * upper
112          */
113         if (!secflags_issubset(delta->psd_assign, sf->psf_upper) ||
114             !secflags_issuperset(delta->psd_assign, sf->psf_lower)) {
115             return (B_FALSE);
116         }
117     }

118     if (!secflags_issubset(delta->psd_assign, PROC_SEC_MASK))
119         return (B_FALSE);
120 } else {
121     /* If we're adding a bit not in upper */
122     if (!secflags_isempty(delta->psd_add)) {
123         if (!secflags_issubset(delta->psd_add, sf->psf_upper)) {

```

```

124         return (B_FALSE);
125     }
126 }

128 /* If we're removing a bit that's in lower */
129 if (!secflags_isempty(delta->psd_rem) {
130     if (secflags_intersection(delta->psd_rem,
131         sf->psf_lower) {
132         return (B_FALSE);
133     }
134 }

136 if (!secflags_issubset(delta->psd_add, PROC_SEC_MASK) ||
137     !secflags_issubset(delta->psd_rem, PROC_SEC_MASK))
138     return (B_FALSE);
139 }

141 return (B_TRUE);
142 }

144 boolean_t
145 psecflags_validate(const psecflags_t *sf)
146 {
147     if (!secflags_issubset(sf->psf_lower, PROC_SEC_MASK) ||
148         !secflags_issubset(sf->psf_inherit, PROC_SEC_MASK) ||
149         !secflags_issubset(sf->psf_effective, PROC_SEC_MASK) ||
150         !secflags_issubset(sf->psf_upper, PROC_SEC_MASK))
151         return (B_FALSE);

153 if (!secflags_issubset(sf->psf_lower, sf->psf_inherit))
154     return (B_FALSE);
155 if (!secflags_issubset(sf->psf_lower, sf->psf_upper))
156     return (B_FALSE);
157 if (!secflags_issubset(sf->psf_inherit, sf->psf_upper))
158     return (B_FALSE);

160 return (B_TRUE);
161 }

163 void
164 psecflags_default(psecflags_t *sf)
165 {
166     secflags_zero(&sf->psf_effective);
167     secflags_zero(&sf->psf_inherit);
168     secflags_zero(&sf->psf_lower);
169     secflags_fullset(&sf->psf_upper);
170 }

172 static struct flagdesc {
173     secflag_t value;
174     const char *name;
175 } flagdescs[] = {
176     { PROC_SEC_ASLR, "aslr" },
177     { PROC_SEC_FORBIDNULLMAP, "forbidnullmap" },
178     { PROC_SEC_NOEXECSTACK, "noexecstack" },
179     { 0x0, NULL }
180 };

182 boolean_t
183 secflag_by_name(const char *str, secflag_t *ret)
184 {
185     struct flagdesc *fd;

187     for (fd = flagdescs; fd->name != NULL; fd++) {
188         if (strcasecmp(str, fd->name) == 0) {
189             *ret = fd->value;

```

```

190         return (B_TRUE);
191     }
192 }

194 return (B_FALSE);
195 }

197 const char *
198 secflag_to_str(secflag_t sf)
199 {
200     struct flagdesc *fd;

202     for (fd = flagdescs; fd->name != NULL; fd++) {
203         if (sf == fd->value)
204             return (fd->name);
205     }

207     return (NULL);
208 }

210 void
211 secflags_to_str(secflagset_t flags, char *buf, size_t buflen)
212 {
213     struct flagdesc *fd;

215     if (buflen >= 1)
216         buf[0] = '\0';

218     if (flags == 0) {
219         (void) strcpy(buf, "none", buflen);
220         return;
221     }

223     for (fd = flagdescs; fd->name != NULL; fd++) {
224         if (secflag_isset(flags, fd->value)) {
225             if (buf[0] != '\0')
226                 (void) strcat(buf, ",", buflen);
227             (void) strcat(buf, fd->name, buflen);
228         }

230         secflag_clear(&flags, fd->value);
231     }

233     if (flags != 0) { /* unknown flags */
234         char hexbuf[19]; /* 0x%16 PRIx64 */

236         (void) snprintf(hexbuf, sizeof (hexbuf), "0x%16" PRIx64, flags);
237         if (buf[0] != '\0')
238             (void) strcat(buf, ",", buflen);
239         (void) strcat(buf, hexbuf, buflen);
240     }
241 }
242 #endif /* !codereview */

```



```

*****
22867 Thu Jun 30 22:01:55 2016
new/usr/src/head/libzonecfg.h
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
_____unchanged_portion_omitted_____

255 #define ZONECFG_SECFLAGS_MAX    1024
256 struct zone_secflagstab {
257     char zone_secflags_lower[ZONECFG_SECFLAGS_MAX];
258     char zone_secflags_upper[ZONECFG_SECFLAGS_MAX];
259     char zone_secflags_default[ZONECFG_SECFLAGS_MAX];
260 };

262 #endif /* ! codereview */
263 typedef struct zone_userauths {
264     char          user[MAXUSERNAME];
265     char          zonename[ZONENAME_MAX];
266     struct zone_userauths *next;
267 } zone_userauths_t;

269 typedef struct {
270     uu_avl_node_t    zpe_entry;
271     char             *zpe_name;
272     char             *zpe_vers;
273 } zone_pkg_entry_t;

275 typedef enum zone_iptype {
276     ZS_SHARED,
277     ZS_EXCLUSIVE
278 } zone_iptype_t;

280 /*
281  * Basic configuration management routines.
282  */
283 extern zone_dochandle_t    zonecfg_init_handle(void);
284 extern int                 zonecfg_get_handle(const char *, zone_dochandle_t);
285 extern int                 zonecfg_get_snapshot_handle(const char *, zone_dochandle_t);
286 extern int                 zonecfg_get_template_handle(const char *, const char *,
287     zone_dochandle_t);
288 extern int                 zonecfg_get_xml_handle(const char *, zone_dochandle_t);
289 extern int                 zonecfg_check_handle(zone_dochandle_t);
290 extern void                zonecfg_fini_handle(zone_dochandle_t);
291 extern int                 zonecfg_destroy(const char *, boolean_t);
292 extern int                 zonecfg_destroy_snapshot(const char *);
293 extern int                 zonecfg_save(zone_dochandle_t);
294 extern int                 zonecfg_create_snapshot(const char *);
295 extern char                *zonecfg_strerror(int);
296 extern int                 zonecfg_access(const char *, int);
297 extern void                zonecfg_set_root(const char *);
298 extern const char *zonecfg_get_root(void);
299 extern boolean_t zonecfg_in_alt_root(void);
300 extern int                 zonecfg_num_resources(zone_dochandle_t, char *);
301 extern int                 zonecfg_del_all_resources(zone_dochandle_t, char *);
302 extern boolean_t zonecfg_valid_ncpus(char *, char *);
303 extern boolean_t zonecfg_valid_importance(char *);
304 extern int                 zonecfg_str_to_bytes(char *, uint64_t *);
305 extern boolean_t zonecfg_valid_memlimit(char *, uint64_t *);
306 extern boolean_t zonecfg_valid_alias_limit(char *, char *, uint64_t *);

308 /*
309  * Zone name, path to zone directory, autoboot setting, pool, boot
310  * arguments, and scheduling-class.

```

```

311 */
312 extern int                 zonecfg_validate_zonename(const char *);
313 extern int                 zonecfg_get_name(zone_dochandle_t, char *, size_t);
314 extern int                 zonecfg_set_name(zone_dochandle_t, char *);
315 extern int                 zonecfg_get_zonepath(zone_dochandle_t, char *, size_t);
316 extern int                 zonecfg_set_zonepath(zone_dochandle_t, char *);
317 extern int                 zonecfg_get_autoboot(zone_dochandle_t, boolean_t *);
318 extern int                 zonecfg_set_autoboot(zone_dochandle_t, boolean_t);
319 extern int                 zonecfg_get_iptype(zone_dochandle_t, zone_iptype_t *);
320 extern int                 zonecfg_set_iptype(zone_dochandle_t, zone_iptype_t);
321 extern int                 zonecfg_get_pool(zone_dochandle_t, char *, size_t);
322 extern int                 zonecfg_set_pool(zone_dochandle_t, char *);
323 extern int                 zonecfg_get_bootargs(zone_dochandle_t, char *, size_t);
324 extern int                 zonecfg_set_bootargs(zone_dochandle_t, char *);
325 extern int                 zonecfg_get_sched_class(zone_dochandle_t, char *, size_t);
326 extern int                 zonecfg_set_sched(zone_dochandle_t, char *);
327 extern int                 zonecfg_get_dflt_sched_class(zone_dochandle_t, char *, int);

329 /*
330  * Set/retrieve the brand for the zone
331  */
332 extern int                 zonecfg_get_brand(zone_dochandle_t, char *, size_t);
333 extern int                 zonecfg_set_brand(zone_dochandle_t, char *);

335 /*
336  * Filesystem configuration.
337  */
338 extern int                 zonecfg_add_filesystem(zone_dochandle_t, struct zone_fstab *);
339 extern int                 zonecfg_delete_filesystem(zone_dochandle_t,
340     struct zone_fstab *);
341 extern int                 zonecfg_modify_filesystem(zone_dochandle_t,
342     struct zone_fstab *, struct zone_fstab *);
343 extern int                 zonecfg_lookup_filesystem(zone_dochandle_t,
344     struct zone_fstab *);
345 extern int                 zonecfg_add_fs_option(struct zone_fstab *, char *);
346 extern int                 zonecfg_remove_fs_option(struct zone_fstab *, char *);
347 extern void                zonecfg_free_fs_option_list(zone_fsopt_t *);
348 extern int                 zonecfg_find_mounts(char *, int(*) (const struct mnttab *,
349     void *), void *);

351 /*
352  * Network interface configuration.
353  */
354 extern int                 zonecfg_add_nwif(zone_dochandle_t, struct zone_nwifstab *);
355 extern int                 zonecfg_delete_nwif(zone_dochandle_t, struct zone_nwifstab *);
356 extern int                 zonecfg_modify_nwif(zone_dochandle_t, struct zone_nwifstab *,
357     struct zone_nwifstab *);
358 extern int                 zonecfg_lookup_nwif(zone_dochandle_t, struct zone_nwifstab *);

360 /*
361  * Hostid emulation configuration.
362  */
363 extern int                 zonecfg_get_hostid(zone_dochandle_t, char *, size_t);
364 extern int                 zonecfg_set_hostid(zone_dochandle_t, const char *);

366 /*
367  * Allowed FS mounts configuration.
368  */
369 extern int                 zonecfg_get_fs_allowed(zone_dochandle_t, char *, size_t);
370 extern int                 zonecfg_set_fs_allowed(zone_dochandle_t, const char *);

372 /*
373  * Device configuration and rule matching.
374  */
375 extern int                 zonecfg_add_dev(zone_dochandle_t, struct zone_devtab *);
376 extern int                 zonecfg_delete_dev(zone_dochandle_t, struct zone_devtab *);

```

```

377 extern int zonecfg_modify_dev(zone_dochandle_t, struct zone_devtab *,
378     struct zone_devtab *);
379 extern int zonecfg_lookup_dev(zone_dochandle_t, struct zone_devtab *);

381 /*
382  * Resource control configuration.
383  */
384 extern int zonecfg_add_rctl(zone_dochandle_t, struct zone_rctltab *);
385 extern int zonecfg_delete_rctl(zone_dochandle_t, struct zone_rctltab *);
386 extern int zonecfg_modify_rctl(zone_dochandle_t, struct zone_rctltab *,
387     struct zone_rctltab *);
388 extern int zonecfg_lookup_rctl(zone_dochandle_t, struct zone_rctltab *);
389 extern int zonecfg_add_rctl_value(struct zone_rctltab *,
390     struct zone_rctlvaltab *);
391 extern int zonecfg_remove_rctl_value(struct zone_rctltab *,
392     struct zone_rctlvaltab *);
393 extern void zonecfg_free_rctl_value_list(struct zone_rctlvaltab *);
394 extern boolean_t zonecfg_aliased_rctl_ok(zone_dochandle_t, char *);
395 extern int zonecfg_set_aliased_rctl(zone_dochandle_t, char *, uint64_t);
396 extern int zonecfg_get_aliased_rctl(zone_dochandle_t, char *, uint64_t *);
397 extern int zonecfg_rm_aliased_rctl(zone_dochandle_t, char *);
398 extern int zonecfg_apply_rctls(char *, zone_dochandle_t);

400 /*
401  * Generic attribute configuration and type/value extraction.
402  */
403 extern int zonecfg_add_attr(zone_dochandle_t, struct zone_attrtab *);
404 extern int zonecfg_delete_attr(zone_dochandle_t, struct zone_attrtab *);
405 extern int zonecfg_modify_attr(zone_dochandle_t, struct zone_attrtab *,
406     struct zone_attrtab *);
407 extern int zonecfg_lookup_attr(zone_dochandle_t, struct zone_attrtab *);
408 extern int zonecfg_get_attr_boolean(const struct zone_attrtab *,
409     boolean_t *);
410 extern int zonecfg_get_attr_int(const struct zone_attrtab *, int64_t *);
411 extern int zonecfg_get_attr_string(const struct zone_attrtab *, char *,
412     size_t);
413 extern int zonecfg_get_attr_uint(const struct zone_attrtab *, uint64_t *);

415 /*
416  * ZFS configuration.
417  */
418 extern int zonecfg_add_ds(zone_dochandle_t, struct zone_dstab *);
419 extern int zonecfg_delete_ds(zone_dochandle_t, struct zone_dstab *);
420 extern int zonecfg_modify_ds(zone_dochandle_t, struct zone_dstab *,
421     struct zone_dstab *);
422 extern int zonecfg_lookup_ds(zone_dochandle_t, struct zone_dstab *);

424 /*
425  * cpu-set configuration.
426  */
427 extern int zonecfg_add_pset(zone_dochandle_t, struct zone_psettab *);
428 extern int zonecfg_delete_pset(zone_dochandle_t);
429 extern int zonecfg_modify_pset(zone_dochandle_t, struct zone_psettab *);
430 extern int zonecfg_lookup_pset(zone_dochandle_t, struct zone_psettab *);

432 /*
433  * mem-cap configuration.
434  */
435 extern int zonecfg_delete_mcap(zone_dochandle_t);
436 extern int zonecfg_modify_mcap(zone_dochandle_t, struct zone_mcaptab *);
437 extern int zonecfg_lookup_mcap(zone_dochandle_t, struct zone_mcaptab *);

439 /* security-flags configuration */
440 extern int zonecfg_add_secflags(zone_dochandle_t,
441     struct zone_secflagstab *);
442 extern int zonecfg_delete_secflags(zone_dochandle_t,

```

```

443     struct zone_secflagstab *);
444 extern int zonecfg_modify_secflags(zone_dochandle_t,
445     struct zone_secflagstab *, struct zone_secflagstab *);
446 extern int zonecfg_lookup_secflags(zone_dochandle_t,
447     struct zone_secflagstab *);

449 #endif /* ! codereview */
450 /*
451  * Temporary pool support functions.
452  */
453 extern int zonecfg_destroy_tmp_pool(char *, char *, int);
454 extern int zonecfg_bind_tmp_pool(zone_dochandle_t, zoneid_t, char *, int);
455 extern int zonecfg_bind_pool(zone_dochandle_t, zoneid_t, char *, int);
456 extern boolean_t zonecfg_warn_poold(zone_dochandle_t);
457 extern int zonecfg_get_poolname(zone_dochandle_t, char *, char *, size_t);

459 /*
460  * Miscellaneous utility functions.
461  */
462 extern int zonecfg_enable_rcapd(char *, int);

464 /*
465  * attach/detach support.
466  */
467 extern int zonecfg_get_attach_handle(const char *, const char *,
468     const char *, boolean_t, zone_dochandle_t);
469 extern int zonecfg_attach_manifest(int, zone_dochandle_t,
470     zone_dochandle_t);
471 extern int zonecfg_detach_save(zone_dochandle_t, uint_t);
472 extern boolean_t zonecfg_detached(const char *);
473 extern void zonecfg_rm_detached(zone_dochandle_t, boolean_t forced);
474 extern int zonecfg_dev_manifest(zone_dochandle_t);
475 extern int zonecfg_devperms_apply(zone_dochandle_t, const char *,
476     uid_t, gid_t, mode_t, const char *);
477 extern void zonecfg_set_swinv(zone_dochandle_t);
478 extern int zonecfg_add_pkg(zone_dochandle_t, char *, char *);

480 /*
481  * External zone verification support.
482  */
483 extern int zonecfg_verify_save(zone_dochandle_t, char *);

485 /*
486  * 'ent' iterator routines.
487  */
488 extern int zonecfg_setfsent(zone_dochandle_t);
489 extern int zonecfg_getfsent(zone_dochandle_t, struct zone_fstab *);
490 extern int zonecfg_endfsent(zone_dochandle_t);
491 extern int zonecfg_setnwifent(zone_dochandle_t);
492 extern int zonecfg_getnwifent(zone_dochandle_t, struct zone_nwifstab *);
493 extern int zonecfg_endnwifent(zone_dochandle_t);
494 extern int zonecfg_setdevent(zone_dochandle_t);
495 extern int zonecfg_getdevent(zone_dochandle_t, struct zone_devtab *);
496 extern int zonecfg_enddevent(zone_dochandle_t);
497 extern int zonecfg_setattrent(zone_dochandle_t);
498 extern int zonecfg_getattrent(zone_dochandle_t, struct zone_attrtab *);
499 extern int zonecfg_endattrent(zone_dochandle_t);
500 extern int zonecfg_setrctlent(zone_dochandle_t);
501 extern int zonecfg_getrctlent(zone_dochandle_t, struct zone_rctltab *);
502 extern int zonecfg_endrctlent(zone_dochandle_t);
503 extern int zonecfg_setdsent(zone_dochandle_t);
504 extern int zonecfg_getdsent(zone_dochandle_t, struct zone_dstab *);
505 extern int zonecfg_enddsent(zone_dochandle_t);
506 extern int zonecfg_getpsetent(zone_dochandle_t, struct zone_psettab *);
507 extern int zonecfg_getmcapent(zone_dochandle_t, struct zone_mcaptab *);
508 extern int zonecfg_getpkgdata(zone_dochandle_t, uu_avl_pool_t *,

```

```

509     uu_avl_t *);
510 extern int zonecfg_setdevperment(zone_dochandle_t);
511 extern int zonecfg_getdevperment(zone_dochandle_t,
512     struct zone_devpermtab *);
513 extern int zonecfg_enddevperment(zone_dochandle_t);
514 extern int zonecfg_setadminent(zone_dochandle_t);
515 extern int zonecfg_getadminent(zone_dochandle_t, struct zone_admintab *);
516 extern int zonecfg_endadminent(zone_dochandle_t);
517 extern int zonecfg_getsecflagstabsent(zone_dochandle_t,
518     struct zone_secflagstab *);
519 #endif /* !codereview */

521 /*
522  * Privilege-related functions.
523  */
524 extern int zonecfg_default_privset(priv_set_t *, const char *);
525 extern int zonecfg_get_privset(zone_dochandle_t, priv_set_t *,
526     char **);
527 extern int zonecfg_get_limitpriv(zone_dochandle_t, char **);
528 extern int zonecfg_set_limitpriv(zone_dochandle_t, char *);

530 /*
531  * Higher-level routines.
532  */
533 extern int zone_get_brand(char *, char *, size_t);
534 extern int zone_get_rootpath(char *, char *, size_t);
535 extern int zone_get_devroot(char *, char *, size_t);
536 extern int zone_get_zonepath(char *, char *, size_t);
537 extern int zone_get_state(char *, zone_state_t *);
538 extern int zone_set_state(char *, zone_state_t);
539 extern char *zone_state_str(zone_state_t);
540 extern int zonecfg_get_name_by_uuid(const uuid_t, char *, size_t);
541 extern int zonecfg_get_uuid(const char *, uuid_t);
542 extern int zonecfg_default_brand(char *, size_t);

544 /*
545  * Iterator for configured zones.
546  */
547 extern FILE *setzoneent(void);
548 extern char *getzoneent(FILE *);
549 extern struct zoneent *getzoneent_private(FILE *);
550 extern void endzoneent(FILE *);

552 /*
553  * File-system-related convenience functions.
554  */
555 extern boolean_t zonecfg_valid_fs_type(const char *);

557 /*
558  * Network-related convenience functions.
559  */
560 extern boolean_t zonecfg_same_net_address(char *, char *);
561 extern int zonecfg_valid_net_address(char *, struct lifreq *);
562 extern boolean_t zonecfg_ifname_exists(sa_family_t, char *);

564 /*
565  * Rctl-related common functions.
566  */
567 extern boolean_t zonecfg_is_rctl(const char *);
568 extern boolean_t zonecfg_valid_rctlname(const char *);
569 extern boolean_t zonecfg_valid_rctlblk(const rctlblk_t *);
570 extern boolean_t zonecfg_valid_rctl(const char *, const rctlblk_t *);
571 extern int zonecfg_construct_rctlblk(const struct zone_rctlvaltab *,
572     rctlblk_t *);

574 /*

```

```

575  * Live Upgrade support functions. Shared between ON and install gate.
576  */
577 extern FILE *zonecfg_open_scratch(const char *, boolean_t);
578 extern int zonecfg_lock_scratch(FILE *);
579 extern void zonecfg_close_scratch(FILE *);
580 extern int zonecfg_get_scratch(FILE *, char *, size_t, char *, size_t, char *,
581     size_t);
582 extern int zonecfg_find_scratch(FILE *, const char *, const char *, char *,
583     size_t);
584 extern int zonecfg_reverse_scratch(FILE *, const char *, char *, size_t,
585     char *, size_t);
586 extern int zonecfg_add_scratch(FILE *, const char *, const char *,
587     const char *);
588 extern int zonecfg_delete_scratch(FILE *, const char *);
589 extern boolean_t zonecfg_is_scratch(const char *);

591 /*
592  * zoneadm support functions. Shared between zoneadm and brand hook code.
593  */
594 extern void zonecfg_init_lock_file(const char *, char **);
595 extern void zonecfg_release_lock_file(const char *, int);
596 extern int zonecfg_grab_lock_file(const char *, int *);
597 extern boolean_t zonecfg_lock_file_held(int *);
598 extern int zonecfg_ping_zoneadm(const char *);
599 extern int zonecfg_call_zoneadm(const char *, zone_cmd_arg_t *, char *,
600     boolean_t);
601 extern int zonecfg_insert_userauths(zone_dochandle_t, char *, char *);
602 extern int zonecfg_remove_userauths(zone_dochandle_t, char *, char *,
603     boolean_t);
604 extern int zonecfg_add_admin(zone_dochandle_t, struct zone_admintab *,
605     char *);
606 extern int zonecfg_delete_admin(zone_dochandle_t,
607     struct zone_admintab *, char *);
608 extern int zonecfg_modify_admin(zone_dochandle_t, struct zone_admintab *,
609     struct zone_admintab *, char *);
610 extern int zonecfg_delete_admins(zone_dochandle_t, char *);
611 extern int zonecfg_lookup_admin(zone_dochandle_t, struct zone_admintab *);
612 extern int zonecfg_authorize_users(zone_dochandle_t, char *);
613 extern int zonecfg_update_userauths(zone_dochandle_t, char *);
614 extern int zonecfg_deauthorize_user(zone_dochandle_t, char *, char *);
615 extern int zonecfg_deauthorize_users(zone_dochandle_t, char *);
616 extern boolean_t zonecfg_valid_auths(const char *, const char *);

618 #ifdef __cplusplus
619 }
620 #endif

622 #endif /* _LIBZONECFG_H */

```

```
*****
28482 Thu Jun 30 22:01:56 2016
new/usr/src/lib/auditd_plugins/syslog/systoken.c
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
_____unchanged_portion_omitted_____
```

```
1474 /*
1475 * -----
1476 * secflags_token() : Process secflags token and display contents
1477 *
1478 * Format of privilege token:
1479 *   secflags token id      adr_char
1480 *   secflag set name      adr_string
1481 *   secflags               adr_string
1482 * -----
1483 */
1484 int
1485 secflags_token(parse_context_t *ctx)
1486 {
1487     skip_bytes(ctx);
1488     skip_bytes(ctx);
1490     return (0);
1491 }
1492 #endif /* ! codereview */

1494 /*
1495 * Format of label token:
1496 *   label ID                1 byte
1497 *   compartment length      1 byte
1498 *   classification          2 bytes
1499 *   compartment words       <compartment length> * 4 bytes
1500 */
1501 int
1502 label_token(parse_context_t *ctx)
1503 {
1504     char c;

1506     ctx->adr.adr_now += sizeof (char); /* label ID */
1507     adrm_char(&(ctx->adr), &c, 1);

1509     ctx->adr.adr_now += sizeof (ushort_t); /* classification */
1510     ctx->adr.adr_now += 4 * c; /* compartments */

1512     return (0);
1513 }

1515 /*
1516 * Format of useofpriv token:
1517 *   priv_type                adr_char
1518 *   priv_set_t               adr_short
1519 *   priv_set                  adr_char*(sizeof (priv_set_t))
1520 */
1521 int
1522 useofpriv_token(parse_context_t *ctx)
1523 {
1524     ctx->adr.adr_now += sizeof (char); /* success / fail */
1525     skip_bytes(ctx);

1527     return (0);
1528 }
```

```

*****
5166 Thu Jun 30 22:01:57 2016
new/usr/src/lib/auditd_plugins/syslog/systoken.h
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
    unchanged portion omitted
47 typedef struct parse_context parse_context_t;

49 #define AU_TEXT_NAME      " text "

51 #ifdef useless
52 /*
53  * the following *_ar_token() functions parallel the *_token()
54  * functions defined in praudit/toktable.h
55  */

57 /*
58  * These tokens are the same for all versions of Solaris
59  */

61 /*
62  * Control tokens
63  */

65 extern void      file_token(adr_t *, uint64_t, uint64_t);
66 extern void      trailer_token(adr_t *, parse_context_t *);
67 extern void      header_token(adr_t *, parse_context_t *);
68 extern void      header32_ex_token(adr_t *, parse_context_t *);

70 /*
71  * Data tokens
72  */

74 extern void      arbitrary_data_token(adr_t *, parse_context_t *);
75 extern void      fmri_token(adr_t *, parse_context_t *);
76 extern void      s5_IPC_token(adr_t *, parse_context_t *);
77 extern void      path_token(adr_t *, parse_context_t *);
78 extern void      subject32_token();
79 extern void      process32_token();
80 extern void      return_value32_token();
81 extern void      text_token(adr_t *, parse_context_t *);
82 extern void      opaque_token(adr_t *, parse_context_t *);
83 extern void      ip_addr_token();
84 extern void      ip_token(adr_t *, parse_context_t *);
85 extern void      ipport_token(adr_t *, parse_context_t *);
86 extern void      argument32_token();
87 extern void      socket_token();
88 extern void      sequence_token(adr_t *, parse_context_t *);

90 /*
91  * Modifier tokens
92  */

94 extern void      acl_token(adr_t *, parse_context_t *);
95 extern void      attribute_token(adr_t *, parse_context_t *);
96 extern void      s5_IPC_perm_token(adr_t *, parse_context_t *);
97 extern void      group_token();
98 extern void      label_token(adr_t *, parse_context_t *);
99 extern void      privilege_token(adr_t *, parse_context_t *);
100 extern void      useofpriv_token(adr_t *, parse_context_t *);
101 extern void      secflags_token(adr_t *, parse_context_t *);
102 #endif /* ! codereview */
103 extern void      zonename_token(adr_t *, parse_context_t *);

```

```

104 extern void      liaison_token(adr_t *, parse_context_t *);
105 extern void      newgroup_token(adr_t *, parse_context_t *);
106 extern void      exec_args_token(adr_t *, parse_context_t *);
107 extern void      exec_env_token(adr_t *, parse_context_t *);
108 extern void      attribute32_token(adr_t *, parse_context_t *);
109 extern void      useofauth_token(adr_t *, parse_context_t *);
110 extern void      user_token(adr_t *, parse_context_t *);

112 /*
113  * X windows tokens
114  */

116 extern void      xatom_token(adr_t *, parse_context_t *);
117 extern void      xselect_token(adr_t *, parse_context_t *);
118 extern void      xcolormap_token(adr_t *, parse_context_t *);
119 extern void      xcursor_token(adr_t *, parse_context_t *);
120 extern void      xfont_token(adr_t *, parse_context_t *);
121 extern void      xgc_token(adr_t *, parse_context_t *);
122 extern void      xpixmap_token(adr_t *, parse_context_t *);
123 extern void      xproperty_token(adr_t *, parse_context_t *);
124 extern void      xwindow_token(adr_t *, parse_context_t *);
125 extern void      xclient_token(adr_t *, parse_context_t *);

127 /*
128  * Command tokens
129  */

131 extern void      cmd_token(adr_t *, parse_context_t *);
132 extern void      exit_token(adr_t *, parse_context_t *);

134 /*
135  * Miscellaneous tokens
136  */

138 extern void      host_token(adr_t *, parse_context_t *);

140 /*
141  * Solaris64 tokens
142  */

144 extern void      argument64_token(adr_t *, parse_context_t *);
145 extern void      return64_token(adr_t *, parse_context_t *);
146 extern void      attribute64_token(adr_t *, parse_context_t *);
147 extern void      header64_token(adr_t *, parse_context_t *);
148 extern void      subject64_token(adr_t *, parse_context_t *);
149 extern void      process64_token(adr_t *, parse_context_t *);
150 extern void      file64_token(adr_t *, parse_context_t *);

152 /*
153  * Extended network address tokens
154  */

156 extern void      header64_ex_token();
157 extern void      subject32_ex_token();
158 extern void      process32_ex_token();
159 extern void      subject64_ex_token(adr_t *, parse_context_t *);
160 extern void      process64_ex_token(adr_t *, parse_context_t *);
161 extern void      ip_addr_ex_token(adr_t *, parse_context_t *);
162 extern void      socket_ex_token(adr_t *, parse_context_t *);
163 extern void      tid_token(adr_t *, parse_context_t *);
164 #endif

166 #ifdef __cplusplus
167 }
168 #endif

```

new/usr/src/lib/auditd_plugins/syslog/systoken.h

3

```
170 #endif /* _SYSTOKEN_H */
```

new/usr/src/lib/brand/ipkg/zone/config.xml

1

```
*****
4946 Thu Jun 30 22:01:58 2016
new/usr/src/lib/brand/ipkg/zone/config.xml
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
1 <?xml version="1.0"?>
3 <!--
4 CDDL HEADER START

6 The contents of this file are subject to the terms of the
7 Common Development and Distribution License (the "License").
8 You may not use this file except in compliance with the License.

10 You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
11 or http://www.opensolaris.org/os/licensing.
12 See the License for the specific language governing permissions
13 and limitations under the License.

15 When distributing Covered Code, include this CDDL HEADER in each
16 file and include the License file at usr/src/OPENSOLARIS.LICENSE.
17 If applicable, add the following below this CDDL HEADER, with the
18 fields enclosed by brackets "[]" replaced with your own identifying
19 information: Portions Copyright [yyyy] [name of copyright owner]

21 CDDL HEADER END

23 Copyright (c) 2006, 2010, Oracle and/or its affiliates. All rights reserved.

25 DO NOT EDIT THIS FILE.
26 Copyright 2014 Nexenta Systems, Inc. All rights reserved.
27 -->

29 <!DOCTYPE brand PUBLIC "-//Sun Microsystems Inc//DTD Brands//EN"
30 "file:///usr/share/lib/xml/dtd/brand.dtd.1">

32 <brand name="ipkg">
33   <modname></modname>

35   <initname>/sbin/init</initname>
36   <login_cmd>/usr/bin/login -z %Z %u</login_cmd>
37   <forcedlogin_cmd>/usr/bin/login -z %Z -f %u</forcedlogin_cmd>
38   <user_cmd>/usr/bin/getent passwd %u</user_cmd>

40   <!-- We may not be able to do the create in pkg(1) proper. -->
41   <install>/usr/lib/brand/ipkg/pkgcreatezone -z %z -R %R</install>
42   <installopts>a:c:d:e:hk:P:p:suv</installopts>
43   <boot></boot>
44   <sysboot>/usr/lib/brand/ipkg/prestate %z %R 2 0</sysboot>
45   <halt></halt>
46   <shutdown>/usr/sbin/shutdown -y -g0 -i5</shutdown>
47   <verify_cfg></verify_cfg>
48   <verify_admin></verify_admin>
49   <postclone></postclone>
50   <postinstall></postinstall>
51   <attach>/usr/lib/brand/ipkg/attach %z %R</attach>
52   <detach>/usr/lib/brand/ipkg/detach -z %z -R %R</detach>
53   <clone>/usr/lib/brand/ipkg/clone -z %z -R %R</clone>
54   <uninstall>/usr/lib/brand/ipkg/uninstall %z %R</uninstall>
55   <prestatechange>/usr/lib/brand/ipkg/prestate %z %R</prestatechange>
56   <poststatechange>/usr/lib/brand/ipkg/poststate %z %R</poststatechange>
57   <query>/usr/lib/brand/shared/query %z %R</query>
```

new/usr/src/lib/brand/ipkg/zone/config.xml

2

```
59   <privilege set="default" name="contract_event" />
60   <privilege set="default" name="contract_identity" />
61   <privilege set="default" name="contract_observer" />
62   <privilege set="default" name="file_chown" />
63   <privilege set="default" name="file_chown_self" />
64   <privilege set="default" name="file_dac_execute" />
65   <privilege set="default" name="file_dac_read" />
66   <privilege set="default" name="file_dac_search" />
67   <privilege set="default" name="file_dac_write" />
68   <privilege set="default" name="file_owner" />
69   <privilege set="default" name="file_setid" />
70   <privilege set="default" name="ipc_dac_read" />
71   <privilege set="default" name="ipc_dac_write" />
72   <privilege set="default" name="ipc_owner" />
73   <privilege set="default" name="net_bindmip" />
74   <privilege set="default" name="net_icmpaccess" />
75   <privilege set="default" name="net_mac_aware" />
76   <privilege set="default" name="net_observability" />
77   <privilege set="default" name="net_privaddr" />
78   <privilege set="default" name="net_rawaccess" ip-type="exclusive" />
79   <privilege set="default" name="proc_chroot" />
80   <privilege set="default" name="sys_audit" />
81   <privilege set="default" name="proc_audit" />
82   <privilege set="default" name="proc_lock_memory" />
83   <privilege set="default" name="proc_owner" />
84   <privilege set="default" name="proc_secflags" />
85 #endif /* ! codereview */
86   <privilege set="default" name="proc_setid" />
87   <privilege set="default" name="proc_taskid" />
88   <privilege set="default" name="sys_acct" />
89   <privilege set="default" name="sys_admin" />
90   <privilege set="default" name="sys_ip_config" ip-type="exclusive" />
91   <privilege set="default" name="sys iptun_config" ip-type="exclusive" />
92   <privilege set="default" name="sys_mount" />
93   <privilege set="default" name="sys_nfs" />
94   <privilege set="default" name="sys_ppp_config" ip-type="exclusive" />
95   <privilege set="default" name="sys_resource" />
96   <privilege set="default" name="sys_smb" />

98   <privilege set="prohibited" name="dtrace_kernel" />
99   <privilege set="prohibited" name="proc_zone" />
100  <privilege set="prohibited" name="sys_config" />
101  <privilege set="prohibited" name="sys_devices" />
102  <privilege set="prohibited" name="sys_ip_config" ip-type="shared" />
103  <privilege set="prohibited" name="sys_linkdir" />
104  <privilege set="prohibited" name="sys_net_config" />
105  <privilege set="prohibited" name="sys_res_config" />
106  <privilege set="prohibited" name="sys_suser_compat" />
107  <privilege set="prohibited" name="xvm_control" />
108  <privilege set="prohibited" name="virt_manage" />
109  <privilege set="prohibited" name="sys_ppp_config" ip-type="shared" />

111  <privilege set="required" name="proc_exec" />
112  <privilege set="required" name="proc_fork" />
113  <privilege set="required" name="sys_ip_config" ip-type="exclusive" />
114  <privilege set="required" name="sys_mount" />
115 </brand>
```

```

*****
4906 Thu Jun 30 22:01:58 2016
new/usr/src/lib/brand/labeled/zone/config.xml
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
1 <?xml version="1.0"?>
3 <!--
4 CDDL HEADER START

6 The contents of this file are subject to the terms of the
7 Common Development and Distribution License (the "License").
8 You may not use this file except in compliance with the License.

10 You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
11 or http://www.opensolaris.org/os/licensing.
12 See the License for the specific language governing permissions
13 and limitations under the License.

15 When distributing Covered Code, include this CDDL HEADER in each
16 file and include the License file at usr/src/OPENSOLARIS.LICENSE.
17 If applicable, add the following below this CDDL HEADER, with the
18 fields enclosed by brackets "[]" replaced with your own identifying
19 information: Portions Copyright [yyyy] [name of copyright owner]

21 CDDL HEADER END

23 Copyright (c) 2006, 2010, Oracle and/or its affiliates. All rights reserved.

25 DO NOT EDIT THIS FILE.
26 Copyright 2014 Nexenta Systems, Inc. All rights reserved.
27 -->

29 <!DOCTYPE brand PUBLIC "-//Sun Microsystems Inc//DTD Brands//EN"
30 "file:///usr/share/lib/xml/dtd/brand.dtd.1">

32 <brand name="labeled">
33   <modname></modname>

35   <initname>/sbin/init</initname>
36   <login_cmd>/usr/bin/login -z %Z %u</login_cmd>
37   <forcedlogin_cmd>/usr/bin/login -z %Z -f %u</forcedlogin_cmd>

39   <user_cmd>/usr/bin/getent passwd %u</user_cmd>

41   <!-- We may not be able to do the create in pkg(1) proper. -->
42   <install>/usr/lib/brand/ipkg/pkgcreatezone -z %z -R %R</install>
43   <installopts>a:c:d:e:hk:P:p:suv</installopts>
44   <boot></boot>
45   <sysboot>/usr/lib/brand/ipkg/prestate %z %R 2 0</sysboot>
46   <halt></halt>
47   <shutdown>/usr/sbin/shutdown -y -g0 -i5</shutdown>
48   <verify_cfg></verify_cfg>
49   <verify_adm></verify_adm>
50   <postclone></postclone>
51   <postinstall></postinstall>
52   <attach>/usr/lib/brand/ipkg/attach %z %R</attach>
53   <detach>/usr/lib/brand/ipkg/detach -z %z -R %R</detach>
54   <clone>/usr/lib/brand/ipkg/clone -z %z -R %R</clone>
55   <uninstall>/usr/lib/brand/ipkg/uninstall %z %R</uninstall>
56   <prestatechange>/usr/lib/brand/ipkg/prestate %z %R</prestatechange>
57   <poststatechange>/usr/lib/brand/ipkg/poststate %z %R</poststatechange>
58   <query>/usr/lib/brand/shared/query %z %R</query>

```

```

60   <privilege set="default" name="contract_event" />
61   <privilege set="default" name="contract_identity" />
62   <privilege set="default" name="contract_observer" />
63   <privilege set="default" name="file_chown" />
64   <privilege set="default" name="file_chown_self" />
65   <privilege set="default" name="file_dac_execute" />
66   <privilege set="default" name="file_dac_read" />
67   <privilege set="default" name="file_dac_search" />
68   <privilege set="default" name="file_dac_write" />
69   <privilege set="default" name="file_owner" />
70   <privilege set="default" name="file_setid" />
71   <privilege set="default" name="ipc_dac_read" />
72   <privilege set="default" name="ipc_dac_write" />
73   <privilege set="default" name="ipc_owner" />
74   <privilege set="default" name="net_bindmip" />
75   <privilege set="default" name="net_icmpaccess" />
76   <privilege set="default" name="net_mac_aware" />
77   <privilege set="default" name="net_observability" />
78   <privilege set="default" name="net_privaddr" />
79   <privilege set="default" name="net_rawaccess" ip-type="exclusive" />
80   <privilege set="default" name="proc_chroot" />
81   <privilege set="default" name="sys_audit" />
82   <privilege set="default" name="proc_audit" />
83   <privilege set="default" name="proc_lock_memory" />
84   <privilege set="default" name="proc_owner" />
85   <privilege set="default" name="proc_secflags" />
86 #endif /* ! codereview */
87   <privilege set="default" name="proc_setid" />
88   <privilege set="default" name="proc_taskid" />
89   <privilege set="default" name="sys_acct" />
90   <privilege set="default" name="sys_admin" />
91   <privilege set="default" name="sys_ip_config" ip-type="exclusive" />
92   <privilege set="default" name="sys iptun_config" ip-type="exclusive" />
93   <privilege set="default" name="sys_mount" />
94   <privilege set="default" name="sys_nfs" />
95   <privilege set="default" name="sys_resource" />
96   <privilege set="default" name="sys_ppp_config" ip-type="exclusive" />

98   <privilege set="prohibited" name="dtrace_kernel" />
99   <privilege set="prohibited" name="proc_zone" />
100  <privilege set="prohibited" name="sys_config" />
101  <privilege set="prohibited" name="sys_devices" />
102  <privilege set="prohibited" name="sys_ip_config" ip-type="shared" />
103  <privilege set="prohibited" name="sys_linkdir" />
104  <privilege set="prohibited" name="sys_net_config" />
105  <privilege set="prohibited" name="sys_res_config" />
106  <privilege set="prohibited" name="sys_suser_compat" />
107  <privilege set="prohibited" name="xvm_control" />
108  <privilege set="prohibited" name="virt_manage" />
109  <privilege set="prohibited" name="sys_ppp_config" ip-type="shared" />

111  <privilege set="required" name="proc_exec" />
112  <privilege set="required" name="proc_fork" />
113  <privilege set="required" name="sys_ip_config" ip-type="exclusive" />
114  <privilege set="required" name="sys_mount" />
115 </brand>

```


new/usr/src/lib/brand/sn1/zone/config.xml

1

```
*****
4650 Thu Jun 30 22:01:59 2016
new/usr/src/lib/brand/sn1/zone/config.xml
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
1 <?xml version="1.0"?>
3 <!--
4 CDDL HEADER START
6 The contents of this file are subject to the terms of the
7 Common Development and Distribution License (the "License").
8 You may not use this file except in compliance with the License.
10 You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
11 or http://www.opensolaris.org/os/licensing.
12 See the License for the specific language governing permissions
13 and limitations under the License.
15 When distributing Covered Code, include this CDDL HEADER in each
16 file and include the License file at usr/src/OPENSOLARIS.LICENSE.
17 If applicable, add the following below this CDDL HEADER, with the
18 fields enclosed by brackets "[]" replaced with your own identifying
19 information: Portions Copyright [yyyy] [name of copyright owner]
21 CDDL HEADER END
23 Copyright (c) 2006, 2010, Oracle and/or its affiliates. All rights reserved.
25 DO NOT EDIT THIS FILE.
26 -->
28 <!DOCTYPE brand PUBLIC "-//Sun Microsystems Inc//DTD Brands//EN"
29 "file:///usr/share/lib/xml/dtd/brand.dtd.1">
31 <brand name="sn1">
32   <modname>sn1_brand</modname>
34   <initname>/sbin/init</initname>
35   <login_cmd>/usr/bin/login -z %Z %u</login_cmd>
36   <forcedlogin_cmd>/usr/bin/login -z %Z -f %u</forcedlogin_cmd>
38   <user_cmd>/usr/bin/getent passwd %u</user_cmd>
40   <install>/usr/lib/brand/ipkg/pkgcreatezone -z %z -R %R</install>
41   <boot>/usr/lib/brand/sn1/sn1_boot %R</boot>
42   <sysboot>/usr/lib/brand/ipkg/prestate %z %R 2 0</sysboot>
43   <verify_cfg></verify_cfg>
44   <verify_adm></verify_adm>
45   <attach>/usr/lib/brand/ipkg/attach %z %R</attach>
46   <detach>/usr/lib/brand/ipkg/detach -z %z -R %R</detach>
47   <clone>/usr/lib/brand/ipkg/clone -z %z -R %R</clone>
48   <uninstall>/usr/lib/brand/ipkg/uninstall %z %R</uninstall>
49   <prestatechange>/usr/lib/brand/ipkg/prestate %z %R</prestatechange>
50   <poststatechange>/usr/lib/brand/ipkg/poststate %z %R</poststatechange>
51   <query>/usr/lib/brand/shared/query %z %R</query>
53   <privilege set="default" name="contract_event" />
54   <privilege set="default" name="contract_identity" />
55   <privilege set="default" name="contract_observer" />
56   <privilege set="default" name="file_chown" />
57   <privilege set="default" name="file_chown_self" />
58   <privilege set="default" name="file_dac_execute" />
```

new/usr/src/lib/brand/sn1/zone/config.xml

2

```
59   <privilege set="default" name="file_dac_read" />
60   <privilege set="default" name="file_dac_search" />
61   <privilege set="default" name="file_dac_write" />
62   <privilege set="default" name="file_owner" />
63   <privilege set="default" name="file_setid" />
64   <privilege set="default" name="ipc_dac_read" />
65   <privilege set="default" name="ipc_dac_write" />
66   <privilege set="default" name="ipc_owner" />
67   <privilege set="default" name="net_bindmip" />
68   <privilege set="default" name="net_icmpaccess" />
69   <privilege set="default" name="net_mac_aware" />
70   <privilege set="default" name="net_observability" />
71   <privilege set="default" name="net_privaddr" />
72   <privilege set="default" name="net_rawaccess" ip-type="exclusive" />
73   <privilege set="default" name="proc_chroot" />
74   <privilege set="default" name="sys_audit" />
75   <privilege set="default" name="proc_audit" />
76   <privilege set="default" name="proc_lock_memory" />
77   <privilege set="default" name="proc_owner" />
78   <privilege set="default" name="proc_secflags" />
79 #endif /* ! codereview */
80   <privilege set="default" name="proc_setid" />
81   <privilege set="default" name="proc_taskid" />
82   <privilege set="default" name="sys_acct" />
83   <privilege set="default" name="sys_admin" />
84   <privilege set="default" name="sys_ip_config" ip-type="exclusive" />
85   <privilege set="default" name="sys_iptun_config" ip-type="exclusive" />
86   <privilege set="default" name="sys_mount" />
87   <privilege set="default" name="sys_nfs" />
88   <privilege set="default" name="sys_resource" />
89   <privilege set="default" name="sys_ppp_config" ip-type="exclusive" />
91   <privilege set="prohibited" name="dtrace_kernel" />
92   <privilege set="prohibited" name="proc_zone" />
93   <privilege set="prohibited" name="sys_config" />
94   <privilege set="prohibited" name="sys_devices" />
95   <privilege set="prohibited" name="sys_ip_config" ip-type="shared" />
96   <privilege set="prohibited" name="sys_linkdir" />
97   <privilege set="prohibited" name="sys_net_config" />
98   <privilege set="prohibited" name="sys_res_config" />
99   <privilege set="prohibited" name="sys_suser_compat" />
100  <privilege set="prohibited" name="xvm_control" />
101  <privilege set="prohibited" name="virt_manage" />
102  <privilege set="prohibited" name="sys_ppp_config" ip-type="shared" />
104  <privilege set="required" name="proc_exec" />
105  <privilege set="required" name="proc_fork" />
106  <privilege set="required" name="sys_ip_config" ip-type="exclusive" />
107  <privilege set="required" name="sys_mount" />
108 </brand>
```

```

*****
8818 Thu Jun 30 22:02:00 2016
new/usr/src/lib/libbasm/adt_record.dtd.1
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
1 <?xml version="1.0" encoding="UTF-8" ?>

3 <!--
4 Copyright 2010 Sun Microsystems, Inc. All rights reserved.
5 Use is subject to license terms.

7 CDDL HEADER START

9 The contents of this file are subject to the terms of the
10 Common Development and Distribution License (the "License").
11 You may not use this file except in compliance with the License.

13 You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
14 or http://www.opensolaris.org/os/licensing.
15 See the License for the specific language governing permissions
16 and limitations under the License.

18 When distributing Covered Code, include this CDDL HEADER in each
19 file and include the License file at usr/src/OPENSOLARIS.LICENSE.
20 If applicable, add the following below this CDDL HEADER, with the
21 fields enclosed by brackets "[]" replaced with your own identifying
22 information: Portions Copyright [yyyy] [name of copyright owner]

24 CDDL HEADER END
25 -->

28 <!--Entity Definitions-->

30 <!-- timeattr or iso8601

32 timeattr:
33     the time/date to the second in strftime(3C) default format,
34     followed by milliseconds offset.

36     Example:         time="Mon May 06 12:10:18 2002" msec="750"

38 iso8601:
39     ISO 8601 standard format date time and timezone;
40     YYYY-MM-DD HH:MM:SS.sss +/-HH:MM; year, month, day 24 hour time with
41     milliseconds + or - offset from Universal Time (UTC, aka GMT)
42
43     Example:         iso8601="2003-09-17 16:47:41.831 -07:00"

45 -->
46 <!ENTITY % timeattr      "time          CDATA #IMPLIED
47                          msec          CDATA #IMPLIED">

49 <!ENTITY % iso8601       "iso8601       CDATA #IMPLIED">

51 <!-- xinfo      Generic info for X related tokens. -->
52 <!ENTITY % xinfo        "xid          CDATA #REQUIRED
53                          xcreator-uid CDATA #REQUIRED">

55 <!-- reserved_toks

57 This represents the set of "reserved" tokens whose placement is
58 fixed.

```

```

60 -->
61 <!ENTITY % reserved_toks "(
62                          file
63                          record
64                          host
65                          sequence
66                          )"
67 ">

69 <!-- normaltoks

71 This represents the set of all tokens other than the "reserved"
72 tokens.

74 -->
75 <!ENTITY % normaltoks "(
76                          acl
77                          arbitrary
78                          argument
79                          attribute
80                          cmd
81                          exit
82                          exec_args
83                          exec_env
84                          fmri
85                          group
86                          ip
87                          ip_address
88                          IPC
89                          IPC_perm
90                          ip_port
91                          liaison
92                          opaque
93                          path
94                          path_attr
95                          privilege
96                          process
97                          return
98                          sensitivity_label
99                          old_socket
100                         socket
101                         subject
102                         text
103                         user
104                         use_of_authorization
105                         use_of_privilege
106                         X_atom
107                         X_client
108                         X_color_map
109                         X_cursor
110                         X_font
111                         X_graphic_context
112                         X_pixmap
113                         X_property
114                         X_selection
115                         X_window
116                         zone
117                         )"
118 ">

120 <!--Element Definitions-->

122 <!--

124 The main element, "audit", consists of a sequence of file & record tokens.

```

```

126 -->
127 <!ELEMENT audit (file | record)*>

129 <!-- file token -->
130 <!ELEMENT file      (#PCDATA)>
131 <!ATTLIST file      %iso8601;>

134 <!-- record token

136 Audit records will have this general layout of tokens after the
137 first token (which is the record token):
138     (tokens),subject,group,(tokens),return,sequence,host

140 (all tokens after the record token are optional; the host token is unused.)

142 -->
143 <!ELEMENT record (
144     (%normaltoks;)*,
145     sequence?,
146     host?
147 )
148 >
149 <!ATTLIST record
150     version      CDATA #REQUIRED
151     event        CDATA #REQUIRED
152     modifier     CDATA #IMPLIED
153     host         CDATA #IMPLIED
154     %iso8601;
155 >

157 <!-- text token -->
158 <!ELEMENT text      (#PCDATA)>

160 <!-- user token -->
161 <!ELEMENT user      EMPTY>
162 <!ATTLIST user
163     uid          CDATA #REQUIRED
164     username    CDATA #REQUIRED
165 >

167 <!-- path token -->
168 <!ELEMENT path      (#PCDATA)>

170 <!-- path_attr token -->
171 <!ELEMENT path_attr (xattr*)>
172 <!ELEMENT xattr     (#PCDATA)>

174 <!-- host token -->
175 <!ELEMENT host      (#PCDATA)>

177 <!-- subject token -->
178 <!ELEMENT subject   EMPTY>
179 <!ATTLIST subject
180     audit-uid   CDATA #REQUIRED
181     uid         CDATA #REQUIRED
182     gid         CDATA #REQUIRED
183     ruid        CDATA #REQUIRED
184     rgid        CDATA #REQUIRED
185     pid         CDATA #REQUIRED
186     sid         CDATA #REQUIRED
187     tid         CDATA #REQUIRED
188 >

190 <!-- process token -->

```

```

191 <!ELEMENT process   EMPTY>
192 <!ATTLIST process
193     audit-uid   CDATA #REQUIRED
194     uid         CDATA #REQUIRED
195     gid         CDATA #REQUIRED
196     ruid        CDATA #REQUIRED
197     rgid        CDATA #REQUIRED
198     pid         CDATA #REQUIRED
199     sid         CDATA #REQUIRED
200     tid         CDATA #REQUIRED
201 >

203 <!-- return token -->
204 <!ELEMENT return    EMPTY>
205 <!ATTLIST return
206     errval      CDATA #REQUIRED
207     retval      CDATA #REQUIRED
208 >

210 <!-- exit token -->
211 <!ELEMENT exit      EMPTY>
212 <!ATTLIST exit
213     errval      CDATA #REQUIRED
214     retval      CDATA #REQUIRED
215 >

217 <!-- sequence token -->
218 <!ELEMENT sequence  EMPTY>
219 <!ATTLIST sequence
220     seq-num     CDATA #REQUIRED
221 >

223 <!-- fmri token -->
224 <!ELEMENT fmri      (#PCDATA)>

226 <!-- group token -->
227 <!ELEMENT group     (gid)*>
228 <!ELEMENT gid       (#PCDATA)>

230 <!-- opaque token -->
231 <!ELEMENT opaque    (#PCDATA)>

233 <!-- liaison token -->
234 <!-- (NOTE: liaison is obsolete and is no longer generated -->
235 <!ELEMENT liaison  (#PCDATA)>

237 <!-- argument token -->
238 <!ELEMENT argument  EMPTY>
239 <!ATTLIST argument
240     arg-num     CDATA #REQUIRED
241     value       CDATA #REQUIRED
242     desc        CDATA #REQUIRED
243 >

245 <!-- attribute token -->
246 <!ELEMENT attribute EMPTY>
247 <!ATTLIST attribute
248     mode        CDATA #REQUIRED
249     uid         CDATA #REQUIRED
250     gid         CDATA #REQUIRED
251     fsid        CDATA #REQUIRED
252     nodeid      CDATA #REQUIRED
253     device      CDATA #REQUIRED
254 >

256 <!-- cmd token -->

```

```

257 <!ELEMENT cmd (argv*, arge*)>
258 <!ELEMENT argv (#PCDATA)>
259 <!ELEMENT arge (#PCDATA)>

261 <!-- exec_args token -->
262 <!ELEMENT exec_args (arg*)>
263 <!ELEMENT arg (#PCDATA)>

265 <!-- exec_env token -->
266 <!ELEMENT exec_env (env*)>
267 <!ELEMENT env (#PCDATA)>

269 <!-- arbitrary token -->
270 <!ELEMENT arbitrary (#PCDATA)>
271 <!ATTLIST arbitrary
272 print CDATA #REQUIRED
273 type CDATA #REQUIRED
274 count CDATA #REQUIRED
275 >

277 <!-- privilege token -->
278 <!ELEMENT privilege (#PCDATA)>
279 <!ATTLIST privilege
280 set-type CDATA #REQUIRED
281 >

283 <!-- secflags token -->
284 <!ELEMENT secflags (#PCDATA)>
285 <!ATTLIST secflags
286 set-type CDATA #REQUIRED
287 >

290 #endif /* !codereview */
291 <!-- use_of_privilege token -->
292 <!ELEMENT use_of_privilege (#PCDATA)>
293 <!ATTLIST use_of_privilege
294 result CDATA #REQUIRED
295 >

297 <!-- sensitivity_label token -->
298 <!ELEMENT sensitivity_label (#PCDATA)>

300 <!-- use_of_authorization token -->
301 <!ELEMENT use_of_authorization (#PCDATA)>

303 <!-- IPC token -->
304 <!ELEMENT IPC EMPTY>
305 <!ATTLIST IPC
306 ipc-type CDATA #REQUIRED
307 ipc-id CDATA #REQUIRED
308 >

310 <!-- IPC_perm token -->
311 <!ELEMENT IPC_perm EMPTY>
312 <!ATTLIST IPC_perm
313 uid CDATA #REQUIRED
314 gid CDATA #REQUIRED
315 creator-uid CDATA #REQUIRED
316 creator-gid CDATA #REQUIRED
317 mode CDATA #REQUIRED
318 seq CDATA #REQUIRED
319 key CDATA #REQUIRED
320 >

322 <!-- ip_address token -->

```

```

323 <!ELEMENT ip_address (#PCDATA)>

325 <!-- ip_port token -->
326 <!-- (NOTE: ip_port is obsolete and is no longer generated -->
327 <!ELEMENT ip_port (#PCDATA)>

329 <!-- ip token -->
330 <!-- (NOTE: ip is obsolete and is no longer generated -->
331 <!ELEMENT ip EMPTY>
332 <!ATTLIST ip
333 version CDATA #REQUIRED
334 service_type CDATA #REQUIRED
335 len CDATA #REQUIRED
336 id CDATA #REQUIRED
337 offset CDATA #REQUIRED
338 time_to_live CDATA #REQUIRED
339 protocol CDATA #REQUIRED
340 cksum CDATA #REQUIRED
341 src_addr CDATA #REQUIRED
342 dest_addr CDATA #REQUIRED
343 >

345 <!-- old_socket token -->
346 <!ELEMENT old_socket EMPTY>
347 <!ATTLIST old_socket
348 type CDATA #REQUIRED
349 port CDATA #REQUIRED
350 addr CDATA #REQUIRED
351 >

353 <!-- socket token -->
354 <!ELEMENT socket EMPTY>
355 <!ATTLIST socket
356 sock_domain CDATA #REQUIRED
357 sock_type CDATA #REQUIRED
358 lport CDATA #REQUIRED
359 laddr CDATA #REQUIRED
360 fport CDATA #REQUIRED
361 faddr CDATA #REQUIRED
362 >

364 <!-- acl token -->
365 <!ELEMENT acl EMPTY>
366 <!ATTLIST acl
367 type CDATA #IMPLIED
368 value CDATA #IMPLIED
369 mode CDATA #IMPLIED
370 flags CDATA #IMPLIED
371 id CDATA #IMPLIED
372 access_mask CDATA #IMPLIED
373 >

375 <!-- tid token -->
376 <!-- future intent: contain one of ipadr | MTUadr | device -->
377 <!ELEMENT tid (ipadr*)>
378 <!ATTLIST tid
379 type CDATA #REQUIRED
380 >

382 <!-- ipadr content of tid token -->
383 <!ELEMENT ipadr EMPTY>
384 <!ATTLIST ipadr
385 local-port CDATA #REQUIRED
386 remote-port CDATA #REQUIRED
387 host CDATA #REQUIRED
388 >

```

```
390 <!-- X_atom token -->
391 <!ELEMENT X_atom          (#PCDATA)>

393 <!-- X_color_map token -->
394 <!ELEMENT X_color_map    EMPTY>
395 <!ATTLIST X_color_map    %xinfo;>

397 <!-- X_cursor token -->
398 <!ELEMENT X_cursor       EMPTY>
399 <!ATTLIST X_cursor       %xinfo;>

401 <!-- X_font token -->
402 <!ELEMENT X_font         EMPTY>
403 <!ATTLIST X_font         %xinfo;>

405 <!-- X_graphic_context token -->
406 <!ELEMENT X_graphic_context EMPTY>
407 <!ATTLIST X_graphic_context %xinfo;>

409 <!-- X_pixmap token -->
410 <!ELEMENT X_pixmap       EMPTY>
411 <!ATTLIST X_pixmap       %xinfo;>

413 <!-- X_window token -->
414 <!ELEMENT X_window       EMPTY>
415 <!ATTLIST X_window       %xinfo;>

417 <!-- X_property token -->
418 <!ELEMENT X_property     (#PCDATA)>
419 <!ATTLIST X_property     %xinfo;>

421 <!-- X_client token -->
422 <!ELEMENT X_client       (#PCDATA)>

424 <!-- X_selection token -->
425 <!ELEMENT X_selection    (xsel_text, xsel_type, xsel_data)>
426 <!ELEMENT x_sel_text    (#PCDATA)>
427 <!ELEMENT x_sel_type    (#PCDATA)>
428 <!ELEMENT x_sel_data    (#PCDATA)>

430 <!-- zonename token -->
431 <!ELEMENT zone          EMPTY>
432 <!ATTLIST zone         name          CDATA #REQUIRED
433
434 >
```

new/usr/src/lib/libbssm/adt_record.xsl.1

1

```
*****
12261 Thu Jun 30 22:02:01 2016
new/usr/src/lib/libbssm/adt_record.xsl.1
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
1 <?xml version="1.0" encoding="UTF-8" ?>

3 <!--
4 Copyright 2010 Sun Microsystems, Inc. All rights reserved.
5 Use is subject to license terms.

7 CDDL HEADER START

9 The contents of this file are subject to the terms of the
10 Common Development and Distribution License (the "License").
11 You may not use this file except in compliance with the License.

13 You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
14 or http://www.opensolaris.org/os/licensing.
15 See the License for the specific language governing permissions
16 and limitations under the License.

18 When distributing Covered Code, include this CDDL HEADER in each
19 file and include the License file at usr/src/OPENSOLARIS.LICENSE.
20 If applicable, add the following below this CDDL HEADER, with the
21 fields enclosed by brackets "[]" replaced with your own identifying
22 information: Portions Copyright [yyyy] [name of copyright owner]

24 CDDL HEADER END
25 -->

27 <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

29 <!-- set the output properties -->
30 <xsl:output method="html"/>

32 <!-- root rule -->
33 <xsl:template match="/">
34 <HTML>
35 <HEAD><TITLE>Audit Trail Data</TITLE></HEAD>
36 <BODY BGColor="#FFFFFF" Text="#000000">
37 <CENTER>
38 <FONT FACE="Arial" SIZE="+1">
39 <B>Audit Trail Data</B>
40 </FONT>
41 <BR/>
42 </CENTER>
43 <xsl:apply-templates/>
44 <HR/>
45 </BODY>
46 </HTML>
47 </xsl:template>

49 <!-- suppress non-selected nodes-->
50 <xsl:template match="*" />

52 <!-- main rule for document element -->
53 <xsl:template match="audit">
54 <HR/>
55 <xsl:for-each select="record | file">
56 <xsl:if test="(self::file)">
57 <BR/>
58 <BR/>
```

new/usr/src/lib/libbssm/adt_record.xsl.1

2

```
59 <B>File: </B>
60 <I>time: </I>
61 <xsl:choose>
62 <xsl:when test="@time">
63 <xsl:value-of select="@time"/>
64 + <xsl:value-of select="@msec"/>msec
65 </xsl:when>
66 <xsl:when test="@iso8601">
67 <xsl:value-of select="@iso8601"/>
68 </xsl:when>
69 </xsl:choose>
70 <BR/>
71 <xsl:value-of select="."/>
72 </xsl:if>
73 <xsl:if test="(self::record)">
74 <BR/>
75 <BR/>
76 <B>Event: </B>
77 <B><xsl:value-of select="@event"/></B><BR/>
78 <I>time: </I>
79 <xsl:choose>
80 <xsl:when test="@time">
81 <xsl:value-of select="@time"/>
82 + <xsl:value-of select="@msec"/>msec
83 </xsl:when>
84 <xsl:when test="@iso8601">
85 <xsl:value-of select="@iso8601"/>
86 </xsl:when>
87 </xsl:choose>
88 <I> vers: </I><xsl:value-of select="@version"/>
89 <I> mod: </I><xsl:value-of select="@modifier"/>
90 <I> host: </I><xsl:value-of select="@host"/>
91 <xsl:apply-templates/>
92 </xsl:if>
93 </xsl:for-each>
94 </xsl:template>

96 <!-- Start of handling for remaining tokens -->

98 <xsl:template match="text">
99 <BR/>
100 <I>TEXT: </I> <xsl:value-of select="."/>
101 </xsl:template>

103 <xsl:template match="path">
104 <BR/>
105 <I>PATH: </I> <xsl:value-of select="."/>
106 </xsl:template>

108 <xsl:template match="path_attr">
109 <BR/>
110 <I>PATH_ATTR </I>
111 <xsl:apply-templates/>
112 </xsl:template>

114 <xsl:template match="xattr">
115 <BR/>
116 <I>xattr: </I> <xsl:value-of select="."/>
117 </xsl:template>

119 <xsl:template match="host">
120 <BR/>
121 <I>HOST: </I> <xsl:value-of select="."/>
122 </xsl:template>

124 <xsl:template match="subject">
```

```

125     <BR/>
126     <I>SUBJECT </I>
127     <I> audit-uid: </I><xsl:value-of select="@audit-uid"/>
128     <I> uid: </I><xsl:value-of select="@uid"/>
129     <I> gid: </I><xsl:value-of select="@gid"/>
130     <I> ruid: </I><xsl:value-of select="@ruid"/>
131     <I> rgid: </I><xsl:value-of select="@rgid"/>
132     <I> pid: </I><xsl:value-of select="@pid"/>
133     <I> sid: </I><xsl:value-of select="@sid"/>
134     <I> tid: </I><xsl:value-of select="@tid"/>
135 </xsl:template>

137 <xsl:template match="process">
138     <BR/>
139     <I>PROCESS </I>
140     <I> audit-uid: </I><xsl:value-of select="@audit-uid"/>
141     <I> uid: </I><xsl:value-of select="@uid"/>
142     <I> gid: </I><xsl:value-of select="@gid"/>
143     <I> ruid: </I><xsl:value-of select="@ruid"/>
144     <I> rgid: </I><xsl:value-of select="@rgid"/>
145     <I> pid: </I><xsl:value-of select="@pid"/>
146     <I> sid: </I><xsl:value-of select="@sid"/>
147     <I> tid: </I><xsl:value-of select="@tid"/>
148 </xsl:template>

150 <xsl:template match="return">
151     <BR/>
152     <I>RETURN </I>
153     <I> errval: </I><xsl:value-of select="@errval"/>
154     <I> retval: </I><xsl:value-of select="@retval"/>
155 </xsl:template>

157 <xsl:template match="exit">
158     <BR/>
159     <I>EXIT </I>
160     <I> errval: </I><xsl:value-of select="@errval"/>
161     <I> retval: </I><xsl:value-of select="@retval"/>
162 </xsl:template>

164 <xsl:template match="sequence">
165     <BR/>
166     <I>SEQUENCE </I>
167     <I> seq-num: </I><xsl:value-of select="@seq-num"/>
168 </xsl:template>

170 <xsl:template match="fmri">
171     <BR/>
172     <I>FMRI: </I> <xsl:value-of select="."/>
173 </xsl:template>

175 <xsl:template match="user">
176     <BR/>
177     <I>USER </I>
178     <I> uid: </I><xsl:value-of select="@uid"/>
179     <I> username: </I><xsl:value-of select="@username"/>
180 </xsl:template>

182 <xsl:template match="group">
183     <BR/>
184     <I>GROUP </I>
185     <xsl:apply-templates/>
186 </xsl:template>

188 <xsl:template match="gid">
189     <BR/>
190     <I>gid: </I> <xsl:value-of select="."/>

```

```

191 </xsl:template>

193 <xsl:template match="opaque">
194     <BR/>
195     <I>OPAQUE: </I> <xsl:value-of select="."/>
196 </xsl:template>

198 <xsl:template match="liaison">
199     <BR/>
200     <I>LIAISON: </I> <xsl:value-of select="."/>
201 </xsl:template>

203 <xsl:template match="argument">
204     <BR/>
205     <I>ARGUMENT </I>
206     <I> arg-num: </I><xsl:value-of select="@arg-num"/>
207     <I> value: </I><xsl:value-of select="@value"/>
208     <I> desc: </I><xsl:value-of select="@desc"/>
209 </xsl:template>

211 <xsl:template match="attribute">
212     <BR/>
213     <I>ATTRIBUTE </I>
214     <I> mode: </I><xsl:value-of select="@mode"/>
215     <I> uid: </I><xsl:value-of select="@uid"/>
216     <I> gid: </I><xsl:value-of select="@gid"/>
217     <I> fsid: </I><xsl:value-of select="@fsid"/>
218     <I> nodeid: </I><xsl:value-of select="@nodeid"/>
219     <I> device: </I><xsl:value-of select="@device"/>
220 </xsl:template>

222 <xsl:template match="cmd">
223     <BR/>
224     <I>CMD </I>
225     <xsl:apply-templates/>
226 </xsl:template>

228 <xsl:template match="argv">
229     <BR/>
230     <I>argv: </I> <xsl:value-of select="."/>
231 </xsl:template>

233 <xsl:template match="arge">
234     <BR/>
235     <I>arge: </I> <xsl:value-of select="."/>
236 </xsl:template>

238 <xsl:template match="exec_args">
239     <BR/>
240     <I>EXEC_ARGS </I>
241     <xsl:apply-templates/>
242 </xsl:template>

244 <xsl:template match="arg">
245     <BR/>
246     <I>arg: </I> <xsl:value-of select="."/>
247 </xsl:template>

249 <xsl:template match="exec_env">
250     <BR/>
251     <I>EXEC_ENV </I>
252     <xsl:apply-templates/>
253 </xsl:template>

255 <xsl:template match="env">
256     <BR/>

```

```

257     <I>env: </I>    <xsl:value-of select="."/>
258 </xsl:template>

260 <xsl:template match="arbitrary">
261   <BR/>
262   <I>ARBITRARY: </I>
263   <I> print: </I><xsl:value-of select="@print"/>
264   <I> type: </I><xsl:value-of select="@type"/>
265   <I> count: </I><xsl:value-of select="@count"/>
266   <BR/>
267   <xsl:value-of select="."/>
268 </xsl:template>

270 <xsl:template match="privilege">
271   <BR/>
272   <I>PRIVILEGE: </I>
273   <I> set-type: </I><xsl:value-of select="@set-type"/>
274   <BR/>
275   <xsl:value-of select="."/>
276 </xsl:template>

278 <xsl:template match="use_of_privilege">
279   <BR/>
280   <I>USE_OF_PRIVILEGE: </I>
281   <I> result: </I><xsl:value-of select="@result"/>
282   <BR/>
283   <xsl:value-of select="."/>
284 </xsl:template>

286 <xsl:template match="secflags">
287   <BR/>
288   <I>SECFLAGS: </I>
289   <I> set-type: </I><xsl:value-of select="@set-type"/>
290   <BR/>
291   <xsl:value-of select="."/>
292 </xsl:template>

294 #endif /* ! codereview */
295 <xsl:template match="sensitivity_label">
296   <BR/>
297   <I>SENSITIVITY_LABEL: </I>    <xsl:value-of select="."/>
298 </xsl:template>

300 <xsl:template match="use_of_authorization">
301   <BR/>
302   <I>USE_OF_AUTHORIZATION: </I>    <xsl:value-of select="."/>
303 </xsl:template>

305 <xsl:template match="IPC">
306   <BR/>
307   <I>IPC </I>
308   <I> ipc-type: </I><xsl:value-of select="@ipc-type"/>
309   <I> ipc-id: </I><xsl:value-of select="@ipc-id"/>
310 </xsl:template>

312 <xsl:template match="IPC_perm">
313   <BR/>
314   <I>IPC_PERM </I>
315   <I> uid: </I><xsl:value-of select="@uid"/>
316   <I> gid: </I><xsl:value-of select="@gid"/>
317   <I> creator-uid: </I><xsl:value-of select="@creator-uid"/>
318   <I> creator-gid: </I><xsl:value-of select="@creator-gid"/>
319   <I> mode: </I><xsl:value-of select="@mode"/>
320   <I> seq: </I><xsl:value-of select="@seq"/>
321   <I> key: </I><xsl:value-of select="@key"/>
322 </xsl:template>

```

```

324 <xsl:template match="ip_address">
325   <BR/>
326   <I>IP_ADDRESS: </I>    <xsl:value-of select="."/>
327 </xsl:template>

329 <xsl:template match="ip_port">
330   <BR/>
331   <I>IP_PORT: </I>    <xsl:value-of select="."/>
332 </xsl:template>

334 <xsl:template match="ip">
335   <BR/>
336   <I>IP </I>
337   <I> version: </I><xsl:value-of select="@version"/>
338   <I> service_type: </I><xsl:value-of select="@service_type"/>
339   <I> len: </I><xsl:value-of select="@len"/>
340   <I> id: </I><xsl:value-of select="@id"/>
341   <I> offset: </I><xsl:value-of select="@offset"/>
342   <I> time_to_live: </I><xsl:value-of select="@time_to_live"/>
343   <I> protocol: </I><xsl:value-of select="@protocol"/>
344   <I> cksum: </I><xsl:value-of select="@cksum"/>
345   <I> src_addr: </I><xsl:value-of select="@src_addr"/>
346   <I> dest_addr: </I><xsl:value-of select="@dest_addr"/>
347 </xsl:template>

349 <xsl:template match="old_socket">
350   <BR/>
351   <I>OLD_SOCKET </I>
352   <I> type: </I><xsl:value-of select="@type"/>
353   <I> port: </I><xsl:value-of select="@port"/>
354   <I> addr: </I><xsl:value-of select="@addr"/>
355 </xsl:template>

357 <xsl:template match="socket">
358   <BR/>
359   <I>SOCKET </I>
360   <I> sock_domain: </I><xsl:value-of select="@sock_domain"/>
361   <I> sock_type: </I><xsl:value-of select="@sock_type"/>
362   <I> lport: </I><xsl:value-of select="@lport"/>
363   <I> laddr: </I><xsl:value-of select="@laddr"/>
364   <I> fport: </I><xsl:value-of select="@fport"/>
365   <I> faddr: </I><xsl:value-of select="@faddr"/>
366 </xsl:template>

368 <xsl:template match="acl">
369   <BR/>
370   <I>ACL </I>
371   <xsl:choose>
372     <xsl:when test="@mode"> <!-- old ACL entry -->
373       <I> type: </I><xsl:value-of select="@type"/>
374       <I> value: </I><xsl:value-of select="@value"/>
375       <I> mode: </I><xsl:value-of select="@mode"/>
376     </xsl:when>
377     <xsl:otherwise>
378       <I> flags: </I><xsl:value-of select="@flags"/>
379       <I> id: </I><xsl:value-of select="@id"/>
380       <I> access_mask: </I><xsl:value-of select="@access_mask"/>
381       <I> type: </I><xsl:value-of select="@type"/>
382     </xsl:otherwise>
383   </xsl:choose>
384 </xsl:template>

386 <xsl:template match="tid">
387   <BR/>
388   <I>terminal id: </I>

```



```

389     <I> type=</I><xsl:value-of select="@type"/>
390     <xsl:apply-templates/>
391 </xsl:template>

393 <xsl:template match="ipadr">
394     <I> local-port: </I><xsl:value-of select="@local-port"/>
395     <I> remote-port: </I><xsl:value-of select="@remote-port"/>
396     <I> host: </I><xsl:value-of select="@host"/>
397 </xsl:template>

399 <xsl:template match="X_atom">
400     <BR/>
401     <I>X_ATOM: </I> <xsl:value-of select="."/>
402 </xsl:template>

404 <xsl:template match="X_color_map">
405     <BR/>
406     <I>X_COLOR_MAP </I>
407     <I> xid: </I><xsl:value-of select="@xid"/><I> xcreator-uid: </I><xsl:val
408 </xsl:template>

410 <xsl:template match="X_cursor">
411     <BR/>
412     <I>X_CURSOR </I>
413     <I> xid: </I><xsl:value-of select="@xid"/><I> xcreator-uid: </I><xsl:val
414 </xsl:template>

416 <xsl:template match="X_font">
417     <BR/>
418     <I>X_FONT </I>
419     <I> xid: </I><xsl:value-of select="@xid"/><I> xcreator-uid: </I><xsl:val
420 </xsl:template>

422 <xsl:template match="X_graphic_context">
423     <BR/>
424     <I>X_GRAPHIC_CONTEXT </I>
425     <I> xid: </I><xsl:value-of select="@xid"/><I> xcreator-uid: </I><xsl:val
426 </xsl:template>

428 <xsl:template match="X_pixmap">
429     <BR/>
430     <I>X_PIXMAP </I>
431     <I> xid: </I><xsl:value-of select="@xid"/><I> xcreator-uid: </I><xsl:val
432 </xsl:template>

434 <xsl:template match="X_window">
435     <BR/>
436     <I>X_WINDOW </I>
437     <I> xid: </I><xsl:value-of select="@xid"/><I> xcreator-uid: </I><xsl:val
438 </xsl:template>

440 <xsl:template match="X_property">
441     <BR/>
442     <I>X_PROPERTY: </I>
443     <I> xid: </I><xsl:value-of select="@xid"/><I> xcreator-uid: </I><xsl:val
444     <BR/>
445     <xsl:value-of select="."/>
446 </xsl:template>

448 <xsl:template match="X_client">
449     <BR/>
450     <I>X_CLIENT: </I> <xsl:value-of select="."/>
451 </xsl:template>

453 <xsl:template match="X_selection">
454     <BR/>

```

```

455     <I>X_SELECTION </I>
456     <xsl:apply-templates/>
457 </xsl:template>

459 <xsl:template match="x_sel_text">
460     <BR/>
461     <I>x_sel_text: </I> <xsl:value-of select="."/>
462 </xsl:template>

464 <xsl:template match="x_sel_type">
465     <BR/>
466     <I>x_sel_type: </I> <xsl:value-of select="."/>
467 </xsl:template>

469 <xsl:template match="x_sel_data">
470     <BR/>
471     <I>x_sel_data: </I> <xsl:value-of select="."/>
472 </xsl:template>

474 <xsl:template match="zone">
475     <BR/>
476     <I>ZONE </I>
477     <I> name: </I><xsl:value-of select="@name"/>
478 </xsl:template>

480 </xsl:stylesheet>

```

new/usr/src/lib/libbssm/audit_event.txt

1

```
*****
28376 Thu Jun 30 22:02:03 2016
new/usr/src/lib/libbssm/audit_event.txt
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
1 #
2 # Copyright (c) 1992, 2010, Oracle and/or its affiliates. All rights reserved.
3 #
4 #
5 # CDDL HEADER START
6 #
7 # The contents of this file are subject to the terms of the
8 # Common Development and Distribution License (the "License").
9 # You may not use this file except in compliance with the License.
10 #
11 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
12 # or http://www.opensolaris.org/os/licensing.
13 # See the License for the specific language governing permissions
14 # and limitations under the License.
15 #
16 # When distributing Covered Code, include this CDDL HEADER in each
17 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
18 # If applicable, add the following below this CDDL HEADER, with the
19 # fields enclosed by brackets "[]" replaced with your own identifying
20 # information: Portions Copyright [yyyy] [name of copyright owner]
21 #
22 # CDDL HEADER END
23 #
24 # Audit Event Database
25 #
26 # File Format:
27 #
28 # event number:event name:event description:event classes (comma separated)
29 #
30 # Used to map audit events to audit classes for preselection and post-selection.
31 # Used by TCB programs that write audit records to preselect audit events
32 # based on event to class mappings.
33 #
34 # NOTE: several events are obsolete but must continue to be defined here for
35 # compatibility reasons. Obsolete events are defined in the "no" (invalid)
36 # class to indicate they will not be generated. Other events in the "no"
37 # class which are not obsolete (but are in this class for other reasons),
38 # are individually noted with a comment for explanation.
39 #
40 # System Administrators: Do NOT modify or add events with an event number less
41 # than 32768. These are reserved by the system.
42 #
43 # 0 Reserved as an invalid event number.
44 # 1 - 2047 Reserved for the Solaris Kernel events.
45 # 2048 - 32767 Reserved for the Solaris TCB programs.
46 # 32768 - 65535 Available for third party TCB applications.
47 #
48 #
49 # Allocation of reserved kernel events:
50 # (NOTE: the kernel event table, and possibly MAX_KEVENTS, must be updated
51 # in audit_kevents.h when changes are made to kernel events.)
52 # 1 - 511 allocated for Solaris
53 # 512 - 2047 (reserved but not allocated)
54 #
55 # Allocation of user level audit events:
56 # 2048 - 5999 (reserved but not allocated)
57 # 6000 - 9999 allocated for Solaris
58 # 10000 - 32767 (reserved but not allocated)
```

new/usr/src/lib/libbssm/audit_event.txt

2

```
59 # 32768 - 65535 (Available for third party TCB applications)
60 #
61 #
62 # kernel audit events
63 # 1 - 511 allocated for Solaris
64 #
65 # 0:AUE_NULL:indir system call:no
66 # 1:AUE_EXIT:exit(2):ps
67 # 2:AUE_FORKALL:forkall(2):ps
68 # AUE_OPEN is a placeholder and will not be generated
69 # 3:AUE_OPEN:open(2) - place holder:no
70 # 4:AUE_CREAT:creat(2):no
71 # 5:AUE_LINK:link(2):fc
72 # 6:AUE_UNLINK:unlink(2):fd
73 # 7:AUE_EXEC:exec(2):no
74 # 8:AUE_CHDIR:chdir(2):pm
75 # 9:AUE_MKNOD:mknod(2):fc
76 # 10:AUE_CHMOD:chmod(2):fm
77 # 11:AUE_CHOWN:chown(2):fm
78 # 12:AUE_UMOUNT:umount(2) - old version:as
79 # 13:AUE_JUNK:junk:no
80 # 14:AUE_ACCESS:access(2):fa
81 # 15:AUE_KILL:kill(2):pm
82 # 16:AUE_STAT:stat(2):fa
83 # 17:AUE_LSTAT:lstat(2):fa
84 # 18:AUE_ACCT:acct(2):as
85 # 19:AUE_MCTL:mctl(2):no
86 # 20:AUE_REBOOT:reboot(2):no
87 # 21:AUE_SYMLINK:symlink(2):fc
88 # 22:AUE_READLINK:readlink(2):fr
89 # 23:AUE_EXECVE:execve(2):ps,ex
90 # 24:AUE_CHROOT:chroot(2):pm
91 # 25:AUE_VFORK:vfork(2):ps
92 # 26:AUE_SETGROUPS:setgroups(2):pm
93 # 27:AUE_SETPGRP:setpgrp(2):pm
94 # 28:AUE_SWAPON:swapon(2):no
95 # 29:AUE_SETHOSTNAME:sethostname(2):no
96 # 30:AUE_FCNTL:fcntl(2):fm
97 # 31:AUE_SETPRIORITY:setpriority(2):no
98 # 32:AUE_CONNECT:connect(2):nt
99 # 33:AUE_ACCEPT:accept(2):nt
100 # 34:AUE_BIND:bind(2):nt
101 # 35:AUE_SETSOCKOPT:setsockopt(2):nt
102 # 36:AUE_VTRACE:vtrace(2):no
103 # 37:AUE_SETTIMEOFDAY:settimeofday(2):no
104 # 38:AUE_FCHOWN:fchown(2):fm
105 # 39:AUE_FCHMOD:fchmod(2):fm
106 # 40:AUE_SETREUID:setreuid(2):pm
107 # 41:AUE_SETREGID:setregid(2):pm
108 # 42:AUE_RENAME:rename(2):fc,fd
109 # 43:AUE_TRUNCATE:truncate(2):no
110 # 44:AUE_FTRUNCATE:ftruncate(2):no
111 # 45:AUE_FLOCK:flock(2):no
112 # 46:AUE_SHUTDOWN:shutdown(2):nt
113 # 47:AUE_MKDIR:mkdir(2):fc
114 # 48:AUE_RMDIR:rmdir(2):fd
115 # 49:AUE_UTIMES:utimes(2):fm
116 # 50:AUE_ADJTIME:adjtime(2):as
117 # 51:AUE_SETRLIMIT:setrlimit(2):ua
118 # 52:AUE_KILLPG:killpg(2):no
119 # 53:AUE_NFS_SVC:nfs_svc(2):no
120 # 54:AUE_STATFS:statfs(2):fa
121 # 55:AUE_FSTATFS:fstatfs(2):fa
122 # 56:AUE_UNMOUNT:unmount(2):no
123 # 57:AUE_ASYNC_DAEMON:async_daemon(2):no
124 # 58:AUE_NFS_GETFH:nfs_getfh(2):no
```

```

125 59:AUE_SETDOMAINNAME:setdomainname(2):no
126 60:AUE_QUOTACTL:quotactl(2):no
127 61:AUE_EXPORTFS:exportfs(2):no
128 62:AUE_MOUNT:mount(2):as
129 # AUE_SEMSYS is a placeholder and will not be generated
130 63:AUE_SEMSYS:semsys(2) - place holder:no
131 # AUE_MSGSYS is a placeholder and will not be generated
132 64:AUE_MSGSYS:msgsys(2) - place holder:no
133 # AUE_SHMSYS is a placeholder and will not be generated
134 65:AUE_SHMSYS:shmsys(2) - place holder:no
135 66:AUE_BSMSYS:bsmsys(2) - place holder:no
136 67:AUE_RFSSYS:rfssys(2) - place holder:no
137 68:AUE_FCHDIR:fchdir(2):pm
138 69:AUE_FCHROOT:fchroot(2):pm
139 70:AUE_VPIXSYS:vpixsys(2) - place holder:no
140 71:AUE_PATHCONF:pathconf(2):fa
141 72:AUE_OPEN_R:open(2) - read:fr
142 73:AUE_OPEN_RC:open(2) - read,creat:fc,fr
143 74:AUE_OPEN_RT:open(2) - read,trunc:fd,fr
144 75:AUE_OPEN_RTC:open(2) - read,creat,trunc:fc,fd,fr
145 76:AUE_OPEN_W:open(2) - write:fw
146 77:AUE_OPEN_WC:open(2) - write,creat:fc,fw
147 78:AUE_OPEN_WT:open(2) - write,trunc:fd,fw
148 79:AUE_OPEN_WTC:open(2) - write,creat,trunc:fc,fd,fw
149 80:AUE_OPEN_RW:open(2) - read,write:fr,fw
150 81:AUE_OPEN_RWC:open(2) - read,write,creat:fc,fw,fr
151 82:AUE_OPEN_RWT:open(2) - read,write,trunc:fd,fr,fw
152 83:AUE_OPEN_RWTC:open(2) - read,write,creat,trunc:fc,fd,fw,fr
153 84:AUE_MSGCTL:msgctl(2) - illegal command:ip
154 85:AUE_MSGCTL_RMID:msgctl(2) - IPC_RMID command:ip
155 86:AUE_MSGCTL_SET:msgctl(2) - IPC_SET command:ip
156 87:AUE_MSGCTL_STAT:msgctl(2) - IPC_STAT command:ip
157 88:AUE_MSGGET:msgget(2):ip
158 89:AUE_MSGRCV:msgrcv(2):ip
159 90:AUE_MSGSND:msgsnd(2):ip
160 91:AUE_SHMCTL:shmctl(2) - illegal command:ip
161 92:AUE_SHMCTL_RMID:shmctl(2) - IPC_RMID command:ip
162 93:AUE_SHMCTL_SET:shmctl(2) - IPC_SET command:ip
163 94:AUE_SHMCTL_STAT:shmctl(2) - IPC_STAT command:ip
164 95:AUE_SHMGET:shmget(2):ip
165 96:AUE_SHMAT:shmat(2):ip
166 97:AUE_SHMDT:shmdt(2):ip
167 98:AUE_SEMCTL:semctl(2) - illegal command:ip
168 99:AUE_SEMCTL_RMID:semctl(2) - IPC_RMID command:ip
169 100:AUE_SEMCTL_SET:semctl(2) - IPC_SET command:ip
170 101:AUE_SEMCTL_STAT:semctl(2) - IPC_STAT command:ip
171 102:AUE_SEMCTL_GETNCNT:semctl(2) - GETNCNT command:ip
172 103:AUE_SEMCTL_GETPID:semctl(2) - GETPID command:ip
173 104:AUE_SEMCTL_GETVAL:semctl(2) - GETVAL command:ip
174 105:AUE_SEMCTL_GETALL:semctl(2) - GETALL command:ip
175 106:AUE_SEMCTL_GETZCNT:semctl(2) - GETZCNT command:ip
176 107:AUE_SEMCTL_SETVAL:semctl(2) - SETVAL command:ip
177 108:AUE_SEMCTL_SETALL:semctl(2) - SETALL command:ip
178 109:AUE_SEMGET:semget(2):ip
179 110:AUE_SEMOP:semop(2):ip
180 111:AUE_CORE:process dumped core:fc
181 112:AUE_CLOSE:close(2):cl
182 113:AUE_SYSTEMBOOT:system booted:na
183 114:AUE_ASYNC_DAEMON_EXIT:async_daemon(2) exited:no
184 115:AUE_NFSSVC_EXIT:nfssvc(2) exited:no
185 116:AUE_PFEEXEC:execve(2) with pfexec enabled:ps,ex,ua,as
186 117:AUE_OPEN_S:open(2) - search:fr
187 118:AUE_OPEN_E:open(2) - exec:fr

189 130:AUE_GETAUID:getauid(2):aa
190 131:AUE_SETAUID:setauid(2):aa

```

```

191 132:AUE_GETAUID:getauid(2):aa
192 133:AUE_SETAUID:setauid(2):aa
193 134:AUE_GETUSERAUDIT:getuseraudit(2):no
194 135:AUE_SETUSERAUDIT:setuseraudit(2):no
195 # AUE_AUDITSVC is a placeholder and will not be generated
196 136:AUE_AUDITSVC:auditsvc(2) - place holder:no
197 # AUE_AUDITON is a placeholder and will not be generated
198 138:AUE_AUDITON:auditon(2) - place holder:no
199 139:AUE_AUDITON_GTERMID:auditon(2) - GETTERMID command:no
200 140:AUE_AUDITON_STERMID:auditon(2) - SETTERMID command:no
201 141:AUE_AUDITON_GPOLICY:auditon(2) - get audit policy flags:aa
202 142:AUE_AUDITON_SPOLICY:auditon(2) - set audit policy flags:as
203 143:AUE_AUDITON_GESTATE:auditon(2) - GESTATE command:no
204 144:AUE_AUDITON_SESTATE:auditon(2) - SESTATE command:no
205 145:AUE_AUDITON_QCTRL:auditon(2) - get queue control parameters:as
206 146:AUE_AUDITON_SQCTRL:auditon(2) - set queue control parameters:as
207 147:AUE_GETKERNSTATE:getkernstate(2):no
208 148:AUE_SETKERNSTATE:setkernstate(2):no
209 149:AUE_GETPORTAUDIT:getportaudit(2):no
210 150:AUE_AUDITSTAT:auditstat(2):no
211 153:AUE_ENTERPROM:enter prom:na
212 154:AUE_EXITPROM:exit prom:na
213 158:AUE_IOCTL:ioctl(2):io
214 173:AUE_ONESIDE:one-sided session record:no
215 174:AUE_MSGGETL:msggetl(2):no
216 175:AUE_MSGRCVL:msgrcvl(2):no
217 176:AUE_MSGSNDL:msgsndl(2):no
218 177:AUE_SEMGETL:semgetl(2):no
219 178:AUE_SHMGETL:shmgetl(2):no
220 183:AUE_SOCKET:socket(2):nt
221 184:AUE_SENDDTO:sendto(2):nt
222 # AUE_PIPE is a potentially very high-volume event, use with caution
223 185:AUE_PIPE:pipe(2):no
224 186:AUE_SOCKETPAIR:socketpair(2):no
225 187:AUE_SEND:send(2):no
226 188:AUE_SENDMSG:sendmsg(2):nt
227 189:AUE_RECV:recv(2):no
228 190:AUE_RECVMSG:recvmsg(2):nt
229 191:AUE_RECVFROM:recvfrom(2):nt
230 # AUE_READ is a potentially very high-volume event, use with caution
231 192:AUE_READ:read(2):no
232 193:AUE_GETDENTS:getdents(2):no
233 194:AUE_LSEEK:lseek(2):no
234 # AUE_WRITE is a potentially very high-volume event, use with caution
235 195:AUE_WRITE:write(2):no
236 196:AUE_WRITEV:writev(2):no
237 197:AUE_NFS:nfs server:no
238 198:AUE_READV:readv(2):no
239 199:AUE_OSTAT:old stat(2):no
240 200:AUE_SETUID:setuid(2):pm
241 201:AUE_STIME:old stime(2):as
242 202:AUE_UTIME:old utime(2):no
243 203:AUE_NICE:old nice(2):pm
244 204:AUE_OSETPRG:old setprgrp(2):no
245 205:AUE_SETGID:old setgid(2):pm
246 206:AUE_READL:readl(2):no
247 207:AUE_READVL:readvl(2):no
248 208:AUE_FSTAT:fstat(2):no
249 209:AUE_DUP2:dup2(2):no
250 # AUE_MMAP is a potentially very high-volume event, use with caution
251 210:AUE_MMAP:mmap(2):no
252 # AUE_AUDIT is a potentially very high-volume event, use with caution
253 211:AUE_AUDIT:audit(2):no
254 212:AUE_PRIOCNLSYS:priocnlsys(2):pm
255 213:AUE_MUNMAP:munmap(2):cl
256 214:AUE_SETEGID:setegid(2):pm

```

```

257 215:AUE_SETEUID:seteuid(2):pm
258 216:AUE_PUTMSG:putmsg(2):nt
259 217:AUE_GETMSG:getmsg(2):nt
260 218:AUE_PUTPMSG:putpmsg(2):nt
261 219:AUE_GETPMSG:getpmsg(2):nt
262 # AUE_AUDITSYS is a placeholder and will not be generated
263 220:AUE_AUDITSYS:audit system calls place holder:no
264 221:AUE_AUDITON_GETKMASK:auditon(2) - get kernel mask:aa
265 222:AUE_AUDITON_SETKMASK:auditon(2) - set kernel mask:as
266 223:AUE_AUDITON_GETCWD:auditon(2) - get current working directory:aa,as
267 224:AUE_AUDITON_GETCAR:auditon(2) - get current active root:aa,as
268 225:AUE_AUDITON_GETSTAT:auditon(2) - get audit statistics:as
269 226:AUE_AUDITON_SETSTAT:auditon(2) - reset audit statistics:as
270 227:AUE_AUDITON_SETUMASK:auditon(2) - set mask per audit uid:as
271 228:AUE_AUDITON_SETSMASK:auditon(2) - set mask per session ID:as
272 229:AUE_AUDITON_GETCOND:auditon(2) - get audit state:aa
273 230:AUE_AUDITON_SETCOND:auditon(2) - set audit state:as
274 231:AUE_AUDITON_GETCLASS:auditon(2) - get event class:aa,as
275 232:AUE_AUDITON_SETCLASS:auditon(2) - set event class:as
276 233:AUE_FUSERS:utssys(2) - fusers:fa
277 234:AUE_STATVFS:statvfs(2):fa
278 235:AUE_XSTAT:xstat(2):no
279 236:AUE_LXSTAT:lxstat(2):no
280 237:AUE_LCHOWN:lchown(2):fm
281 238:AUE_MEMCNTL:memcntl(2):ot
282 239:AUE_SYSINFO:sysinfo(2):as
283 240:AUE_XMKNOD:xmknod(2):no
284 241:AUE_FORK1:fork1(2):ps
285 # AUE_MODCTL is a placeholder and will not be generated
286 242:AUE_MODCTL:modctl(2) system call place holder:no
287 243:AUE_MODLOAD:modctl(2) - load module:as
288 244:AUE_MODUNLOAD:modctl(2) - unload module:as
289 # AUE_MODCONFIG is a place holder and will not be generated
290 245:AUE_MODCONFIG:modctl(2) - no longer generated:no
291 246:AUE_MODADDMJ:modctl(2) - bind module:as
292 247:AUE_SOCKACCEPT:getmsg-accept:nt
293 248:AUE_SOCKCONNECT:putmsg-connect:nt
294 249:AUE_SOCKSEND:putmsg-send:nt
295 250:AUE_SOCKRECEIVE:getmsg-receive:nt
296 251:AUE_ACLSET:acl(2) - SETACL command:fm
297 252:AUE_FACLSET:facl(2) - SETACL command:fm
298 # AUE_DOORFS is a placeholder and will not be generated
299 253:AUE_DOORFS:doorfs(2) - system call place holder:no
300 254:AUE_DOORFS_DOOR_CALL:doorfs(2) - DOOR_CALL:ip
301 255:AUE_DOORFS_DOOR_RETURN:doorfs(2) - DOOR_RETURN:ip
302 256:AUE_DOORFS_DOOR_CREATE:doorfs(2) - DOOR_CREATE:ip
303 257:AUE_DOORFS_DOOR_REVOKE:doorfs(2) - DOOR_REVOKE:ip
304 258:AUE_DOORFS_DOOR_INFO:doorfs(2) - DOOR_INFO:ip
305 259:AUE_DOORFS_DOOR_CRED:doorfs(2) - DOOR_CRED:ip
306 260:AUE_DOORFS_DOOR_BIND:doorfs(2) - DOOR_BIND:ip
307 261:AUE_DOORFS_DOOR_UNBIND:doorfs(2) - DOOR_UNBIND:ip
308 262:AUE_P_ONLINE:p_online(2):as
309 263:AUE_PROCESSOR_BIND:processor_bind(2):as
310 264:AUE_INST_SYNC:inst_sync(2):as
311 265:AUE_SOCKCONFIG:configure socket:nt
312 266:AUE_SETAUDIT_ADDR:setaudit_addr(2):aa
313 267:AUE_GETAUDIT_ADDR:getaudit_addr(2):aa
314 268:AUE_UMOUNT2:umount2(2):as
315 # AUE_FSAT and all AUE_OPENAT_* codes are obsolete and will not be generated
316 269:AUE_FSAT:fsat(2) - place holder:no
317 270:AUE_OPENAT_R:openat(2) - read:no
318 271:AUE_OPENAT_RC:openat(2) - read,creat:no
319 272:AUE_OPENAT_RT:openat(2) - read,trunc:no
320 273:AUE_OPENAT_RTC:openat(2) - read,creat,trunc:no
321 274:AUE_OPENAT_W:openat(2) - write:no
322 275:AUE_OPENAT_WC:openat(2) - write,creat:no

```

```

323 276:AUE_OPENAT_WT:openat(2) - write,trunc:no
324 277:AUE_OPENAT_WTC:openat(2) - write,creat,trunc:no
325 278:AUE_OPENAT_RW:openat(2) - read,write:no
326 279:AUE_OPENAT_RWC:openat(2) - read,write,creat:no
327 280:AUE_OPENAT_RWT:openat(2) - read,write,trunc:no
328 281:AUE_OPENAT_RWTC:openat(2) - read,write,creat,trunc:no
329 282:AUE_RENAMEAT:renameat(2):no
330 283:AUE_FSTATAT:fstatat(2):no
331 284:AUE_FCHOWNAT:fchownat(2):no
332 285:AUE_FUTIMESAT:futimesat(2):no
333 286:AUE_UNLINKAT:unlinkat(2):no
334 287:AUE_CLOCK_SETTIME:clock_settime(3RT):as
335 288:AUE_NTP_ADJTIME:ntp_adjtime(2):as
336 289:AUE_SETPPRIV:setppriv(2):pm
337 290:AUE_MODDEVPLCY:modctl(2) - configure device policy:as
338 291:AUE_MODADDPRIV:modctl(2) - configure additional privilege:as
339 292:AUE_CRYPTADM:kernel cryptographic framework:as
340 293:AUE_CONFIGKSSL:configure kernel SSL:as
341 294:AUE_BRANDSYS:brandsys(2):ot
342 295:AUE_PF_POLICY_ADDRULE:Add IPsec policy rule:as
343 296:AUE_PF_POLICY_DELRULE>Delete IPsec policy rule:as
344 297:AUE_PF_POLICY_CLONE:Clone IPsec policy:as
345 298:AUE_PF_POLICY_FLIP:Flip IPsec policy:as
346 299:AUE_PF_POLICY_FLUSH:Flush IPsec policy rules:as
347 300:AUE_PF_POLICY_ALGS:Update IPsec algorithms:as
348 # AUE_PORTFS is a placeholder and won't be generated.
349 301:AUE_PORTFS:portfs(2) - file events source - place holder:no
350 #
351 302:AUE_LABELSYS_TNRH:tnrh(2) - config TN remote host cache:as
352 303:AUE_LABELSYS_TNRHTP:tnrhtp(2) - config TN remote host template cache:as
353 304:AUE_LABELSYS_TNMLP:tnmlp(2) - config TN multi-level port entry:as
354 #
355 305:AUE_PORTFS_ASSOCIATE:portfs(2) - file events source - PORT_ASSOCIATE:fa
356 306:AUE_PORTFS_DISSOCIATE:portfs(2) - file events source - PORT_DISSOCIATE:fa
357 #
358 307:AUE_SETSID:setsid(2):pm
359 308:AUE_SETPGID:setpgid(2):pm
360 309:AUE_FACCESSAT:faccessat(2):no
361 310:AUE_AUDITON_GETAMASK:auditon(2) - get default user preselection mask:aa
362 311:AUE_AUDITON_SETAMASK:auditon(2) - set default user preselection mask:as
363 312:AUE_PSECFLAGS:psecflags(2) - set process security flags:pm
364 #endif /* ! codereview */
365 #
366 # user level audit events
367 #      2048 - 6143      Reserved
368 #
369 #      6000 - 7999      allocated for Solaris
370 #
371 6144:AUE_at_create:at-create atjob:ua
372 6145:AUE_at_delete:at-delete atjob (at or atrm):ua
373 6146:AUE_at_perm:at-permission:no
374 6147:AUE_cron_invoke:cron-invoke:ua
375 6148:AUE_crontab_create:crontab-crontab created:ua
376 6149:AUE_crontab_delete:crontab-crontab deleted:ua
377 6150:AUE_crontab_perm:crontab-persmission:no
378 6151:AUE_inetd_connect:inetd connect:na
379 6152:AUE_login:login - local:lo
380 6153:AUE_logout:logout:lo
381 6154:AUE_telnet:login - telnet:lo
382 6155:AUE_rlogin:login - rlogin:lo
383 6156:AUE_mountd_mount:mount:na
384 6157:AUE_mountd_umount:unmount:na
385 6158:AUE_rshd:rsh access:lo
386 6159:AUE_su:su:lo
387 6160:AUE_halt_solaris:halt(1m):ss
388 6161:AUE_reboot_solaris:reboot(1m):ss

```

```

389 6162:AUE_rexecd:rexecd:lo
390 6163:AUE_passwd:passwd:lo
391 6164:AUE_rexd:rexd:lo
392 6165:AUE_ftpd:ftp access:lo
393 6166:AUE_init_solaris:init(lm):ss
394 6167:AUE_uadmin_solaris:uadmin(lm):no
395 6168:AUE_shutdown_solaris:shutdown(lb):ss
396 6169:AUE_poweroff_solaris:poweroff(lm):ss
397 6170:AUE_crontab_mod:crontab-modify:ua
398 6171:AUE_ftpd_logout:ftp logout:lo
399 6172:AUE_ssh:login - ssh:lo
400 6173:AUE_role_login:role login:lo
401 6180:AUE_prof_cmd:profile command:ua,as
402 6181:AUE_filesystem_add:add filesystem:as
403 6182:AUE_filesystem_delete:delete filesystem:as
404 6183:AUE_filesystem_modify:modify filesystem:as
405 6184:AUE_network_add:add network attributes:as
406 6185:AUE_network_delete:delete network attributes:as
407 6186:AUE_network_modify:modify network attributes:as
408 6187:AUE_printer_add:add printer:as
409 6188:AUE_printer_delete:delete printer:as
410 6189:AUE_printer_modify:modify printer:as
411 6190:AUE_scheduledjob_add:add scheduled job:ua
412 6191:AUE_scheduledjob_delete:delete scheduled job:ua
413 6192:AUE_scheduledjob_modify:modify scheduled job:ua
414 6193:AUE_serialport_add:add serial port:as
415 6194:AUE_serialport_delete:delete serial port:as
416 6195:AUE_serialport_modify:modify serial port:as
417 6196:AUE_usermgr_add:add user/user attributes:ua
418 6197:AUE_usermgr_delete:delete user/user attributes:ua
419 6198:AUE_usermgr_modify:modify user/user attributes:ua
420 6199:AUE_uauth:authorization used:ua,as
421 6200:AUE_allocate_succ:allocate-device success:ot
422 6201:AUE_allocate_fail:allocate-device failure:ot
423 6202:AUE_deallocate_succ:deallocate-device success:ot
424 6203:AUE_deallocate_fail:deallocate-device failure:ot
425 6205:AUE_listdevice_succ:allocate-list devices success:ot
426 6206:AUE_listdevice_fail:allocate-list devices failure:ot
427 6207:AUE_create_user:create user:no
428 6208:AUE_modify_user:modify user:no
429 6209:AUE_delete_user:delete user:no
430 6210:AUE_disable_user:disable user:no
431 6211:AUE_enable_user:enable user:no
432 6212:AUE_newgrp_login:newgrp login:lo
433 6213:AUE_admin_authenticate:admin login:lo
434 6214:AUE_kadmind_auth:authenticated kadmind request:ua
435 6215:AUE_kadmind_unauth:unauthenticated kadmind req:ua
436 6216:AUE_krb5kdc_as_req:kdc authentication svc request:ap
437 6217:AUE_krb5kdc_tgs_req:kdc tkt-grant svc request:ap
438 6218:AUE_krb5kdc_tgs_req_2ndtktmm:kdc tgs 2ndtkt mismtch:ap
439 6219:AUE_krb5kdc_tgs_req_alt_tgt:kdc tgs issue alt tgt:ap
440 6220:AUE_smserverd:smserverd:ot
441 6221:AUE_screenlock:screenlock - lock:lo
442 6222:AUE_screenunlock:screenlock - unlock:lo
443 6223:AUE_zone_state:zoneadm:ss
444 6224:AUE_inetd_copylim:inetd copylimit:na
445 6225:AUE_inetd_failrate:inetd failrate:na
446 6226:AUE_inetd_ratelimit:inetd ratelimit:na
447 6227:AUE_zlogin:login - zlogin:lo
448 6228:AUE_su_logout:su logout:lo
449 6229:AUE_role_logout:role logout:lo
450 6230:AUE_attach:attach device:ot
451 6231:AUE_detach:detach device:ot
452 6232:AUE_remove:remove/eject device:ot
453 6233:AUE_pool_import:import device into pool:ot
454 6234:AUE_pool_export:export device from pool:ot

```

```

455 6235:AUE_dladm_create_secobj:create network security object:as,cy
456 6236:AUE_dladm_delete_secobj:delete network security object:as,cy
457 6237:AUE_uadmin_shutdown:uadmin(lm) - shutdown:ss
458 6238:AUE_uadmin_reboot:uadmin(lm) - reboot:ss
459 6239:AUE_uadmin_dump:uadmin(lm) - dump:ss
460 6240:AUE_uadmin_freeze:uadmin(lm) - freeze:ss
461 6241:AUE_uadmin_remount:uadmin(lm) - remount:ss
462 6242:AUE_uadmin_ftrace:uadmin(lm) - ftrace:ss
463 6243:AUE_uadmin_swapctl:uadmin(lm) - swapctl:ss
464 6244:AUE_smbd_session:smbd(lm) session setup:lo
465 6245:AUE_smbd_logoff:smbd(lm) session logoff:lo
466 6246:AUE_vscan_quarantine:vscand(lm) quarantine infected file:na
467 6247:AUE_ndmp_connect:ndmp connect:na
468 6248:AUE_ndmp_disconnect:ndmp disconnect:na
469 6249:AUE_ndmp_backup:ndmp backup:na
470 6250:AUE_ndmp_restore:ndmp restore:na
471 6251:AUE_cpu_ondemand:set ondemand CPU freq governor:ss
472 6252:AUE_cpu_performance:set max CPU freq governor:ss
473 6253:AUE_cpu_threshold:set CPU freq threshold:ss
474 6254:AUE_uadmin_thaw:uadmin(lm) - thaw after freeze:ss,na
475 6255:AUE_uadmin_config:uadmin(lm) - config:ss

477 #
478 # SMF(5) svc.configd events (svcadm(1M) related)
479 #
480 6260:AUE_smf_enable:persistently enable service instance:ss
481 6261:AUE_smf_tmp_enable:temporarily enable service instance:ss
482 6262:AUE_smf_disable:persistently disable service instance:ss
483 6263:AUE_smf_tmp_disable:temporarily disable service instance:ss
484 6264:AUE_smf_restart:restart service instance:ss
485 6265:AUE_smf_refresh:refresh service instance:ss
486 6266:AUE_smf_clear:clear service instance state:ss
487 6267:AUE_smf_degrade:set service instance degraded state:ss
488 6268:AUE_smf_immediate_degrade:immediately set service instance degraded state:ss
489 6269:AUE_smf_maintenance:set service instance persistent maintenance state:ss
490 6270:AUE_smf_immediate_maintenance:immediately set service instance persistent m
491 6271:AUE_smf_imtmp_maintenance:immediately set service instance temporary maint
492 6272:AUE_smf_tmp_maintenance:set service instance maintenance temporary state:ss
493 6273:AUE_smf_milestone:set service management facility milestone:ss
494 #
495 # SMF(5) svc.configd miscellaneous events
496 #
497 6275:AUE_smf_read_prop:read restricted access property value:as
498 #
499 # SMF(5) svc.configd events (svccfg(1M) related)
500 #
501 6280:AUE_smf_create:create service instance object:as
502 6281:AUE_smf_delete:delete service instance object:as
503 6282:AUE_smf_create_pg:create persistent service property group:as
504 6283:AUE_smf_create_npg:create non-persistent service property group:as
505 6284:AUE_smf_delete_pg:delete persistent service property group:as
506 6285:AUE_smf_delete_npg:delete non-persistent service property group:as
507 6286:AUE_smf_create_snap:create repository snapshot:as
508 6287:AUE_smf_delete_snap:delete repository snapshot:as
509 6288:AUE_smf_attach_snap:attach repository snapshot:as
510 6289:AUE_smf_annotation:annotate transaction:as,ss
511 6290:AUE_smf_create_prop:create service instance property:as
512 6291:AUE_smf_change_prop:change service instance property:as
513 6292:AUE_smf_delete_prop:delete service instance property:as
514 #
515 # nwamd(1M) events
516 #
517 6300:AUE_nwam_enable:enable nwam profile object:ss
518 6301:AUE_nwam_disable:disable nwam profile object:ss
519 #
520 # ilbd(1M) events

```

```

521 #
522 6310:AUE_ilb_create_healthcheck:create ILB health check:as
523 6311:AUE_ilb_delete_healthcheck:delete ILB health check:as
524 6312:AUE_ilb_create_rule:create ILB rule:as
525 6313:AUE_ilb_delete_rule:delete ILB rule:as
526 6314:AUE_ilb_disable_rule:disable ILB rule:as
527 6315:AUE_ilb_enable_rule:enable ILB rule:as
528 6316:AUE_ilb_add_server:add ILB server:as
529 6317:AUE_ilb_disable_server:disable ILB server:as
530 6318:AUE_ilb_enable_server:enable ILB server:as
531 6319:AUE_ilb_remove_server:remove ILB server:as
532 6320:AUE_ilb_create_servergroup:create ILB server group:as
533 6321:AUE_ilb_delete_servergroup:delete ILB server group:as
534 #
535 # netcfgd(1M) events
536 #
537 6330:AUE_netcfg_update:create or modify configuration object:ss
538 6331:AUE_netcfg_remove:remove configuration object from repository:ss
539 #
540 # TCSD(8) events
541 #
542 6400:AUE_tpm_takeownership:take ownership of TPM:as
543 6401:AUE_tpm_clearowner:clear ownership of TPM:as
544 6402:AUE_tpm_setoperatorauth:set TPM operator authorization:as
545 6403:AUE_tpm_setownerinstall:set TPM ownership flag:as
546 6404:AUE_tpm_selftestfull:test all TPM protected capabilities:as
547 6405:AUE_tpm_certifyselftest:perform full TPM self-test:as
548 6406:AUE_tpm_continueselftest:complete TPM self-test:as
549 6407:AUE_tpm_ownersetdisable:change the status of TPM disable flag:as
550 6408:AUE_tpm_ownerclear:perform the clear operation under TPM owner auth:as
551 6409:AUE_tpm_disableownerclear:disable TPM OwnerClear command permanently:as
552 6410:AUE_tpm_forceclear:perform TPM clear operation under physical access:as
553 6411:AUE_tpm_disableforceclear:disable ForceClear execution until next startup:a
554 6412:AUE_tpm_physicaldisable:disable TPM physical presence:as
555 6413:AUE_tpm_physicalenable:enable TPM physical presence:as
556 6414:AUE_tpm_physicaldeactivate:set TPM deactivated flag:as
557 6415:AUE_tpm_settempdeactivated:set volatile TPM deactivated flag to TRUE:as
558 6416:AUE_tpm_settempdeactivated2:set volatile TPM deactivated flag TRUE with aut
559 6417:AUE_tpm_physicalpresence:set the TPM physical presence flag:as
560 6418:AUE_tpm_fieldupgrade:update TPM protected capabilities:as
561 6419:AUE_tpm_resetlockvalue:reset TPM failed authorization attempt lock:as
562 #
563 # hotplugd(1M) events
564 #
565 6500:AUE_hotplug_state:change hotplug connection state:ss
566 6501:AUE_hotplug_set:set hotplug bus private options:ss

568 #
569 # Trusted Extensions events:
570 #
571 9035:AUE_sl_change:Workspace label change:ap
572 9036:AUE_file_relabel:relabel file:fm
573 9037:AUE_file_copy:file copy:fm
574 9038:AUE_file_move:file move:no
575 9039:AUE_sel_xfer_mgr_xfer:selection manager transfer:fm
576 9101:AUE_ClientConnect:client connection to x server:lo
577 9102:AUE_ClientDisconnect:client disconn. from x server:lo
578 9120:AUE_ChangeProperty:XChangeProperty(3X11):xc
579 9121:AUE_DeleteProperty:XDeleteProperty(3X11):xc
580 9137:AUE_GrabServer:XGrabServer(3X11):ot
581 9138:AUE_UngrabServer:XUngrabServer(3X11):ot
582 9146:AUE_SetFontPath:XSetFontPath(3X11):ot
583 9173:AUE_InstallColormap:XInstallColormap(3X11):ot
584 9174:AUE_UninstallColormap:XUninstallColormap(3X11):xp
585 9193:AUE_SetScreenSaver:XSetScreenSaver(3X11):xp
586 9194:AUE_ChangeHosts:XChangeHosts(3X11):ot

```

```

587 9195:AUE_SetAccessControl:XSetAccessControl(3X11):xp
588 9196:AUE_SetCloseDownMode:XSetCloseDownMode(3X11):xs
589 9197:AUE_KillClient:XKillClient(3X11):xc
590 9202:AUE_XExtensions:X server extensions:xp
591 9103:AUE_CreateWindow:XCreateWindow(3X11):xc
592 9104:AUE_ChangeWindowAttributes:XChangeWindowAttributes(3X11):xp
593 9105:AUE_GetWindowAttributes:XGetWindowAttributes(3X11):xp
594 9106:AUE_DestroyWindow:XDestroyWindow(3X11):xc
595 9107:AUE_DestroySubwindows:XDestroySubwindows(3X11):xc
596 9108:AUE_ChangeSaveSet:XChangeSaveSet(3X11):xp
597 9109:AUE_ReparentWindow:XReparentWindow(3X11):xp
598 9110:AUE_MapWindow:XMapWindow(3X11):xp
599 9111:AUE_MapSubwindows:XMapSubwindows(3X11):xp
600 9112:AUE_UnmapWindow:XUnmapWindow(3X11):xp
601 9113:AUE_UnmapSubwindows:XUnmapSubwindows(3X11):xp
602 9114:AUE_ConfigureWindow:XConfigureWindow(3X11):xp
603 9115:AUE_CirculateWindow:XCirculateWindow(3X11):xp
604 9116:AUE_GetGeometry:XGetGeometry(3X11):xp
605 9117:AUE_QueryTree:XQueryTree(3X11):xp
606 9118:AUE_InternAtom:XInternAtom(3X11):xs
607 9119:AUE_GetAtomName:XGetAtomName(3X11):xs
608 9122:AUE_GetProperty:XGetProperty(3X11):xp
609 9123:AUE_ListProperties:XListProperties(3X11):xp
610 9124:AUE_SetSelectionOwner:XSetSelectionOwner(3X11):xp
611 9125:AUE_GetSelectionOwner:XGetSelectionOwner(3X11):xs
612 9126:AUE_ConvertSelection:XConvertSelection(3X11):xs
613 9127:AUE_SendEvent:XSendEvent(3X11):xs
614 9128:AUE_GrabPointer:XGrabPointer(3X11):xs
615 9129:AUE_UngrabPointer:XUngrabPointer(3X11):xs
616 9130:AUE_GrabButton:XGrabButton(3X11):xp
617 9131:AUE_UngrabButton:XUngrabButton(3X11):xs
618 9132:AUE_ChangeActivePointerGrab:XChangeActivePointerGrab(3X11):xs
619 9133:AUE_GrabKeyboard:XGrabKeyboard(3X11):xp
620 9134:AUE_UngrabKeyboard:XUngrabKeyboard(3X11):xs
621 9135:AUE_GrabKey:XGrabKey(3X11):xp
622 9136:AUE_UngrabKey:XUngrabKey(3X11):xp
623 9139:AUE_QueryPointer:XQueryPointer(3X11):xp
624 9140:AUE_GetMotionEvents:XGetMotionEvents(3X11):xp
625 9141:AUE_TranslateCoords:XTranslateCoords(3X11):xp
626 9142:AUE_WarpPointer:XWarpPointer(3X11):xs
627 9143:AUE_SetInputFocus:XSetInputFocus(3X11):xs
628 9144:AUE_GetInputFocus:XGetInputFocus(3X11):xs
629 9145:AUE_QueryKeymap:XQueryKeymap(3X11):xp
630 9147:AUE_FreePixmap:XFreePixmap(3X11):xc
631 9148:AUE_ChangeGC:XChangeGC(3X11):xp
632 9149:AUE_CopyGC:XCopyGC(3X11):xp
633 9150:AUE_SetDashes:XSetDashes(3X11):xp
634 9151:AUE_SetClipRectangles:XSetClipRectangles(3X11):xp
635 9152:AUE_FreeGC:XFreeGC(3X11):xc
636 9153:AUE_ClearArea:XClearArea(3X11):xp
637 9154:AUE_CopyArea:XCopyArea(3X11):xs
638 9155:AUE_CopyPlane:XCopyPlane(3X11):xs
639 9156:AUE_PolyPoint:XPolyPoint(3X11):xp
640 9157:AUE_PolyLine:XPolyLine(3X11):xp
641 9158:AUE_PolySegment:XPolySegment(3X11):xp
642 9159:AUE_PolyRectangle:XPolyRectangle(3X11):xs
643 9160:AUE_PolyArc:XPolyArc(3X11):xp
644 9161:AUE_FillPolygon:XFillPolygon(3X11):xp
645 9162:AUE_PolyFillRectangle:XPolyFillRectangle(3X11):xp
646 9163:AUE_PolyFillArc:XPolyFillArc(3X11):xp
647 9164:AUE_PutImage:XPutImage(3X11):xp
648 9165:AUE_GetImage:XGetImage(3X11):xs
649 9166:AUE_PolyText8:XPolyText8(3X11):xp
650 9167:AUE_PolyText16:XPolyText16(3X11):xp
651 9168:AUE_ImageText8:XImageText8(3X11):xp
652 9169:AUE_ImageText16:XImageText16(3X11):xp

```

```
653 9170:AUE_CreateColormap:XCreateColormap(3X11):xc
654 9171:AUE_FreeColormap:XFreeColormap(3X11):xc
655 9172:AUE_CopyColormapAndFree:XCOPYColormapAndFree(3X11):xp
656 9175:AUE_ListInstalledColormaps:XListInstalledColormaps(3X11):xs
657 9176:AUE_AllocColor:XAllocColor(3X11):xc
658 9177:AUE_AllocNamedColor:XAllocNamedColor(3X11):xc
659 9178:AUE_AllocColorCells:XAllocColorCells(3X11):xc
660 9179:AUE_AllocColorPlanes:XAllocColorPlanes(3X11):xc
661 9180:AUE_FreeColors:XFreeColors(3X11):xc
662 9181:AUE_StoreColors:XStoreColors(3X11):xp
663 9182:AUE_StoreNamedColor:XStoreNamedColor(3X11):xp
664 9183:AUE_QueryColors:XQueryColors(3X11):xp
665 9184:AUE_LookupColor:XLookupColor(3X11):xp
666 9185:AUE_CreateCursor:XCreateCursor(3X11):xc
667 9186:AUE_CreateGlyphCursor:XCreateGlyphCursor(3X11):xc
668 9187:AUE_FreeCursor:XFreeCursor(3X11):xc
669 9188:AUE_RecolorCursor:XRecolorCursor(3X11):xp
670 9189:AUE_ChangeKeyboardMapping:XChangeKeyboardMapping(3X11):xs
671 9190:AUE_ChangeKeyboardControl:XChangeKeyboardControl(3X11):xs
672 9191:AUE_Bell:XBell(3X11):xs
673 9192:AUE_ChangePointerControl:XChangePointerControl(3X11):xs
674 9198:AUE_RotateProperties:XRotateProperties(3X11):xp
675 9199:AUE_ForceScreenSaver:XForceScreenSaver(3X11):xp
676 9200:AUE_SetPointerMapping:XSetPointerMapping(3X11):xs
677 9201:AUE_SetModifierMapping:XSetModifierMapping(3X11):xs
```

```

*****
31126 Thu Jun 30 22:02:04 2016
new/usr/src/lib/libbssm/audit.xml
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
_____unchanged_portion_omitted_____

```

```

575 sub generateTableC {
576     my $event = shift;
577     my $eventId = shift;
578     my $eventType = shift;
579     my $eventHeader = shift;
580     my $omit = shift;

582     my %tokenType = (
583         #
584         #     tokenTypes are the ones that are actually defined
585         #     for use in adt.xml audit records
586         #
588         #     'acl'           => 'AUT_ACL',           # not defined
589         #     'arbitrary'    => 'AUT_ARBITRARY',      # not defined
590         #     'arg'          => 'AUT_ARG',           # not defined
591         #     'attr'         => 'AUT_ATTR',          # not defined
592         #     'command'      => 'AUT_CMD',           # not defined
593         #     'command_alt'  => 'ADT_CMD_ALT',        # dummy token id
594         #     'date'         => 'AUT_TEXT',           # not used
595         #     'exec_args'    => 'AUT_EXEC_ARGS',      # not defined
596         #     'exec_env'     => 'AUT_EXEC_ENV',       # not defined
597         #     'exit'         => 'AUT_EXIT',           # not defined
598         #     'fmri'         => 'AUT_FMRI',           # not defined
599         #     'groups'       => 'AUT_GROUPS',         # not defined
600         #     'header'       => 'AUT_HEADER',         # not defined
601         #     'in_peer'      => 'ADT_IN_PEER',        # dummy token id
602         #     'in_remote'    => 'ADT_IN_REMOTE',      # dummy token id
603         #     'ipc'          => 'AUT_IPC',           # not defined
604         #     'ipc_perm'     => 'AUT_IPC_PERM',       # not defined
605         #     'ipport'       => 'AUT_IPORT',          # not defined
606         #     'label'        => 'AUT_LABEL',          # not defined
607         #     'newgroups'    => 'AUT_NEWGROUPS',      # not defined
608         #     'opaque'       => 'AUT_OPAQUE',         # not defined
609         #     'path'         => 'AUT_PATH',           # not defined
610         #     'path_list'    => '-AUT_PATH',         # dummy token id
611         #     'process'      => 'AUT_PROCESS',        # not defined
612         #     'priv_effective' => 'ADT_AUT_PRIV_E',    # dummy token id
613         #     'priv_limit'   => 'ADT_AUT_PRIV_L',     # dummy token id
614         #     'priv_inherit' => 'ADT_AUT_PRIV_I',     # dummy token id
615         #     'return'       => 'AUT_RETURN',        # not defined
616         #     'secflags'     => 'AUT_SECFLAGS',       # not defined
617 #endif /* ! codereview */
618         #     'seq'          => 'AUT_SEQ',           # not defined
619         #     'socket'       => 'AUT_SOCKET',         # not defined
620         #     'socket-inet'  => 'AUT_SOCKET_INET',    # not defined
621         #     'subject'      => 'AUT_SUBJECT',        # not defined
622         #     'text'         => 'AUT_TEXT',           # not defined
623         #     'tid'          => 'AUT_TID',            # not defined
624         #     'trailer'      => 'AUT_TRAILER',        # not defined
625         #     'uauth'        => 'AUT_UAUTH',          # not defined
626         #     'user'         => 'AUT_USER',           # not defined
627         #     'zonename'     => 'AUT_ZONENAME',       # not defined
628         );

630     my @xlateEntryList = ();

```

```

632     my $external = $event->getExternal();
633     my $internal = $event->getInternal();

635     unless ($external) {
636         print STDERR "No external object captured for event $eventId\n";
637         return;
638     }
639     if ($eventType) {
640         $nameTranslation{$eventId} = $eventId;
641     } else {
642         $nameTranslation{$eventId} = $external->getInternalName();
643     }
644     unless ($internal) {
645         print STDERR "No internal object captured for event $eventId\n";
646         return;
647     }
648     my @entryRef = $internal->getEntries();
649     my $entryRef;
650     my @tokenOrder = ();
651     my $firstTokenIndex = 0; # djdj not used yet, djdj BUG!
652                             # needs to be used by translate table

654     if ($internal->isReorder()) { # prescan the entry list to get the token orde
655         my @inputOrder;
656         foreach $entryRef (@entryRef) {
657             my ($intEntry, $entry) = @$entryRef;
658             push (@inputOrder, $intEntry->getAttr('order'));
659         }

661         my $i; # walk down the inputOrder list once
662         my $k = 1; # discover next in line
663         my $l = 0; # who should point to next in line
664         for ($i = 0; $i <= $#inputOrder; $i++) {
665             my $j;
666             for ($j = 0; $j <= $#inputOrder; $j++) {
667                 if ($k == $inputOrder[$j]) {
668                     if ($k == 1) {
669                         $firstTokenIndex = $j;
670                     } else {
671                         $tokenOrder[$l] = "&(selfReference[$j])";
672                     }
673                     $l = $j;
674                     last;
675                 }
676             }
677             $k++;
678         }
679         $tokenOrder[$l] = 'NULL';
680     }
681     else { # default order -- input order same as output
682         my $i;
683         my $j;
684         for ($i = 0; $i < $#entryRef; $i++) {
685             my $j = $i + 1;
686             $tokenOrder[$i] = "&(selfReference[$j])";
687         }
688         $tokenOrder[$#entryRef] = 'NULL';
689     }

691     my $sequence = 0;
692     foreach $entryRef (@entryRef) {
693         my ($intEntry, $entry) = @$entryRef;
694         my $entryId = $entry->getAttr('id');

696         my ($extEntry, $unusedEntry, $tokenId) =

```



```

697     $external->getEntry($entryId);
698     my $opt = $extEntry->getAttr('opt');

700     if ($opt eq 'none') {
701         if (defined ($doc->getToken($tokenId))) {
702             if (defined ($tokenType{$tokenId})) {
703                 $tokenId = $tokenType{$tokenId};
704             }
705             else {
706                 print STDERR "token id $tokenId not implemented\n";
707             }
708         }
709         else {
710             print STDERR "token = $tokenId is undefined\n";
711             $tokenId = 'error';
712         }
713         my ($xlate, $jni) =
714             formatTableEntry ('', $tokenId, $eventId, '', 0, 0,
715                 $tokenOrder[$sequence], 'NULL', '', $omit);
716         push (@xlateEntryList, $xlate);
717     }
718     else {
719         my $dataType = $extEntry->getAttr('type');
720         $dataType =~ s/\s+//g; # remove blanks (char * => char*)

722         my $enumGroup = '';
723         if ($dataType =~ /^msg/i) {
724             $enumGroup = $dataType;
725             $enumGroup =~ s/^msg\s*//i;
726             $enumGroup = "${pfx_adt}_". $enumGroup;
727         }
728         my $required = ($opt eq 'required') ? 1 : 0;
729         my $tsol = 0;
730         my $tokenId = $intEntry->getAttr('token');
731         my $token;
732         my $tokenName;
733         my $tokenFormat = $intEntry->getAttr('format');
734         if (defined ($tokenFormat)) {
735             $tokenFormat = "\"$tokenFormat\"";
736         }
737         else {
738             $tokenFormat = 'NULL';
739         }

741         if (defined ($token = $doc->getToken($tokenId))) {
742             $tsol = (lc $token->getUsage() eq 'tsol') ? 1 : 0;
743             if (defined ($tokenType{$tokenId})) {
744                 $tokenName = $tokenType{$tokenId};
745             }
746             else {
747                 print STDERR "token id $tokenId not implemented\n";
748             }
749         }
750         else {
751             print STDERR
752                 "$tokenId is an unimplemented token ($entryId in $eventId)\n";
753             $tokenName = 'AUT_TEXT';
754         }
755         my ($xlate, $jni) =
756             formatTableEntry($entryId, $tokenName, $eventId, $dataType, $required,
757                 $tsol, $tokenOrder[$sequence], $tokenFormat,
758                 $enumGroup, $omit);
759         push (@xlateEntryList, $xlate);
760     }
761     $sequence++;
762 }

```

```

763     $xlateEventTable{$eventId} = [\@xlateEntryList, $eventId, $firstTokenIndex
764         $eventHeader];
765 }

767 sub formatTableEntry {
768     my ($id, $token, $eventId, $type, $required, $tsol, $sequence, $format,
769         $enumGroup, $omitEntry) = @_;

772     # does this map belong in the xml source? (at least the defaults?)
773     # fill in the default value only if it is other than zero.
774     #         base type          adt name,      default value
775     my %entryDef = ( 'au_asid_t'   => ['ADT_UINT32',      ''],
776         'uint_t'       => ['ADT_UINT32',      ''],
777         'int'          => ['ADT_INT',         ''],
778         'int32_t'      => ['ADT_INT32',      ''],
779         'uid_t'        => ['ADT_UID',         'AU_NOAUDITID'],
780         'gid_t'        => ['ADT_GID',         'AU_NOAUDITID'],
781         'uid_t*'       => ['ADT_UIDSTAR',    ''],
782         'gid_t*'       => ['ADT_GIDSTAR',    ''],
783         'char'         => ['ADT_CHAR',       ''],
784         'char*'        => ['ADT_CHARSTAR',   ''],
785         'char**'       => ['ADT_CHAR2STAR',  ''],
786         'long'         => ['ADT_LONG',       ''],
787         'pid_t'        => ['ADT_PID',        ''],
788         'priv_set_t*'  => ['ADT_PRIVSTAR',   ''],
789         'ulong_t'      => ['ADT_ULONG',     ''],
790         'uint16_t'     => ['ADT_UINT16',    ''],
791         'uint32_t'     => ['ADT_UINT32',    ''],
792         'uint32_t*'    => ['ADT_UINT32STAR', ''],
793         'uint32_t[[]]' => ['ADT_UINT32ARRAY', ''],
794         'uint64_t'     => ['ADT_UINT64',    ''],
795         'uint64_t*'    => ['ADT_UINT64STAR', ''],
796         'm_label_t*'  => ['ADT_MLABELSTAR', ''],
797         'fd_t'         => ['ADT_FD',        '-1'],
798     );
799     my $xlateLabel = $uniLabel.$xlateUniLabelInc;
800     my $xlateLabelInc = 0;
801     my $xlateLine = '';
802     my @jniLine = ();

804     # the list handling should be a simple loop with a loop of one
805     # falling out naturally.

807     unless ($type =~ /\./) { # if list, then generate sequence of entries
808         my $dataType;
809         my $dataSize;
810         my $xlateLabelRef = '';

812         my $arraySize = '';
813         $arraySize = $1 if ($type =~ s/[(\d+)\]/[[]]/);

815         my $entryType = ${entryDef{$type}}[0];

817         my @xlateType = (); # for adt_xlate.c
818         my $typeCount = 1;

820         if ($entryType) {
821             $dataType = $entryType;
822             $type =~ s/([*+])\s*(\d+)$/1 $2/;
823             $type =~ s/[[]//;
824             $dataSize = "sizeof ($type)";
825             if ($arraySize) {
826                 $dataSize = "$arraySize * ". $dataSize;
827             }
828             $xlateLine = "${dataType, $dataSize}";

```

```

829   push (@jniLine, [$id, $dataType, $format, $enumGroup, $required]);
830   } elseif ($type eq '') {
831     $xlateLabelRef = 'NULL';
832   } elseif ($type =~ /^msg/i) {
833     $type =~ s/^msg//i;
834     $dataType = 'ADT_MSG';
835     my $dataEnum = 'ADT_LIST_' . uc $type;
836     $xlateLine = "{{{$dataType, $dataEnum}}";
837     push (@jniLine, [$id, $dataType, $format, $enumGroup, $required]);
838   } elseif ($type =~ /time_t/i) {
839     $dataType = 'ADT_DATE';
840     $dataSize = "sizeof (time_t)";
841     $xlateLine = "{{{$dataType, $dataSize}}";
842     push (@jniLine, [$id, $dataType, $format, $enumGroup, $required]);
843   } elseif ($type =~ /termid/i) {
844     $dataType = 'ADT_TERMIDSTAR';
845     $dataSize = "sizeof (au_tid_addr_t *)";
846     $xlateLine = "{{{$dataType, $dataSize}}";
847     push (@jniLine, [$id, $dataType, $format, $enumGroup, $required]);
848   } elseif (uc $omitEntry eq 'JNI') {
849     $xlateLabelRef = 'NULL';
850   } else {
851     print STDERR "$type is not an implemented data type\n";
852     $xlateLabelRef = 'NULL';
853   }
854   if ($xlateLine && !($xlateTypeList{$xlateLine})) {
855     $xlateTypeList{$xlateLine} = $xlateLabel;
856     push (@xlateTypeList, "datadef\t$xlateLabel\[\1\] = \t$xlateLine;");
857     $xlateLabelInc = 1;
858   } else {
859     $xlateLabel = $xlateTypeList{$xlateLine};
860   }
861   $xlateLabelRef = '&' . $xlateLabel . '[0]'
862   unless $xlateLabelRef eq 'NULL';

864   # "EOL" is where a comma should go unless end of list
865   $xlateLine = "{$token, \t1, \t$xlateLabelRef, \t$sequence, \n" .
866     "\t\t0, \t$required, \t$tsol, \t$format}EOL";
867
868   if (uc $omitEntry ne 'ALWAYS' && ${$entryDef{$type}}[1]) {
869     my @list = ();
870     if ($xlateDefault{$seventId}) {
871       @list = @{$xlateDefault{$seventId}};
872     } else {
873       push (@xlateDefaults, $seventId);
874     }
875     push (@list, $id, ${$entryDef{$type}}[1]);
876     $xlateDefault{$seventId} = \@list;
877   }
878   } else { # is a list
879     my @type = split(/,/, $type);
880     my @arraySize = ();
881     my @id = split(/,/, $id);
882     my @jniId = @id;
883     my $dataType;
884     my $typeCount = (#$type + 1);
885     my @xlateType = ();
886     my @default = ();

888     foreach my $dtype (@type) {
889       my $jniId = shift @jniId;
890       my $id = shift @id;
891       my $arraySize = '';
892       $arraySize = $!1 if ($dtype =~ s/[^\d+]/[//]);

894       my $entryType = ${$entryDef{$dtype}}[0];

```

```

895     if ($entryType) {
896       my $type = $dtype;
897       $type =~ s/[^\d+]/[//];
898       $type =~ s/[^\d+]/[//];

900       my $sizeString = "sizeof";
901       $sizeString = "$arraySize * " . $sizeString if $arraySize;
902       push (@xlateType, "\${$entryType, $sizeString ($type)\}");
903       push (@jniLine, [$jniId, $entryType, $format, $enumGroup, $required]);
904     } elseif ($type =~ /^msg/i) {
905       $type =~ s/^msg//i;
906       $dataType = 'ADT_MSG';
907       my $dataEnum = 'ADT_LIST_' . uc $type;
908       push (@xlateType, "\${$dataType, $dataEnum\}");
909       push (@jniLine, [$jniId, $dataType, $format, $enumGroup, $required]);
910     } elseif ($type =~ /time_t/i) {
911       $dataType = 'ADT_DATE';
912       push (@xlateType, "\${$entryType, sizeof ($type)\}");
913       push (@jniLine, [$jniId, $entryType, $format, $enumGroup, $required]);
914     } elseif ($type =~ /termid/i) {
915       $dataType = 'ADT_TERMIDSTAR';
916       push (@xlateType, "\${$dataType, sizeof (au_tid_addr_t)\}");
917       push (@jniLine, [$jniId, $dataType, $format, $enumGroup, $required]);
918     } elseif (uc $omitEntry eq 'JNI') {
919       # nothing to do.
920     } else {
921       print STDERR "$dtype is not an implemented data type\n";
922     }
923     if (uc $omitEntry ne 'ALWAYS' && ${$entryDef{$dtype}}[1]) {
924       push (@default, $id, ${$entryDef{$dtype}}[1]);
925     }
926   }
927   my $xlateArray = "\[$typeCount\] = \t{" . join(", \n\t\t\t", @xlateType) .
928
929   unless ($xlateTypeList{$xlateArray}) {
930     $xlateTypeList{$xlateArray} = $xlateLabel;
931     $xlateArray = "datadef\t$xlateLabel" . $xlateArray;
932     push (@xlateTypeList, $xlateArray);
933     $xlateLabelInc = 1;
934   } else {
935     $xlateLabel = $xlateTypeList{$xlateArray};
936   }
937   $xlateLine =
938     "{$token, \t$typeCount, \t$xlateLabel\[\0\], \t$sequence, \n" .
939     "\t\t0, \t$required, \t$tsol, \t$format}EOL";
940   if (@default) {
941     my @list = ();
942     if ($xlateDefault{$seventId}) {
943       @list = @{$xlateDefault{$seventId}};
944     } else {
945       push (@xlateDefaults, $seventId);
946     }
947     push (@list, @default);
948     $xlateDefault{$seventId} = \@list;
949   }
950   }
951   $xlateUniLabelInc++ if $xlateLabelInc;
952   return ($xlateLine, \@jniLine);
953 }

955 sub generateAPIFile {
956   my $event = shift;
957   my $seventId = shift;
958   my $seventType = shift;
959   my $seventHeader = shift;
960   my $idNo = shift;

```

```

962 my @entryList = ();
964 my $external = $event->getExternal();
966 if ($eventType && $debug) {
967     print STDERR "event $eventId is of type $eventType\n";
968 }
970 return unless $external;
972 my ($extEntry, $entry, $tokenId, $format);
973 while (($extEntry, $entry, $tokenId, $format) = $external->getNextEntry()) {
974     last unless $entry;
975     my $entryId = $entry->getAttr('id');
977     unless (defined $entryId) {
978         print STDERR "undefined entry id for external $eventId\n";
979         next;
980     }
981     my $option = $extEntry->getAttr('opt');
982     next if ($option eq 'none');
984     if (defined (my $token = $doc->getToken($tokenId))) {
985         $option = 'Trusted Solaris only'
986         if (lc $token->getUsage() eq 'tsol') ? 1 : 0;
987     }
988     $option .= " (format: $format)" if $format;
990     my $dataType = $extEntry->getAttr('type');
991     unless (defined $dataType) {
992         print STDERR "no type defined for external tag for $eventId\n";
993         $dataType = "error";
994     }
996     my $comment = $entry->getContent();
998     if (($dataType =~ /,/) || ($entryId =~ /,/)) {
999         my @type = split(/\s*,\s*/, $dataType);
1000         my @id = split(/\s*,\s*/, $entryId);
1001         if ($#type != $#id) {
1002             print STDERR
1003                 "number of data types ($dataType) does not match number of ids ($e
1004                 " for event $eventId\n";
1005             if ($#type < $#id) {
1006                 $#id = $#type;
1007             }
1008             else {
1009                 $#type = $#id;
1010             }
1011         }
1013         my $i;
1014         my $line = '';
1015         $line = "/* $comment */\n\t" if defined $comment;
1016         for ($i = 0; $i <= $#type; $i++) {
1017             my ($primitive, $dereference) =
1018                 ($type[$i] =~ /^[^*]+\s*(\**)/);
1019             $id[$i] .= $i if ($primitive =~ s/([\d+\\])/);
1020             $line .= "$primitive\t$dereference$id[$i];\t/* $option */";
1021             push (@entryList, $line);
1022             $line = '';
1023         }
1024     }
1025     else {
1026         my $line = '';

```

```

1027     $line = "/* $comment */\n\t" if defined $comment;
1028     if ($dataType =~ /^msg/i) {
1029         $dataType =~ s/^msg\s*//i;
1030         $line .= "enum ${pfx_adt}_$dataType" . "\t$entryId;\t/* $option
1031     }
1032     elsif ($dataType =~ /time_t/i) {
1033         $line .= "time_t\t$entryId;\t/* $option */";
1034     }
1035     else {
1036         my ($primitive, $dereference) =
1037             ($dataType =~ /^[^*]+\s*(\**)/);
1038         $entryId .= $i if ($primitive =~ s/([\d+\\])/);
1039         $line .= "$primitive\t$dereference$entryId;\t/* $option */";
1040     }
1041     push (@entryList, $line);
1042 }
1043 }
1044 $eventExtra{$eventId} = [$eventHeader, $idNo];
1045 $eventAPI{$eventId} = \@entryList;
1046 }
1048 sub generateMsgLists {
1049     my $textList = shift;
1051     my $textName = $textList->getId();
1052     my $header = $textList->getHeader();
1053     my $start = $textList->getMsgStart();
1054     my $public = $textList->getMsgPublic();
1055     my $deprecated = $textList->getDeprecated();
1057     addHeader($header);
1058     print "$textName starts at $start\n" if $debug;
1060     my $entry;
1061     my @entry;
1062     while ($entry = $textList->getNextMsg()) {
1063         if ($debug) {
1064             my ($id, $text) = split(/\s*::\s*/, $entry);
1065             print " $id = $text\n";
1066         }
1067         unshift (@entry, $entry);
1068     }
1069     $msg_list{$textName} =
1070         [@entry, [$header, $start, $public, $deprecated]];
1071 }
1073 sub addHeader {
1074     my $header_index = shift;
1076     die "invalid adt_event_N.h index: $header_index\n"
1077         unless ($header_index =~ /^[d+$/);
1079     $headers{$header_index} = $header_index;
1080 }
1082 # $header = 0 is a special case; it is for adt_event.h
1083 # $header > 0 creates adt_event_N.h, where N = $header
1085 sub openHeaderFiles {
1086     my $outfile = shift; # path to an adt_event_N.h file
1088     my $header;
1089     my @Hfile = (); # potentially sparse array of file handles
1090     my @HfileName = (); # parallel array to Hfile, file name (not path)
1091     foreach $header (sort keys %headers) {
1092         my $file = $outfile;

```

```
1093     if ($header > 0) {
1094         $file =~ s/_N/_$header/;
1095     } else {
1096         $file =~ s/_N//;
1097     }
1098     unless (open($Hfile[$header], ">$file")) {
1099         print STDERR "can't open output ($file): $!\n";
1100         $HfileName[$header] = '';
1101         $Hfile[$header] = '';
1102     } else {
1103         my @tmp = split(/\/, $file);
1104         $HfileName[$header] = $tmp[$#tmp];
1105     }
1106 }
1107 return (@Hfile);
1108 }

1110 sub closeHeaderFiles {
1111     my @Hfile = @_;

1113     my $header;
1114     foreach $header (sort keys %headers) {
1115         close $Hfile[$header] if $Hfile[$header];
1116     }
1117 }
```

```

*****
94986 Thu Jun 30 22:02:05 2016
new/usr/src/lib/libbssm/common/adt.xml
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
1 <?xml version="1.0" standalone="yes"?>
2 <!DOCTYPE specification SYSTEM "audit.dtd">
3 <!--
4 CDDL HEADER START

6 The contents of this file are subject to the terms of the
7 Common Development and Distribution License (the "License").
8 You may not use this file except in compliance with the License.

10 You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
11 or http://www.opensolaris.org/os/licensing.
12 See the License for the specific language governing permissions
13 and limitations under the License.

15 When distributing Covered Code, include this CDDL HEADER in each
16 file and include the License file at usr/src/OPENSOLARIS.LICENSE.
17 If applicable, add the following below this CDDL HEADER, with the
18 fields enclosed by brackets "[]" replaced with your own identifying
19 information: Portions Copyright [yyyy] [name of copyright owner]

21 CDDL HEADER END

23 Copyright (c) 2009, 2010, Oracle and/or its affiliates. All rights reserved.

25 -->

27 <specification>

29 <!-- comments are displayed to stderr if debug is on -->
30 <debug set="off"/>

32 <!-- The order of events is arbitrary EXCEPT generic events must
33 precede their instances -->
34 <!-- The order of entries within an event determine the order
35 data is defined in the external API -->
36 <!-- The order of internal / external is arbitrary -->

38 <!--
39 tags:
40 The following top level tags are defined:
41 <event> <token> <msg_list> <debug>

43 event defines an audit record
44 - id is the record id from audit_uevents.h
45 - reorder="yes" or "no". (default is "no").
46 if "yes" then the order of the tokens to be
47 output does not match the order of the data
48 input. (see order attribute of <entry>)
49 - header defines the header file to contain the external
50 definitions for this event type. The header file
51 name is adt_event_N.h, where N is the value supplied
52 header="0" is for "stable" events, > 0 for new ones.
53 with this attribute. (header="1").
54 - idNo is the number associated with the external
55 name of this event. (For AUE_login, ADT_login is
56 the external name and idNo is the value for
57 ADT_login.)
58 - omit is by default 'no' (i.e., don't omit) and can be

```

```

59 'always' or 'JNI'. In the latter case, C interface
60 code is generated but neither Java nor JNI code is.
61 - included text is just a comment
62 Within an event block, the following tags are defined:
63 <entry>, <debug>, <altname>, <title>, <program>, <see>

65 altname defines the internal name of an audit record; if
66 omitted, the internal name is the same as the
67 external name.

69 title, these tags are used by auditrecord(1M) build to create
70 program, audit_record_attr database from adt events.
71 see Following example demonstrates their semantics:

73 auditrecord -p passwd
74 passwd <- <title>
75 program various See passwd(1)
76 ^- <see>
77 event ID 6163 AUE_passwd
78 class lo (0x00001000)
79 header
80 subject
81 [text] username...
82 ^- <comment>
83 return

85 If the length of string in any of the given
86 elements is longer than defined, the string is
87 silently truncated to the defined length in the
88 auditrecord(1M) runtime:
89
90 element <= max (non-truncated) string length
91 title <= 46
92 program <= 20
93 see <= 39
94 comment <= unlimited
95 token <= 28

97 entry defines the correspondence between the data
98 supplied by the caller and the token to be
99 output.
100 - id is the data name that shows up in the structures
101 of adt_event.h If it is a comma separated list,
102 it is the list of names of data to be associated
103 with one output token. (See <external>, below)
104 Within an entry block, the following tags are defined:
105 <internal>, <external>, <debug>, <comment>

107 internal defines the token to be generated.
108 - token is a name that must also be defined with
109 a <token> tag elsewhere in this file. (order is
110 not important).
111 - order="some number" determines the order of the
112 tokens to be output, starting with 1. The subject
113 token is normally order="1". The use is to insure
114 that the order of fields listed in adt_event.h does
115 not change when we arbitrarily change the order of
116 tokens. If the <event reorder="yes"> is not set,
117 order is ignored.
118 - format is a printf-like string that will be used
119 in to format the data supplied by the user.

121 external defines the data to be supplied for creating the
122 token defined via <internal>
123 - opt is one of four values: "required", "optional",
124 "obsolete", or "none". The first two values

```

```

125         indicate that this token's data must or may
126         be supplied by the user; the third value is
127         equivalent to "optional" but shows in the
128         comment that this field is no longer used;
129         the forth value indicates that this token
130         does not require any user-supplied data. If
131         data is required, then a token is always
132         output, while optional data is output only
133         if data is supplied.
134     - type describes the C data type to be associated
135       with the <entry id="dataName">. The following
136       data types are representative:

138         au_asid_t (uint32_t)
139         char
140         char * (blank is optional)
141         char ** (blank is optional)
142         fd_t (int, a file descriptor)
143         uint_t, int, int32_t, uid_t, gid_t
144         uid_t *, gid_t *
145         long, ulong_t
146         m_label_t *
147         pid_t
148         priv_set_t *
149         uint16_t, uint32_t, uint64_t
150         uint32_t *, uint32_t[], uint64_t *
151         msg (not a C type, see below)

153     Below is what Tony said. Above seems to be
154     what is implemented
155         char
156         char * (blank is optional)
157         char ** (blank is optional)
158         int, uid_t, gid_t
159         int *, uid_t *, gid_t *
160         msg (not a C type, see below)
161         time_t
162         uint, uint *

164     The msg type refers to an enumerated type
165     that must be defined via a <msg> description
166     else where in this file. The syntax is
167     special. Example: <external opt="optional"
168     type="msg login_text"/> "login_text" is the
169     id of a <msg_list> descriptor given
170     elsewhere in this file.

172     If the <entry> id is a list, the type must also
173     be a comma-separated list, where the types are
174     in the same order as the id's.
175     If the type is an array, its length must be given
176     explicitly.

178     comment      Used by auditrecord(1M) build to generate
179                  audit_record_attr. Comment is explanation note
180                  printed with token type. Colon (':') may not be
181                  used in a comment. See example above for other
182                  tags related to auditrecord(1M).

184     token        Define allowed token names.
185                  - id is the name of token; this name is used
186                    as an <internal> id.
187                  - a token id name may not end in digits.
188                  - usage is an optional value. At present, only
189                    "TSOL" is defined; it means that this data is
190                    to be used only in Trusted Solaris implementations.

```

```

191         See also example above for "token" tag relation to
192         the auditrecord(1M) output.

194     msg_list     Define a set of text strings.
195                  - id is the name to be used for this group of text
196                    strings in adt_event.h
197                  - header is as defined for <event>
198                  - start is a number where produced enum type begins;
199                    ensure msg lists do not overlap
200                  Within a msg_list block, <msg> and <debug> are defined.
201                  The order of <msg> tags in a msg_list is reflected
202                  directly in adt_event.h. Also add ADT_LIST_<id> to
203                  enum adt_msg_list in adt_xlate.h.

205     msg          Define one string.
206                  - id is the name to be used in the enum describing
207                    this set of strings. Convention: use upper case.
208                  The content (text between <msg> and </msg>) is the
209                    actual string. Extra white space, including line
210                    feeds, is ignored. If empty, no output token
211                    is generated unless the <external> opt attribute is
212                    set to "required", in which case a blank text token
213                    is generated.
214                  Within a msg block, <debug> is defined, but has not been
215                  tested and may have no effect.

217     debug        This turns on/off debug messages during the processing
218                  of the xml data. It affects the block within which it
219                  is defined.
220                  - set may have one of two values: "on" or "off". If
221                    set is omitted, the debug state for the current block
222                    is toggled.
223                  The use of the <debug> tag does not affect the output
224                  of data to the various files created, but does generate
225                  potentially large amounts of output to stderr.

227     -->
228     <!--        template for an event record definition

230     <event id="" header="0" idNo="">
231         <entry id="subject">
232             <internal token="subject"/>
233             <external opt="none"/>
234         </entry>
235         <entry id="">
236             <internal token=""/>
237             <external opt="" type="" />
238         </entry>
239         <entry id="return">
240             <internal token="return"/>
241             <external opt="none"/>
242         </entry>
243     </event>

245     Generic events must precede Instance events; within each
246     group, please group the AUE_* by area and event idNo-s in order,
247     gaps in idNo-s are OK.
248     N.B. Renumbering idNo-s requires recompilation of consumers. See
249     the contracts for whom to notify if/when this happens.
250     -->

252     <!-- generic events -->

254     <!--
255         'omit="always"' means that this record type is not reflected
256         in the generated header and table files.

```

```

257     -->
259     <event id="AUE_generic_basic" type="generic" omit="always">
260         <!--
262             This is a template for the event types that have no tokens
263             other than the header and return. There is no allowed_type
264             list because the template is not externally visible due to the
265             omit="always".
267         -->
268         <entry id="subject">
269             <internal token="subject"/>
270             <external opt="none"/>
271         </entry>
272         <entry id="return">
273             <internal token="return"/>
274             <external opt="none"/>
275         </entry>
276     </event>
278     <event id="AUE_generic_login" type="generic" omit="always">
279         <!--
281             This is a template for the various login event types
282             AUE_login, AUE_ftp, etc which match this template. There is
283             no allowed_type list because the template is not externally
284             visible due to the omit="always".
286         -->
287         <entry id="subject">
288             <internal token="subject"/>
289             <external opt="none"/>
290         </entry>
292         <!-- This field is still in use for SMC until it is cleaned up,
293             it must remain, see login_text msg list at the end of the
294             file.
295         -->
296         <entry id="message">
297             <internal token="text"/>
298             <external opt="optional" type="msg login_text"/>
299             <comment>error message</comment>
300         </entry>
301         <entry id="return">
302             <internal token="return"/>
303             <external opt="none"/>
304         </entry>
305     </event>
307 <!-- generic SMC events -->
309     <event id="AUE_generic_SMC_add" type="generic" omit="always">
310         <entry id="subject">
311             <internal token="subject"/>
312             <external opt="none"/>
313         </entry>
314         <entry id="object_name">
315             <internal token="text"/>
316             <external opt="required" type="char **"/>
317             <comment>object name</comment>
318         </entry>
319         <entry id="domain">
320             <internal token="text"/>
321             <external opt="optional" type="char **"/>
322             <comment>domain</comment>

```

```

323     </entry>
324     <entry id="name_service">
325         <internal token="text"/>
326         <external opt="required" type="char **"/>
327         <comment>name_service</comment>
328     </entry>
329     <entry id="auth_used">
330         <internal token="uauth"/>
331         <external opt="optional" type="char **"/>
332         <comment>authorization used</comment>
333     </entry>
334     <!--
335         This should really be its own token type, not "text"
336     -->
337     <entry id="initial_values">
338         <internal token="text"/>
339         <external opt="required" type="char **"/>
340         <comment>initial values</comment>
341     </entry>
342     <entry id="return">
343         <internal token="return"/>
344         <external opt="none"/>
345     </entry>
346 </event>
348 <event id="AUE_generic_SMC_delete" type="generic" omit="always">
349     <entry id="subject">
350         <internal token="subject"/>
351         <external opt="none"/>
352     </entry>
353     <entry id="object_name">
354         <internal token="text"/>
355         <external opt="required" type="char **"/>
356         <comment>object name</comment>
357     </entry>
358     <entry id="domain">
359         <internal token="text"/>
360         <external opt="optional" type="char **"/>
361         <comment>domain</comment>
362     </entry>
363     <entry id="name_service">
364         <internal token="text"/>
365         <external opt="required" type="char **"/>
366         <comment>name_service</comment>
367     </entry>
368     <entry id="auth_used">
369         <internal token="uauth"/>
370         <external opt="optional" type="char **"/>
371         <comment>authorization used</comment>
372     </entry>
373     <entry id="delete_values">
374         <internal token="text"/>
375         <external opt="required" type="char **"/>
376         <comment>deleted values</comment>
377     </entry>
378     <entry id="return">
379         <internal token="return"/>
380         <external opt="none"/>
381     </entry>
382 </event>
384 <event id="AUE_generic_SMC_modify" type="generic" omit="always">
385     <entry id="subject">
386         <internal token="subject"/>
387         <external opt="none"/>
388     </entry>

```

```

389     <entry id="object_name">
390         <internal token="text"/>
391         <external opt="required" type="char *"/>
392         <comment>object name</comment>
393     </entry>
394     <entry id="domain">
395         <internal token="text"/>
396         <external opt="optional" type="char *"/>
397         <comment>domain</comment>
398     </entry>
399     <entry id="name_service">
400         <internal token="text"/>
401         <external opt="required" type="char *"/>
402         <comment>name_service</comment>
403     </entry>
404     <entry id="auth_used">
405         <internal token="uauth"/>
406         <external opt="optional" type="char *"/>
407         <comment>authorization used</comment>
408     </entry>
409     <entry id="changed_values">
410         <internal token="text"/>
411         <external opt="required" type="char *"/>
412         <comment>changed values</comment>
413     </entry>
414     <entry id="return">
415         <internal token="return"/>
416         <external opt="none"/>
417     </entry>
418 </event>

420 <!-- instances -->

422 <!--
423     Java needed for SMC events.  Since the SMC events grow less
424     often than the C related events.  They come first.  It
425     would be nice to reorder the idNo-s, but that's an ABI
426     change and should rev libbasm version no.  If reordered
427     start with 1 and eliminate the comment at the end about
428     the highest idNo.
429 -->
430 <event id="AUE_admin_authenticate" instance_of="AUE_generic_login"
431     header="0" idNo="3">
432     <title>Admin Server Authentication</title>
433     <program>admin (various)</program>
434     <see>SMC, WBEM, or AdminSuite</see>
435 </event>

437 <event id="AUE_filesystem_add" instance_of="AUE_generic_SMC_add"
438     header="0" idNo="4">
439     <title>SMC: filesystem add</title>
440     <program>SMC server</program>
441 </event>
442 <event id="AUE_filesystem_delete" instance_of="AUE_generic_SMC_delete"
443     header="0" idNo="5">
444     <title>SMC: filesystem delete</title>
445     <program>SMC server</program>
446 </event>
447 <event id="AUE_filesystem_modify" instance_of="AUE_generic_SMC_modify"
448     header="0" idNo="6">
449     <title>SMC: filesystem modify</title>
450     <program>SMC server</program>
451 </event>

453 <event id="AUE_network_add" instance_of="AUE_generic_SMC_add"
454     header="0" idNo="7">

```

```

455     <title>SMC: network add</title>
456     <program>SMC server</program>
457 </event>
458 <event id="AUE_network_delete" instance_of="AUE_generic_SMC_delete"
459     header="0" idNo="8">
460     <title>SMC: network delete</title>
461     <program>SMC server</program>
462 </event>
463 <event id="AUE_network_modify" instance_of="AUE_generic_SMC_modify"
464     header="0" idNo="9">
465     <title>SMC: network modify</title>
466     <program>SMC server</program>
467 </event>

469 <event id="AUE_printer_add" instance_of="AUE_generic_SMC_add"
470     header="0" idNo="10">
471     <title>SMC: printer add</title>
472     <program>SMC server</program>
473 </event>
474 <event id="AUE_printer_delete" instance_of="AUE_generic_SMC_delete"
475     header="0" idNo="11">
476     <title>SMC: printer delete</title>
477     <program>SMC server</program>
478 </event>
479 <event id="AUE_printer_modify" instance_of="AUE_generic_SMC_modify"
480     header="0" idNo="12">
481     <title>SMC: printer modify</title>
482     <program>SMC server</program>
483 </event>

485 <!--
486     This is SMC; it's also used in su and should probably be used in
487     desktop role login.  If we fix the SMC to not record NO_MSG here,
488     we can fix to record failed user.  See su.c and AUE_su.
489 -->
490 <event id="AUE_role_login" instance_of="AUE_generic_login"
491     header="0" idNo="13">
492     <title>RBAC: role login</title>
493     <program>SMC server</program>
494     <program>/usr/bin/su</program>
495 </event>

497 <event id="AUE_scheduledjob_add" instance_of="AUE_generic_SMC_add"
498     header="0" idNo="14">
499     <title>SMC: scheduled job add</title>
500     <program>SMC server</program>
501 </event>
502 <event id="AUE_scheduledjob_delete" instance_of="AUE_generic_SMC_delete"
503     header="0" idNo="15">
504     <title>SMC: scheduled job delete</title>
505     <program>SMC server</program>
506 </event>
507 <event id="AUE_scheduledjob_modify" instance_of="AUE_generic_SMC_modify"
508     header="0" idNo="16">
509     <title>SMC: scheduled job modify</title>
510     <program>SMC server</program>
511 </event>

513 <event id="AUE_serialport_add" instance_of="AUE_generic_SMC_add"
514     header="0" idNo="17">
515     <title>SMC: serial port add</title>
516     <program>SMC server</program>
517 </event>
518 <event id="AUE_serialport_delete" instance_of="AUE_generic_SMC_delete"
519     header="0" idNo="18">
520     <title>SMC: serial port delete</title>

```



```

521     <program>SMC server</program>
522 </event>
523 <event id="AUE_serialport_modify" instance_of="AUE_generic_SMC_modify"
524     header="0" idNo="19">
525     <title>SMC: serial port modify</title>
526     <program>SMC server</program>
527 </event>

529 <!-- This is SMC; should this also be used elsewhere? -->
530 <event id="AUE_uauth" header="0" idNo="20">
531     <title>SMC: Use of Authorization</title>
532     <program>SMC server</program>
533     <entry id="subject">
534         <internal token="subject"/>
535         <external opt="none"/>
536     </entry>
537     <entry id="auth_used">
538         <internal token="uauth"/>
539         <external opt="required" type="char **"/>
540         <comment>authorization used</comment>
541     </entry>
542     <entry id="objectname">
543         <internal token="text"/>
544         <external opt="required" type="char **"/>
545         <comment>object name</comment>
546     </entry>
547     <entry id="return">
548         <internal token="return"/>
549         <external opt="none"/>
550     </entry>
551 </event>

553 <event id="AUE_usermgr_add" instance_of="AUE_generic_SMC_add"
554     header="0" idNo="21">
555     <title>SMC: User Manager add</title>
556     <program>SMC server</program>
557 </event>
558 <event id="AUE_usermgr_delete" instance_of="AUE_generic_SMC_delete"
559     header="0" idNo="22">
560     <title>SMC: User Manager delete</title>
561     <program>SMC server</program>
562 </event>
563 <event id="AUE_usermgr_modify" instance_of="AUE_generic_SMC_modify"
564     header="0" idNo="23">
565     <title>SMC: User Manager modify</title>
566     <program>SMC server</program>
567 </event>
568 <!-- end of Java needed for SMC events -->
569 <!--
570     while not used by SMC logout is used by Lockhart
571 -->
572 <event id="AUE_logout" header="0" idNo="1">
573     <title>login: logout</title>
574     <program>various</program>
575     <see>login(1)</see>
576     <entry id="subject">
577         <internal token="subject"/>
578         <external opt="none"/>
579     </entry>
580 <!--
581     not used by C code, used by Lockhart,
582     get them to change and remove
583     event.user_name("logout " + session.getUserName());
584     from /ws/lockhart-nv-gate/src/bundled/app/webmgt/lib/services/
585     com/sun/management/services/audit/SolarisAuditEvent_Logout.java
586 -->

```

```

587     <entry id="user_name">
588         <internal token="text" format="logout %s"/>
589         <external opt="optional" type="char **"/>
590         <comment>"logout" username</comment>
591     </entry>
592     <entry id="return">
593         <internal token="return"/>
594         <external opt="none"/>
595     </entry>
596 </event>

599 <!-- C Only events -->
600 <event id="AUE_init_solaris" header="0" idNo="32" omit="JNI">
601     <title>init</title>
602     <program>/sbin/init</program>
603     <program>/usr/sbin/init</program>
604     <program>/usr/sbin/shutdown</program>
605     <entry id="subject">
606         <internal token="subject"/>
607         <external opt="none"/>
608     </entry>
609     <entry id="info">
610         <internal token="text"/>
611         <external opt="optional" type="char **"/>
612         <comment>init level or zone name</comment>
613     </entry>
614     <entry id="return">
615         <internal token="return"/>
616         <external opt="none"/>
617     </entry>
618 </event>

620 <event id="AUE_login" instance_of="AUE_generic_login" header="0"
621     idNo="25" omit="JNI">
622     <title>terminal login</title>
623     <program>/usr/sbin/login</program>
624     <program>/usr/dt/bin/dtlogin</program>
625     <see>login(1)</see>
626     <see>dtlogin</see>
627 </event>
628 <event id="AUE_rlogin" instance_of="AUE_generic_login" header="0"
629     idNo="28" omit="JNI">
630     <title>rlogin</title>
631     <program>/usr/sbin/login</program>
632     <see>login(1) - rlogin</see>
633 </event>
634 <event id="AUE_telnet" instance_of="AUE_generic_login" header="0"
635     idNo="29" omit="JNI">
636     <title>telnet login</title>
637     <program>/usr/sbin/login</program>
638     <see>login(1) - telnet</see>
639 </event>
640 <event id="AUE_ssh" instance_of="AUE_generic_login" header="0"
641     idNo="2" omit="JNI">
642     <program>/usr/lib/ssh/sshd</program>
643 </event>

645 <event id="AUE_zlogin" header="0" idNo="38" omit="JNI">
646     <title>zone login</title>
647     <program>/usr/sbin/login</program>
648     <see>zlogin(1)</see>
649     <entry id="subject">
650         <internal token="subject"/>
651         <external opt="none"/>
652     </entry>

```

```

653     <entry id="message">
654         <internal token="text"/>
655         <external opt="optional" type="char *"/>
656         <comment>error message</comment>
657     </entry>
658     <entry id="return">
659         <internal token="return"/>
660         <external opt="none"/>
661     </entry>
662 </event>

664 <event id="AUE_su" header="0" idNo="30" omit="JNI">
665     <title>su</title>
666     <program>/usr/bin/su</program>
667     <see>su(1M)</see>
668     <entry id="subject">
669         <internal token="subject"/>
670         <external opt="none"/>
671     </entry>
672 <!--
673     should be changed to "fail_user" and su.c updated
674     However, the jni stuff is broken, so for now it's "message"
675 -->
676     <entry id="message">
677         <internal token="text"/>
678         <external opt="optional" type="char *"/>
679         <comment>"user name" of failed new user/role</comment>
680     </entry>
681     <entry id="return">
682         <internal token="return"/>
683         <external opt="none"/>
684     </entry>
685 </event>

687 <event id="AUE_passwd" header="0" idNo="27" omit="JNI">
688     <title>passwd</title>
689     <program>various</program>
690     <see>passwd(1)</see>
691     <entry id="subject">
692         <internal token="subject"/>
693         <external opt="none"/>
694     </entry>
695     <entry id="uid,username">
696         <internal token="user"/>
697         <external opt="optional" type="uid_t,char *"/>
698         <comment>user if different than caller</comment>
699     </entry>
700     <entry id="return">
701         <internal token="return"/>
702         <external opt="none"/>
703     </entry>
704 </event>

706 <event id="AUE_screenlock" instance_of="AUE_generic_basic" header="0"
707     idNo="26" omit="JNI">
708     <program>desktop screen lock</program>
709 </event>
710 <event id="AUE_screenunlock" instance_of="AUE_generic_basic" header="0"
711     idNo="31" omit="JNI">
712     <program>desktop screen unlock</program>
713 </event>

715 <!--
716     AUE_prof_cmd is not supportable for Java due to the structure of
717     the priv token.  When and if a Java program needs to generate
718     a priv token, we'll need to look at the data format in the

```

```

719     Java code and provide an appropriate java and jni implementation.
720 -->

722 <event id="AUE_prof_cmd" header="0" idNo="24" omit="JNI">
723     <title>pfexec</title>
724     <program>/usr/bin/pfexec</program>
725     <see>pfexec(1)</see>
726     <entry id="subject">
727         <internal token="subject"/>
728         <external opt="none"/>
729     </entry>
730     <entry id="cwdpath">
731         <internal token="path"/>
732         <external opt="required" type="char *"/>
733         <comment>working directory</comment>
734     </entry>
735     <entry id="cmdpath">
736         <internal token="path"/>
737         <external opt="required" type="char *"/>
738         <comment>command pathname</comment>
739     </entry>
740     <entry id="argc,argv,envp">
741         <internal token="command"/>
742         <external opt="required" type="int,char**,char**"/>
743     </entry>
744     <entry id="proc_auid,proc_euid,proc_egid,proc_ruid,proc_rgid,proc_pid,pr
745         <internal token="process"/>
746         <external opt="required"
747             type="uid_t,uid_t,gid_t,uid_t,gid_t,pid_t,au_asid_t,termid*"/>
748     </entry>
749     <entry id="limit_set">
750         <internal token="priv_limit"/>
751         <external opt="optional" type="priv_set_t*"/>
752     </entry>
753     <entry id="inherit_set">
754         <internal token="priv_inherit"/>
755         <external opt="optional" type="priv_set_t*"/>
756     </entry>
757     <entry id="return">
758         <internal token="return"/>
759         <external opt="none"/>
760     </entry>
761 </event>

763 <event id="AUE_inetd_connect" header="0" idNo="34" omit="JNI">
764     <title>inetd</title>
765     <program>/usr/sbin/inetd</program>
766     <entry id="subject">
767         <internal token="subject"/>
768         <external opt="none"/>
769     </entry>
770     <entry id="service_name">
771         <internal token="text"/>
772         <external opt="optional" type="char *"/>
773         <comment>service name</comment>
774     </entry>
775     <entry id="ip_type,ip_remote_port,ip_local_port,ip_adr">
776         <internal token="tid"/>
777         <external opt="required"
778             type="uint32_t,uint16_t,uint16_t,uint32_t[4]"/>
779         <comment>client address</comment>
780     </entry>
781     <entry id="cmd">
782         <internal token="command_alt"/>
783         <external opt="required" type="char *"/>
784         <comment>inetd command</comment>

```

```

785     </entry>
786     <entry id="privileges">
787       <internal token="priv_effective"/>
788       <external opt="required" type="priv_set_t **"/>
789     </entry>
790     <entry id="return">
791       <internal token="return"/>
792       <external opt="none"/>
793     </entry>
794 </event>

796 <event id="AUE_inetd_ratelimit" header="0" idNo="35" omit="JNI">
797   <title>inetd</title>
798   <program>/usr/sbin/inetd</program>
799   <entry id="subject">
800     <internal token="subject"/>
801     <external opt="none"/>
802   </entry>
803   <entry id="service_name">
804     <internal token="text"/>
805     <external opt="optional" type="char **"/>
806     <comment>service name</comment>
807   </entry>
808   <entry id="limit">
809     <internal token="text"/>
810     <external opt="required" type="char **"/>
811     <comment>limit value</comment>
812   </entry>
813   <entry id="return">
814     <internal token="return"/>
815     <external opt="none"/>
816   </entry>
817 </event>

819 <event id="AUE_inetd_copylimit" header="0" idNo="36" omit="JNI">
820   <title>inetd</title>
821   <program>/usr/sbin/inetd</program>
822   <entry id="subject">
823     <internal token="subject"/>
824     <external opt="none"/>
825   </entry>
826   <entry id="service_name">
827     <internal token="text"/>
828     <external opt="optional" type="char **"/>
829     <comment>service name</comment>
830   </entry>
831   <entry id="limit">
832     <internal token="text"/>
833     <external opt="required" type="char **"/>
834     <comment>limit value</comment>
835   </entry>
836   <entry id="return">
837     <internal token="return"/>
838     <external opt="none"/>
839   </entry>
840 </event>

842 <event id="AUE_inetd_failrate" header="0" idNo="37" omit="JNI">
843   <title>inetd</title>
844   <program>/usr/sbin/inetd</program>
845   <entry id="subject">
846     <internal token="subject"/>
847     <external opt="none"/>
848   </entry>
849   <entry id="service_name">
850     <internal token="text"/>

```

```

851     <external opt="optional" type="char **"/>
852     <comment>service name</comment>
853   </entry>
854   <entry id="values">
855     <internal token="text"/>
856     <external opt="required" type="char **"/>
857     <comment>limit value, interval</comment>
858   </entry>
859   <entry id="return">
860     <internal token="return"/>
861     <external opt="none"/>
862   </entry>
863 </event>

865 <event id="AUE_zone_state" header="0" idNo="33" omit="JNI">
866   <entry id="subject">
867     <internal token="subject"/>
868     <external opt="none"/>
869   </entry>
870   <entry id="new_state">
871     <internal token="text"/>
872     <external opt="required" type="char **"/>
873     <comment>New zone state</comment>
874   </entry>
875   <entry id="zonename">
876     <internal token="zonename"/>
877     <external opt="required" type="char **"/>
878     <comment>zone name</comment>
879   </entry>
880   <entry id="return">
881     <internal token="return"/>
882     <external opt="none"/>
883   </entry>
884 </event>

886 <event id="AUE_su_logout" instance_of="AUE_generic_basic"
887   header="0" idNo="39" omit="JNI">
888   <title>su</title>
889   <program>/usr/bin/su</program>
890   <see>su(1M)</see>
891 </event>

893 <event id="AUE_role_logout" instance_of="AUE_generic_basic"
894   header="0" idNo="40" omit="JNI">
895   <title>su</title>
896   <program>/usr/bin/su</program>
897   <see>su(1M)</see>
898 </event>

900 <event id="AUE_newgrp_login" header="0" idNo="41" omit="JNI">
901   <program>newgrp</program>
902   <entry id="subject">
903     <internal token="subject"/>
904     <external opt="none"/>
905   </entry>
906   <entry id="groupname">
907     <internal token="text"/>
908     <external opt="required" type="char **"/>
909     <comment>group name</comment>
910   </entry>
911   <entry id="return">
912     <internal token="return"/>
913     <external opt="none"/>
914   </entry>
915 </event>

```

```

917 <event id="AUE_generic_mountable" type="generic" omit="always">
918 <!--
920     User device mounting related functions
922     -->
923     <entry id="subject">
924         <internal token="subject"/>
925         <external opt="none"/>
926     </entry>
927     <entry id="auth_used">
928         <internal token="uauth"/>
929         <external opt="required" type="char **"/>
930         <comment>authorization used</comment>
931     </entry>
932     <entry id="mount_point">
933         <internal token="path"/>
934         <external opt="required" type="char **"/>
935         <comment>mount point</comment>
936     </entry>
937     <entry id="device">
938         <internal token="path"/>
939         <external opt="required" type="char **"/>
940         <comment>device</comment>
941     </entry>
942     <entry id="options">
943         <internal token="text"/>
944         <external opt="optional" type="char **"/>
945         <comment>options</comment>
946     </entry>
947     <entry id="return">
948         <internal token="return"/>
949         <external opt="none"/>
950     </entry>
951 </event>
953 <event id="AUE_attach" instance_of="AUE_generic_mountable"
954     header="0" idNo="42" omit="JNI">
955     <program>hald</program>
956 </event>
957 <event id="AUE_detach" instance_of="AUE_generic_mountable"
958     header="0" idNo="43" omit="JNI">
959     <program>hald</program>
960 </event>
961 <event id="AUE_remove" header="0" idNo="44" omit="JNI">
962     <program>hald</program>
963     <entry id="subject">
964         <internal token="subject"/>
965         <external opt="none"/>
966     </entry>
967     <entry id="auth_used">
968         <internal token="uauth"/>
969         <external opt="required" type="char **"/>
970         <comment>authorization used</comment>
971     </entry>
972     <entry id="mount_point">
973         <internal token="path"/>
974         <external opt="optional" type="char **"/>
975         <comment>mount point</comment>
976     </entry>
977     <entry id="device">
978         <internal token="path"/>
979         <external opt="required" type="char **"/>
980         <comment>device</comment>
981     </entry>
982     <entry id="return">

```

```

983         <internal token="return"/>
984         <external opt="none"/>
985     </entry>
986 </event>
987
988 <event id="AUE_pool_import" header="0" idNo="45" omit="JNI">
989     <program>hald</program>
990     <entry id="subject">
991         <internal token="subject"/>
992         <external opt="none"/>
993     </entry>
994     <entry id="auth_used">
995         <internal token="uauth"/>
996         <external opt="required" type="char **"/>
997         <comment>authorization used</comment>
998     </entry>
999     <entry id="pool">
1000         <internal token="text"/>
1001         <external opt="required" type="char **"/>
1002         <comment>pool</comment>
1003     </entry>
1004     <entry id="device">
1005         <internal token="path"/>
1006         <external opt="required" type="char **"/>
1007         <comment>device</comment>
1008     </entry>
1009     <entry id="return">
1010         <internal token="return"/>
1011         <external opt="none"/>
1012     </entry>
1013 </event>
1014 <event id="AUE_pool_export" header="0" idNo="46" omit="JNI">
1015     <program>hald</program>
1016     <entry id="subject">
1017         <internal token="subject"/>
1018         <external opt="none"/>
1019     </entry>
1020     <entry id="auth_used">
1021         <internal token="uauth"/>
1022         <external opt="required" type="char **"/>
1023         <comment>authorization used</comment>
1024     </entry>
1025     <entry id="pool">
1026         <internal token="text"/>
1027         <external opt="required" type="char **"/>
1028         <comment>pool</comment>
1029     </entry>
1030     <entry id="device">
1031         <internal token="path"/>
1032         <external opt="required" type="char **"/>
1033         <comment>device</comment>
1034     </entry>
1035     <entry id="return">
1036         <internal token="return"/>
1037         <external opt="none"/>
1038     </entry>
1039 </event>
1041 <!-- dladm security objected events -->
1042 <event id="AUE_dladm_generic" type="generic" omit="always">
1043     <entry id="subject">
1044         <internal token="subject"/>
1045         <external opt="none"/>
1046     </entry>
1047     <entry id="auth_used">
1048         <internal token="uauth"/>

```

```

1049     <external opt="required" type="char **"/>
1050     <comment>authorization used</comment>
1051 </entry>
1052 <entry id="obj_class">
1053     <internal token="text"/>
1054     <external opt="required" type="char **"/>
1055     <comment>object class name</comment>
1056 </entry>
1057 <entry id="obj_name">
1058     <internal token="text"/>
1059     <external opt="required" type="char **"/>
1060     <comment>object name</comment>
1061 </entry>
1062 <entry id="return">
1063     <internal token="return"/>
1064     <external opt="none"/>
1065 </entry>
1066 </event>

1068 <event id="AUE_dladm_create_secobj" instance_of="AUE_dladm_generic"
1069     header="0" idNo="47" omit="JNI">
1070     <title>create wifi security object</title>
1071     <program>/usr/sbin/dladm</program>
1072     <see>dladm(1M)</see>
1073 </event>
1074 <event id="AUE_dladm_delete_secobj" instance_of="AUE_dladm_generic"
1075     header="0" idNo="48" omit="JNI">
1076     <title>delete wifi security object</title>
1077     <program>/usr/sbin/dladm</program>
1078     <see>dladm(1M)</see>
1079 </event>

1081 <!-- Trusted eXtensions (TX) events -->

1083 <!-- labeld events -->
1084 <event id="AUE_file_relabel" header="0" idNo="49" omit="JNI">
1085     <title>relabel file from one zone to another</title>
1086     <program>setlabel(1)</program>
1087     <see>setlabel(3TOS)</see>
1088     <entry id="subject">
1089         <internal token="subject"/>
1090         <external opt="none"/>
1091     </entry>
1092     <entry id="auth_used">
1093         <internal token="uauth"/>
1094         <external opt="required" type="char **"/>
1095         <comment>authorization used</comment>
1096     </entry>
1097     <entry id="file">
1098         <internal token="path"/>
1099         <external opt="required" type="char **"/>
1100         <comment>file relabeled</comment>
1101     </entry>
1102     <entry id="src_label">
1103         <internal token="label"/>
1104         <external opt="required" type="m_label_t **"/>
1105         <comment>original label</comment>
1106     </entry>
1107     <entry id="dst_label">
1108         <internal token="label"/>
1109         <external opt="required" type="m_label_t **"/>
1110         <comment>new label</comment>
1111     </entry>
1112     <entry id="return">
1113         <internal token="return"/>
1114         <external opt="none"/>

```

```

1115     </entry>
1116 </event>

1118 <event id="AUE_file_copy" header="0" idNo="50" omit="JNI">
1119     <title>copy file to another zone</title>
1120     <program>dtfile(1X)</program>
1121     <entry id="subject">
1122         <internal token="subject"/>
1123         <external opt="none"/>
1124     </entry>
1125     <entry id="auth_used">
1126         <internal token="uauth"/>
1127         <external opt="required" type="char **"/>
1128         <comment>authorization used</comment>
1129     </entry>
1130     <entry id="src_file">
1131         <internal token="path"/>
1132         <external opt="required" type="char **"/>
1133         <comment>source file</comment>
1134     </entry>
1135     <entry id="src_label">
1136         <internal token="label"/>
1137         <external opt="required" type="m_label_t **"/>
1138         <comment>source label</comment>
1139     </entry>
1140     <entry id="dst_file">
1141         <internal token="path"/>
1142         <external opt="required" type="char **"/>
1143         <comment>destination directory</comment>
1144     </entry>
1145     <entry id="dst_label">
1146         <internal token="label"/>
1147         <external opt="required" type="m_label_t **"/>
1148         <comment>destination label</comment>
1149     </entry>
1150     <entry id="return">
1151         <internal token="return"/>
1152         <external opt="none"/>
1153     </entry>
1154 </event>

1156 <!-- uadmin(1M) events -->
1157 <event id="AUE_uadmin_generic" type="generic" omit="always">
1158     <entry id="subject">
1159         <internal token="subject"/>
1160         <external opt="none"/>
1161     </entry>
1162     <entry id="fcn">
1163         <internal token="text"/>
1164         <external opt="required" type="msg uadmin_fcn"/>
1165         <comment>next action</comment>
1166     </entry>
1167     <entry id="mdep">
1168         <internal token="text"/>
1169         <external opt="optional" type="char **"/>
1170         <comment>machine dependent argument</comment>
1171     </entry>
1172     <entry id="return">
1173         <internal token="return"/>
1174         <external opt="none"/>
1175     </entry>
1176 </event>
1177 <event id="AUE_uadmin_generic_fcn" type="generic" omit="always">
1178     <entry id="subject">
1179         <internal token="subject"/>
1180         <external opt="none"/>

```

```

1181     </entry>
1182     <entry id="fcn">
1183         <internal token="text"/>
1184         <external opt="required" type="msg uadmin_fcn"/>
1185         <comment>next action</comment>
1186     </entry>
1187     <entry id="return">
1188         <internal token="return"/>
1189         <external opt="none"/>
1190     </entry>
1191 </event>
1192 <event id="AUE_uadmin_shutdown" instance_of="AUE_uadmin_generic"
1193     header="0" idNo="51" omit="JNI">
1194     <title>uadmin shutdown</title>
1195     <program>/sbin/uadmin</program>
1196     <program>/usr/sbin/uadmin</program>
1197     <see>uadmin(1M)</see>
1198 </event>
1199 <event id="AUE_uadmin_reboot" instance_of="AUE_uadmin_generic"
1200     header="0" idNo="52" omit="JNI">
1201     <title>uadmin reboot</title>
1202     <program>/sbin/uadmin</program>
1203     <program>/usr/sbin/uadmin</program>
1204     <see>uadmin(1M)</see>
1205 </event>
1206 <event id="AUE_uadmin_dump" instance_of="AUE_uadmin_generic"
1207     header="0" idNo="53" omit="JNI">
1208     <title>uadmin dump</title>
1209     <program>/sbin/uadmin</program>
1210     <program>/usr/sbin/uadmin</program>
1211     <see>uadmin(1M)</see>
1212 </event>
1213 <event id="AUE_uadmin_freeze" instance_of="AUE_uadmin_generic"
1214     header="0" idNo="54" omit="JNI">
1215     <title>uadmin freeze</title>
1216     <program>/sbin/uadmin</program>
1217     <program>/usr/sbin/uadmin</program>
1218     <see>uadmin(1M)</see>
1219 </event>
1220 <event id="AUE_uadmin_remount" header="0" idNo="55" omit="JNI">
1221     <title>uadmin remount</title>
1222     <program>/sbin/uadmin</program>
1223     <program>/usr/sbin/uadmin</program>
1224     <see>uadmin(1M)</see>
1225     <entry id="subject">
1226         <internal token="subject"/>
1227         <external opt="none"/>
1228     </entry>
1229     <entry id="return">
1230         <internal token="return"/>
1231         <external opt="none"/>
1232     </entry>
1233 </event>
1234 <!-- uadmin ftrace and swapctl are not documented in uadmin(2) -->
1235 <event id="AUE_uadmin_ftrace" instance_of="AUE_uadmin_generic"
1236     header="0" idNo="56" omit="JNI">
1237     <title>uadmin ftrace</title>
1238     <program>/sbin/uadmin</program>
1239     <program>/usr/sbin/uadmin</program>
1240     <see>uadmin(1M)</see>
1241 </event>
1242 <event id="AUE_uadmin_swapctl" instance_of="AUE_uadmin_generic_fcn"
1243     header="0" idNo="57" omit="JNI">
1244     <title>uadmin swapctl</title>
1245     <program>/sbin/uadmin</program>
1246     <program>/usr/sbin/uadmin</program>

```

```

1247     <see>uadmin(1M)</see>
1248 </event>
1249 <event id="AUE_uadmin_thaw" header="0" idNo="96" omit="JNI">
1250     <title>thaw after freeze</title>
1251     <program>/sbin/uadmin</program>
1252     <program>/usr/sbin/uadmin</program>
1253     <see>uadmin(1M)</see>
1254     <entry id="subject">
1255         <internal token="subject"/>
1256         <external opt="none"/>
1257     </entry>
1258     <entry id="fcn">
1259         <internal token="text"/>
1260         <external opt="required" type="msg uadmin_fcn"/>
1261         <comment>freeze action type</comment>
1262     </entry>
1263     <entry id="return">
1264         <internal token="return"/>
1265         <external opt="none"/>
1266     </entry>
1267 </event>
1268 <!-- uadmin config is not documented in uadmin(2) -->
1269 <event id="AUE_uadmin_config" instance_of="AUE_uadmin_generic"
1270     header="0" idNo="119" omit="JNI">
1271     <title>uadmin config</title>
1272     <program>/sbin/uadmin</program>
1273     <program>/usr/sbin/uadmin</program>
1274     <see>uadmin(1M)</see>
1275 </event>
1276
1277 <!-- smbd service event; smbd session setup -->
1278 <event id="AUE_smbd_session" header="0" idNo="58" omit="JNI">
1279     <title>smbd</title>
1280     <program>/usr/lib/smbd/smbd</program>
1281     <entry id="subject">
1282         <internal token="subject"/>
1283         <external opt="none"/>
1284     </entry>
1285     <entry id="domain">
1286         <internal token="text"/>
1287         <external opt="required" type="char*"/>
1288     <comment>domain</comment>
1289     </entry>
1290     <entry id="username">
1291         <internal token="text"/>
1292         <external opt="required" type="char*"/>
1293     <comment>username</comment>
1294     </entry>
1295     <entry id="sid">
1296         <internal token="text"/>
1297         <external opt="optional" type="char*"/>
1298     <comment>sid</comment>
1299     </entry>
1300     <entry id="return">
1301         <internal token="return"/>
1302         <external opt="none"/>
1303     </entry>
1304 </event>
1305
1306 <!-- smbd service event; smbd session logoff -->
1307 <event id="AUE_smbd_logoff" header="0" idNo="59" omit="JNI">
1308     <title>smbd</title>
1309     <program>/usr/lib/smbd/smbd</program>
1310     <entry id="subject">
1311         <internal token="subject"/>
1312         <external opt="none"/>

```

```

1313     </entry>
1314     <entry id="domain">
1315         <internal token="text"/>
1316         <external opt="required" type="char*" />
1317         <comment>domain</comment>
1318     </entry>
1319     <entry id="username">
1320         <internal token="text"/>
1321         <external opt="required" type="char*" />
1322         <comment>username</comment>
1323     </entry>
1324     <entry id="return">
1325         <internal token="return"/>
1326         <external opt="none"/>
1327     </entry>
1328 </event>

1330 <!-- vscan service event; infected file detected -->
1331 <event id="AUE_vscan_quarantine" header="0" idNo="60" omit="JNI">
1332     <title>VSCAN: quarantine infected file</title>
1333     <program>/usr/lib/vscan/vscand</program>
1334     <see>vscand(1M), ICAP RFC 3507 (Extensions)</see>
1335     <entry id="subject">
1336         <internal token="subject"/>
1337         <external opt="none"/>
1338     </entry>
1339     <entry id="file">
1340         <internal token="path"/>
1341         <external opt="required" type="char*" />
1342         <comment>infected file</comment>
1343     </entry>
1344     <entry id="violations,nviolations">
1345         <internal token="text"/>
1346         <external opt="optional" type="char**,int"/>
1347         <comment>ID - threat description</comment>
1348     </entry>
1349     <entry id="return">
1350         <internal token="return"/>
1351         <external opt="none"/>
1352     </entry>
1353 </event>

1355 <!-- ndmp service event; ndmp client connect -->
1356 <event id="AUE_ndmp_connect" instance_of="AUE_generic_basic" header="0"
1357     idNo="61" omit="JNI">
1358     <title>NDMP Connect</title>
1359     <program>/usr/lib/ndmp/ndmpd</program>
1360     <see>ndmpd(1M)</see>
1361 </event>

1363 <!-- ndmp service event; ndmp client disconnect -->
1364 <event id="AUE_ndmp_disconnect" instance_of="AUE_generic_basic" header="0"
1365     idNo="62" omit="JNI">
1366     <title>NDMP Disconnect</title>
1367     <program>/usr/lib/ndmp/ndmpd</program>
1368     <see>ndmpd(1M)</see>
1369 </event>

1371 <!-- ndmp service event; ndmp backup -->
1372 <event id="AUE_ndmp_backup" header="0" idNo="63" omit="JNI">
1373     <title>NDMP Backup</title>
1374     <program>/usr/lib/ndmp/ndmpd</program>
1375     <see>ndmpd(1M)</see>
1376     <entry id="subject">
1377         <internal token="subject"/>
1378         <external opt="none"/>

```

```

1379     </entry>
1380     <entry id="source">
1381         <internal token="path"/>
1382         <external opt="required" type="char*" />
1383         <comment>path to be backed up</comment>
1384     </entry>
1385     <entry id="local_dest">
1386         <internal token="path"/>
1387         <external opt="optional" type="char*" />
1388         <comment>local path of backup destination</comment>
1389     </entry>
1390     <entry id="remote_dest">
1391         <internal token="in_peer"/>
1392         <external opt="optional" type="fd_t"/>
1393         <comment>remote ip address and port of backup destination</comment>
1394     </entry>
1395     <entry id="return">
1396         <internal token="return"/>
1397         <external opt="none"/>
1398     </entry>
1399 </event>

1401 <!-- ndmp service event; ndmp restore -->
1402 <event id="AUE_ndmp_restore" header="0" idNo="64" omit="JNI">
1403     <title>NDMP Restore</title>
1404     <program>/usr/lib/ndmp/ndmpd</program>
1405     <see>ndmpd(1M)</see>
1406     <entry id="subject">
1407         <internal token="subject"/>
1408         <external opt="none"/>
1409     </entry>
1410     <entry id="destination">
1411         <internal token="path"/>
1412         <external opt="required" type="char*" />
1413         <comment>path to restore to</comment>
1414     </entry>
1415     <entry id="local_source">
1416         <internal token="path"/>
1417         <external opt="optional" type="char*" />
1418         <comment>local path to restore from</comment>
1419     </entry>
1420     <entry id="remote_source">
1421         <internal token="in_peer"/>
1422         <external opt="optional" type="fd_t"/>
1423         <comment>remote ip address and port to restore from</comment>
1424     </entry>
1425     <entry id="return">
1426         <internal token="return"/>
1427         <external opt="none"/>
1428     </entry>
1429 </event>

1431 <!-- SMF related events -->
1432 <event id="AUE_smf_generic" type="generic" omit="always">
1433     <!--
1434     This is a template for the event types that have no tokens
1435     other than the header and return. There is no allowed_type
1436     list because the template is not externally visible due to the
1437     omit="always".
1438     -->
1439     <entry id="subject">
1440         <internal token="subject"/>
1441         <external opt="none"/>
1442     </entry>
1443     <entry id="auth_used">
1444         <internal token="uauth"/>

```

```

1445     <external opt="required" type="char **"/>
1446     <comment>authorization used</comment>
1447 </entry>
1448 <entry id="fmri">
1449   <internal token="fmri"/>
1450   <external opt="required" type="char **"/>
1451   <comment>name</comment>
1452 </entry>
1453 <entry id="return">
1454   <internal token="return"/>
1455   <external opt="none"/>
1456 </entry>
1457 </event>

1459 <event id="AUE_smf_generic_pg" type="generic" omit="always">
1460   <!--
1461   This is a template for the event types related to property groups.
1462   There is no allowed_type list because the template is not externally
1463   visible due to the omit="always".
1464   -->
1465   <entry id="subject">
1466     <internal token="subject"/>
1467     <external opt="none"/>
1468   </entry>
1469   <entry id="auth_used">
1470     <internal token="uauth"/>
1471     <external opt="required" type="char **"/>
1472     <comment>authorization used</comment>
1473   </entry>
1474   <entry id="fmri">
1475     <internal token="fmri"/>
1476     <external opt="required" type="char **"/>
1477   </entry>
1478   <entry id="type">
1479     <internal token="text"/>
1480     <external opt="required" type="char **"/>
1481     <comment>property group type</comment>
1482   </entry>
1483   <entry id="return">
1484     <internal token="return"/>
1485     <external opt="none"/>
1486   </entry>
1487 </event>

1489 <event id="AUE_smf_enable" instance_of="AUE_smf_generic" header="0"
1490   idNo="65" omit="JNI">
1491   <program>svc.configd(1M)</program>
1492   <see>svcadm(1M)</see>
1493 </event>
1494 <event id="AUE_smf_tmp_enable" instance_of="AUE_smf_generic" header="0"
1495   idNo="66" omit="JNI">
1496   <program>svc.configd(1M)</program>
1497   <see>svcadm(1M)</see>
1498 </event>
1499 <event id="AUE_smf_disable" instance_of="AUE_smf_generic" header="0"
1500   idNo="67" omit="JNI">
1501   <program>svc.configd(1M)</program>
1502   <see>svcadm(1M)</see>
1503 </event>
1504 <event id="AUE_smf_tmp_disable" instance_of="AUE_smf_generic" header="0"
1505   idNo="68" omit="JNI">
1506   <program>svc.configd(1M)</program>
1507   <see>svcadm(1M)</see>
1508 </event>
1509 <event id="AUE_smf_restart" instance_of="AUE_smf_generic" header="0"
1510   idNo="69" omit="JNI">

```

```

1511     <program>svc.configd(1M)</program>
1512     <see>svcadm(1M)</see>
1513 </event>
1514 <event id="AUE_smf_refresh" instance_of="AUE_smf_generic" header="0"
1515   idNo="70" omit="JNI">
1516   <program>svc.configd(1M)</program>
1517   <see>svcadm(1M)</see>
1518 </event>
1519 <event id="AUE_smf_clear" instance_of="AUE_smf_generic" header="0"
1520   idNo="71" omit="JNI">
1521   <program>svc.configd(1M)</program>
1522   <see>svcadm(1M)</see>
1523 </event>
1524 <event id="AUE_smf_degrade" instance_of="AUE_smf_generic" header="0"
1525   idNo="72" omit="JNI">
1526   <program>svc.configd(1M)</program>
1527   <see>svcadm(1M)</see>
1528 </event>
1529 <event id="AUE_smf_immediate_degrade" instance_of="AUE_smf_generic"
1530   header="0" idNo="73" omit="JNI">
1531   <program>svc.configd(1M)</program>
1532   <see>svcadm(1M)</see>
1533 </event>
1534 <event id="AUE_smf_maintenance" instance_of="AUE_smf_generic" header="0"
1535   idNo="74" omit="JNI">
1536   <program>svc.configd(1M)</program>
1537   <see>svcadm(1M)</see>
1538 </event>
1539 <event id="AUE_smf_immediate_maintenance" instance_of="AUE_smf_generic"
1540   header="0" idNo="75" omit="JNI">
1541   <program>svc.configd(1M)</program>
1542   <see>svcadm(1M)</see>
1543 </event>
1544 <event id="AUE_smf_immtmp_maintenance" instance_of="AUE_smf_generic"
1545   header="0" idNo="76" omit="JNI">
1546   <program>svc.configd(1M)</program>
1547   <see>svcadm(1M)</see>
1548 </event>
1549 <event id="AUE_smf_tmp_maintenance" instance_of="AUE_smf_generic" header="0"
1550   idNo="77" omit="JNI">
1551   <program>svc.configd(1M)</program>
1552   <see>svcadm(1M)</see>
1553 </event>
1554 <event id="AUE_smf_milestone" instance_of="AUE_smf_generic" header="0"
1555   idNo="78" omit="JNI">
1556   <program>svc.configd(1M)</program>
1557   <see>svcadm(1M)</see>
1558 </event>

1560 <event id="AUE_smf_create" instance_of="AUE_smf_generic" header="0"
1561   idNo="79" omit="JNI">
1562   <program>svc.configd(1M)</program>
1563   <see>svccfg(1M)</see>
1564 </event>
1565 <event id="AUE_smf_delete" instance_of="AUE_smf_generic" header="0"
1566   idNo="80" omit="JNI">
1567   <program>svc.configd(1M)</program>
1568   <see>svccfg(1M)</see>
1569 </event>

1571 <event id="AUE_smf_create_pg" instance_of="AUE_smf_generic_pg" header="0"
1572   idNo="81" omit="JNI">
1573   <program>svc.configd(1M)</program>
1574   <see>svccfg(1M)</see>
1575 </event>
1576 <event id="AUE_smf_create_npg" instance_of="AUE_smf_generic_pg" header="0"

```



```

1577     idNo="82" omit="JNI">
1578     <program>svc.configd(1M)</program>
1579     <see>svccfg(1M)</see>
1580 </event>
1581 <event id="AUE_smf_delete_pg" instance_of="AUE_smf_generic_pg" header="0"
1582     idNo="83" omit="JNI">
1583     <program>svc.configd(1M)</program>
1584     <see>svccfg(1M)</see>
1585 </event>
1586 <event id="AUE_smf_delete_npg" instance_of="AUE_smf_generic_pg" header="0"
1587     idNo="84" omit="JNI">
1588     <program>svc.configd(1M)</program>
1589     <see>svccfg(1M)</see>
1590 </event>

1592 <event id="AUE_smf_create_snap" header="0" idNo="85" omit="JNI">
1593     <program>svc.configd(1M)</program>
1594     <see>svccfg(1M)</see>
1595     <entry id="subject">
1596         <internal token="subject"/>
1597         <external opt="none"/>
1598     </entry>
1599     <entry id="auth_used">
1600         <internal token="uauth"/>
1601         <external opt="required" type="char **"/>
1602         <comment>authorization used</comment>
1603     </entry>
1604     <entry id="fmri">
1605         <internal token="fmri"/>
1606         <external opt="required" type="char **"/>
1607         <comment>name</comment>
1608     </entry>
1609     <entry id="name">
1610         <internal token="text"/>
1611         <external opt="required" type="char **"/>
1612         <comment>snapshot name</comment>
1613     </entry>
1614     <entry id="return">
1615         <internal token="return"/>
1616         <external opt="none"/>
1617     </entry>
1618 </event>
1619 <event id="AUE_smf_delete_snap" header="0" idNo="86" omit="JNI">
1620     <program>svc.configd(1M)</program>
1621     <see>svccfg(1M)</see>
1622     <entry id="subject">
1623         <internal token="subject"/>
1624         <external opt="none"/>
1625     </entry>
1626     <entry id="auth_used">
1627         <internal token="uauth"/>
1628         <external opt="required" type="char **"/>
1629         <comment>authorization used</comment>
1630     </entry>
1631     <entry id="fmri">
1632         <internal token="fmri"/>
1633         <external opt="required" type="char **"/>
1634         <comment>name</comment>
1635     </entry>
1636     <entry id="name">
1637         <internal token="text"/>
1638         <external opt="required" type="char **"/>
1639         <comment>snapshot name</comment>
1640     </entry>
1641     <entry id="return">
1642         <internal token="return"/>

```

```

1643         <external opt="none"/>
1644     </entry>
1645 </event>
1646 <event id="AUE_smf_attach_snap" header="0" idNo="87" omit="JNI">
1647     <program>svc.configd(1M)</program>
1648     <see>svccfg(1M)</see>
1649     <entry id="subject">
1650         <internal token="subject"/>
1651         <external opt="none"/>
1652     </entry>
1653     <entry id="auth_used">
1654         <internal token="uauth"/>
1655         <external opt="required" type="char **"/>
1656         <comment>authorization used</comment>
1657     </entry>
1658     <entry id="old_fmri">
1659         <internal token="fmri"/>
1660         <external opt="required" type="char **"/>
1661         <comment>old name</comment>
1662     </entry>
1663     <entry id="old_name">
1664         <internal token="text"/>
1665         <external opt="required" type="char **"/>
1666         <comment>old snapshot</comment>
1667     </entry>
1668     <entry id="new_fmri">
1669         <internal token="fmri"/>
1670         <external opt="required" type="char **"/>
1671         <comment>new name</comment>
1672     </entry>
1673     <entry id="new_name">
1674         <internal token="text"/>
1675         <external opt="required" type="char **"/>
1676         <comment>new snapshot</comment>
1677     </entry>
1678     <entry id="return">
1679         <internal token="return"/>
1680         <external opt="none"/>
1681     </entry>
1682 </event>

1684 <event id="AUE_smf_annotation" header="0" idNo="88" omit="JNI">
1685     <program>svc.configd(1M)</program>
1686     <see>svccfg(1M)</see>
1687     <entry id="subject">
1688         <internal token="subject"/>
1689         <external opt="none"/>
1690     </entry>
1691     <entry id="operation">
1692         <internal token="text"/>
1693         <external opt="required" type="char **"/>
1694         <comment>operation</comment>
1695     </entry>
1696     <entry id="file">
1697         <internal token="path"/>
1698         <external opt="required" type="char **"/>
1699         <comment>imported file</comment>
1700     </entry>
1701     <entry id="return">
1702         <internal token="return"/>
1703         <external opt="none"/>
1704     </entry>
1705 </event>

1707 <event id="AUE_smf_create_prop" header="0" idNo="89" omit="JNI">
1708     <program>svc.configd(1M)</program>

```

```

1709     <see>svccfg(1M)</see>
1710     <entry id="subject">
1711         <internal token="subject"/>
1712         <external opt="none"/>
1713     </entry>
1714     <entry id="auth_used">
1715         <internal token="uauth"/>
1716         <external opt="required" type="char *"/>
1717         <comment>authorization used</comment>
1718     </entry>
1719     <entry id="fmri">
1720         <internal token="fmri"/>
1721         <external opt="required" type="char *"/>
1722         <comment>name</comment>
1723     </entry>
1724     <entry id="type">
1725         <internal token="text"/>
1726         <external opt="required" type="char *"/>
1727         <comment>type</comment>
1728     </entry>
1729     <entry id="value">
1730         <internal token="text"/>
1731         <external opt="optional" type="char *"/>
1732         <comment>value</comment>
1733     </entry>
1734     <entry id="return">
1735         <internal token="return"/>
1736         <external opt="none"/>
1737     </entry>
1738 </event>

1740 <event id="AUE_smf_change_prop" header="0" idNo="90" omit="JNI">
1741     <program>svc.configd(1M)</program>
1742     <see>svccfg(1M)</see>
1743     <entry id="subject">
1744         <internal token="subject"/>
1745         <external opt="none"/>
1746     </entry>
1747     <entry id="auth_used">
1748         <internal token="uauth"/>
1749         <external opt="required" type="char *"/>
1750         <comment>authorization used</comment>
1751     </entry>
1752     <entry id="fmri">
1753         <internal token="fmri"/>
1754         <external opt="required" type="char *"/>
1755         <comment>name</comment>
1756     </entry>
1757     <entry id="type">
1758         <internal token="text"/>
1759         <external opt="required" type="char *"/>
1760         <comment>type</comment>
1761     </entry>
1762     <entry id="value">
1763         <internal token="text"/>
1764         <external opt="optional" type="char *"/>
1765         <comment>value</comment>
1766     </entry>
1767     <entry id="return">
1768         <internal token="return"/>
1769         <external opt="none"/>
1770     </entry>
1771 </event>
1772 <event id="AUE_smf_delete_prop" header="0" idNo="91" omit="JNI">
1773     <program>svc.configd(1M)</program>
1774     <see>svccfg(1M)</see>

```

```

1775     <entry id="subject">
1776         <internal token="subject"/>
1777         <external opt="none"/>
1778     </entry>
1779     <entry id="auth_used">
1780         <internal token="uauth"/>
1781         <external opt="required" type="char *"/>
1782         <comment>authorization used</comment>
1783     </entry>
1784     <entry id="fmri">
1785         <internal token="fmri"/>
1786         <external opt="required" type="char *"/>
1787         <comment>name</comment>
1788     </entry>
1789     <entry id="return">
1790         <internal token="return"/>
1791         <external opt="none"/>
1792     </entry>
1793 </event>

1795 <event id="AUE_smf_read_prop" instance_of="AUE_smf_generic" header="0"
1796     idNo="92" omit="JNI">
1797     <program>svc.configd(1M)</program>
1798     <see>svccfg(1M)</see>
1799 </event>

1801 <!-- CPUFreq related events -->

1803 <event id="AUE_cpu_ondemand" header="0" idNo="93" omit="JNI">
1804     <title>set CPU freq to minimal unless load increases</title>
1805     <program>/usr/lib/hal/hald-addon-cpufreq</program>
1806     <see>hald(1M)</see>
1807     <entry id="subject">
1808         <internal token="subject"/>
1809         <external opt="none"/>
1810     </entry>
1811     <entry id="auth_used">
1812         <internal token="uauth"/>
1813         <external opt="required" type="char *"/>
1814         <comment>authorization used</comment>
1815     </entry>
1816     <entry id="return">
1817         <internal token="return"/>
1818         <external opt="none"/>
1819     </entry>
1820 </event>
1821 <event id="AUE_cpu_performance" header="0" idNo="94" omit="JNI">
1822     <title>set CPU freq to Max</title>
1823     <program>/usr/lib/hal/hald-addon-cpufreq</program>
1824     <see>hald(1M)</see>
1825     <entry id="subject">
1826         <internal token="subject"/>
1827         <external opt="none"/>
1828     </entry>
1829     <entry id="auth_used">
1830         <internal token="uauth"/>
1831         <external opt="required" type="char *"/>
1832         <comment>authorization used</comment>
1833     </entry>
1834     <entry id="return">
1835         <internal token="return"/>
1836         <external opt="none"/>
1837     </entry>
1838 </event>
1839 <event id="AUE_cpu_threshold" header="0" idNo="95" omit="JNI">
1840     <title>set CPU frequency threshold percentage</title>

```

```

1841     <program>/usr/lib/hal/hald-addon-cpufreq</program>
1842     <see>hald(1M)</see>
1843     <entry id="subject">
1844         <internal token="subject"/>
1845         <external opt="none"/>
1846     </entry>
1847     <entry id="auth_used">
1848         <internal token="uauth"/>
1849         <external opt="required" type="char *"/>
1850         <comment>authorization used</comment>
1851     </entry>
1852     <entry id="threshold">
1853         <internal token="text"/>
1854         <external opt="required" type="int"/>
1855         <comment>threshold percent 1-100</comment>
1856     </entry>
1857     <entry id="return">
1858         <internal token="return"/>
1859         <external opt="none"/>
1860     </entry>
1861 </event>

1863 <!-- TPM events recorded by tcsd(8) -->

1865     <event id="AUE_generic_tpm" type="generic" omit="always">
1866         <entry id="subject">
1867             <internal token="subject"/>
1868             <external opt="none"/>
1869         </entry>
1870         <entry id="message">
1871             <internal token="text"/>
1872             <external opt="optional" type="msg tpm_e"/>
1873             <comment>TPM error message</comment>
1874         </entry>
1875         <entry id="return">
1876             <internal token="return"/>
1877             <external opt="none"/>
1878         </entry>
1879     </event>

1881     <event id="AUE_tpm_takeownership" instance_of="AUE_generic_tpm">
1882         header="0" idNo="99" omit="JNI">
1883         <title>TPM_TakeOwnership</title>
1884         <program>/usr/lib/tcsd</program>
1885         <see>tcsd(8)</see>
1886     </event>
1887     <event id="AUE_tpm_setoperatorauth" instance_of="AUE_generic_tpm">
1888         header="0" idNo="100" omit="JNI">
1889         <title>TPM_SetOperatorAuth</title>
1890         <program>/usr/lib/tcsd</program>
1891         <see>tcsd(8)</see>
1892     </event>
1893     <event id="AUE_tpm_setownerinstall" instance_of="AUE_generic_tpm">
1894         header="0" idNo="101" omit="JNI">
1895         <title>TPM_SetOwnerInstall</title>
1896         <program>/usr/lib/tcsd</program>
1897         <see>tcsd(8)</see>
1898     </event>
1899     <event id="AUE_tpm_selftestfull" instance_of="AUE_generic_tpm">
1900         header="0" idNo="102" omit="JNI">
1901         <title>TPM_SelfTestFull</title>
1902         <program>/usr/lib/tcsd</program>
1903         <see>tcsd(8)</see>
1904     </event>
1905     <event id="AUE_tpm_certifyselftest" instance_of="AUE_generic_tpm">
1906         header="0" idNo="103" omit="JNI">

```

```

1907         <title>TPM_CertifySelfTest</title>
1908         <program>/usr/lib/tcsd</program>
1909         <see>tcsd(8)</see>
1910     </event>
1911     <event id="AUE_tpm_continueselftest" instance_of="AUE_generic_tpm">
1912         header="0" idNo="104" omit="JNI">
1913         <title>TPM_ContinueSelfTest</title>
1914         <program>/usr/lib/tcsd</program>
1915         <see>tcsd(8)</see>
1916     </event>
1917     <event id="AUE_tpm_ownersetdisable" instance_of="AUE_generic_tpm">
1918         header="0" idNo="105" omit="JNI">
1919         <title>TPM_OwnerSetDisable</title>
1920         <program>/usr/lib/tcsd</program>
1921         <see>tcsd(8)</see>
1922     </event>
1923     <event id="AUE_tpm_ownerclear" instance_of="AUE_generic_tpm">
1924         header="0" idNo="106" omit="JNI">
1925         <title>TPM_OwnerClear</title>
1926         <program>/usr/lib/tcsd</program>
1927         <see>tcsd(8)</see>
1928     </event>
1929     <event id="AUE_tpm_disableownerclear" instance_of="AUE_generic_tpm">
1930         header="0" idNo="107" omit="JNI">
1931         <title>TPM_DisableOwnerClear</title>
1932         <program>/usr/lib/tcsd</program>
1933         <see>tcsd(8)</see>
1934     </event>
1935     <event id="AUE_tpm_forceclear" instance_of="AUE_generic_tpm">
1936         header="0" idNo="108" omit="JNI">
1937         <title>TPM_ForceClear</title>
1938         <program>/usr/lib/tcsd</program>
1939         <see>tcsd(8)</see>
1940     </event>
1941     <event id="AUE_tpm_disableforceclear" instance_of="AUE_generic_tpm">
1942         header="0" idNo="109" omit="JNI">
1943         <title>TPM_DisableForceClear</title>
1944         <program>/usr/lib/tcsd</program>
1945         <see>tcsd(8)</see>
1946     </event>
1947     <event id="AUE_tpm_physicaldisable" instance_of="AUE_generic_tpm">
1948         header="0" idNo="110" omit="JNI">
1949         <title>TPM_PhysicalDisable</title>
1950         <program>/usr/lib/tcsd</program>
1951         <see>tcsd(8)</see>
1952     </event>
1953     <event id="AUE_tpm_physicalenable" instance_of="AUE_generic_tpm">
1954         header="0" idNo="111" omit="JNI">
1955         <title>TPM_PhysicalEnable</title>
1956         <program>/usr/lib/tcsd</program>
1957         <see>tcsd(8)</see>
1958     </event>
1959     <event id="AUE_tpm_physicaldeactivate" instance_of="AUE_generic_tpm">
1960         header="0" idNo="112" omit="JNI">
1961         <title>TPM_PhysicalSetDeactivated</title>
1962         <program>/usr/lib/tcsd</program>
1963         <see>tcsd(8)</see>
1964     </event>
1965     <event id="AUE_tpm_settempdeactivated" instance_of="AUE_generic_tpm">
1966         header="0" idNo="113" omit="JNI">
1967         <title>TPM_SetTempDeactivated</title>
1968         <program>/usr/lib/tcsd</program>
1969         <see>tcsd(8)</see>
1970     </event>
1971     <event id="AUE_tpm_physicalpresence" instance_of="AUE_generic_tpm">
1972         header="0" idNo="114" omit="JNI">

```

```

1973     <title>TPM_PhysicalPresence</title>
1974     <program>/usr/lib/tcsd</program>
1975     <see>tcsd(8)</see>
1976 </event>
1977 <event id="AUE_tpm_fieldupgrade" instance_of="AUE_generic_tpm"
1978     header="0" idNo="115" omit="JNI">
1979     <title>TPM_FieldUpgrade</title>
1980     <program>/usr/lib/tcsd</program>
1981     <see>tcsd(8)</see>
1982 </event>
1983 <event id="AUE_tpm_resetlockvalue" instance_of="AUE_generic_tpm"
1984     header="0" idNo="116" omit="JNI">
1985     <title>TPM_ResetLockValue</title>
1986     <program>/usr/lib/tcsd</program>
1987     <see>tcsd(8)</see>
1988 </event>

1990 <!-- hotplug events recorded by hotplugd(1m) -->

1992 <event id="AUE_hotplug_state" header="0" idNo="117" omit="JNI">
1993     <title>change hotplug connection state</title>
1994     <program>/usr/lib/hotplugd</program>
1995     <see>hotplugd(1M)</see>
1996     <entry id="subject">
1997         <internal token="subject"/>
1998         <external opt="none"/>
1999     </entry>
2000     <entry id="auth_used">
2001         <internal token="uauth"/>
2002         <external opt="required" type="char **"/>
2003         <comment>authorization used</comment>
2004     </entry>
2005     <entry id="device_path">
2006         <internal token="path"/>
2007         <external opt="required" type="char **"/>
2008         <comment>device path</comment>
2009     </entry>
2010     <entry id="connection">
2011         <internal token="text"/>
2012         <external opt="required" type="char **"/>
2013         <comment>connector or port</comment>
2014     </entry>
2015     <entry id="new_state">
2016         <internal token="text"/>
2017         <external opt="required" type="char **"/>
2018         <comment>new connection state</comment>
2019     </entry>
2020     <entry id="old_state">
2021         <internal token="text"/>
2022         <external opt="required" type="char **"/>
2023         <comment>old connection state</comment>
2024     </entry>
2025     <entry id="return">
2026         <internal token="return"/>
2027         <external opt="none"/>
2028     </entry>
2029 </event>

2031 <event id="AUE_hotplug_set" header="0" idNo="118" omit="JNI">
2032     <title>set hotplug bus private options</title>
2033     <program>/usr/lib/hotplugd</program>
2034     <see>hotplugd(1M)</see>
2035     <entry id="subject">
2036         <internal token="subject"/>
2037         <external opt="none"/>
2038     </entry>

```

```

2039     <entry id="auth_used">
2040         <internal token="uauth"/>
2041         <external opt="required" type="char **"/>
2042         <comment>authorization used</comment>
2043     </entry>
2044     <entry id="device_path">
2045         <internal token="path"/>
2046         <external opt="required" type="char **"/>
2047         <comment>device path</comment>
2048     </entry>
2049     <entry id="connection">
2050         <internal token="text"/>
2051         <external opt="required" type="char **"/>
2052         <comment>connector or port</comment>
2053     </entry>
2054     <entry id="options">
2055         <internal token="text"/>
2056         <external opt="required" type="char **"/>
2057         <comment>bus private options</comment>
2058     </entry>
2059     <entry id="return">
2060         <internal token="return"/>
2061         <external opt="none"/>
2062     </entry>
2063 </event>

2065 <event id="AUE_ilb_create_healthcheck" header="0" idNo="120" omit="JNI">
2066     <title>Create Integrated Loadbalancer healthcheck object</title>
2067     <program>/usr/sbin/ilbadm</program>
2068     <see>ilbadm(1m)</see>
2069     <entry id="subject">
2070         <internal token="subject"/>
2071         <external opt="none"/>
2072     </entry>
2073     <entry id="auth_used">
2074         <internal token="uauth"/>
2075         <external opt="required" type="char **"/>
2076         <comment>authorization used</comment>
2077     </entry>
2078     <entry id="hc_test">
2079         <internal token="path"/>
2080         <external opt="required" type="char **"/>
2081         <comment>healthcheck type-PING,TCP,UDP or 3rd party script</comment>
2082     </entry>
2083     <entry id="hc_name">
2084         <internal token="text"/>
2085         <external opt="required" type="char **"/>
2086         <comment>healthcheck name</comment>
2087     </entry>
2088     <entry id="hc_timeout">
2089         <internal token="text"/>
2090         <external opt="required" type="int32_t" />
2091         <comment>timeout(secs) to kill a hung healthcheck probe
2092             - 0 means default value (see man page)
2093     </comment>
2094     </entry>
2095     <entry id="hc_count">
2096         <internal token="text"/>
2097         <external opt="required" type="int"/>
2098         <comment>number of times to run a health check probe
2099             before declaring a server to be dead - 0 means
2100             default value (see man page)
2101     </comment>
2102     </entry>
2103     <entry id="hc_interval">
2104         <internal token="text"/>

```

```

2105     <external opt="required" type="int32_t"/>
2106     <comment>time(secs) between 2 healthcheck events -
2107         0 means default value(see man page)
2108     </comment>
2109 </entry>
2110 <entry id="return">
2111     <internal token="return"/>
2112     <external opt="none"/>
2113 </entry>
2114 </event>

2116 <event id="AUE_ilb_delete_healthcheck" header="0" idNo="121" omit="JNI">
2117     <title>Delete Integrated Loadbalancer healthcheck object</title>
2118     <program>/usr/sbin/ilbadm</program>
2119     <see>ilbadm(1m)</see>
2120     <entry id="subject">
2121         <internal token="subject"/>
2122         <external opt="none"/>
2123     </entry>
2124     <entry id="auth_used">
2125         <internal token="uauth"/>
2126         <external opt="required" type="char *"/>
2127         <comment>authorization used</comment>
2128     </entry>
2129     <entry id="hc_name">
2130         <internal token="text"/>
2131         <external opt="required" type="char *"/>
2132         <comment>healthcheck name</comment>
2133     </entry>
2134     <entry id="return">
2135         <internal token="return"/>
2136         <external opt="none"/>
2137 </entry>
2138 </event>

2140 <event id="AUE_ilb_create_rule" header="0" idNo="122" omit="JNI">
2141     <title>Create Integrated Loadbalancer rule</title>
2142     <program>/usr/sbin/ilbadm</program>
2143     <see>ilbadm(1m)</see>
2144     <entry id="subject">
2145         <internal token="subject"/>
2146         <external opt="none"/>
2147     </entry>
2148     <entry id="auth_used">
2149         <internal token="uauth"/>
2150         <external opt="required" type="char *"/>
2151         <comment>authorization used</comment>
2152     </entry>
2153     <entry id="virtual_ipaddress_type,virtual_ipaddress">
2154         <internal token="in_remote"/>
2155         <external opt="required" type="int32_t,uint32_t[4]"/>
2156         <comment>LB virtual IP address</comment>
2157     </entry>
2158     <entry id="min_port">
2159         <internal token="iport"/>
2160         <external opt="required" type="uint16_t"/>
2161         <comment>minimum value in port range</comment>
2162     </entry>
2163     <entry id="max_port">
2164         <internal token="iport"/>
2165         <external opt="required" type="uint16_t"/>
2166         <comment>maximum value in port range - max=min means single
2167             port is specified
2168         </comment>
2169     </entry>
2170     <entry id="protocol">

```

```

2171     <internal token="text"/>
2172     <external opt="required" type="char *"/>
2173     <comment>protocol</comment>
2174 </entry>
2175 <entry id="algo_optype">
2176     <internal token="text"/>
2177     <external opt="required" type="char *"/>
2178     <comment>[rr,hip,hipp,hipv],[dsr,nat,half-nat]</comment>
2179 </entry>
2180 <entry id="proxy_src_min_type,proxy_src_min">
2181     <internal token="in_remote"/>
2182     <external opt="optional" type="int32_t,uint32_t[4]"/>
2183     <comment>min value for proxy source address for NAT</comment>
2184 </entry>
2185 <entry id="proxy_src_max_type,proxy_src_max">
2186     <internal token="in_remote"/>
2187     <external opt="optional" type="int32_t,uint32_t[4]"/>
2188     <comment>max value in proxy source address range for NAT
2189         - max=min means single address is specified
2190     </comment>
2191 </entry>
2192 <entry id="persist_mask">
2193     <internal token="text"/>
2194     <external opt="required" type="char *"/>
2195     <comment>prefix length</comment>
2196 </entry>
2197 <entry id="hcname">
2198     <internal token="text"/>
2199     <external opt="optional" type="char *"/>
2200     <comment>healthcheck name</comment>
2201 </entry>
2202 <entry id="hcport">
2203     <internal token="text"/>
2204     <external opt="optional" type="char *"/>
2205     <comment>healthcheck port - ANY(dynamically determined by ilbd)
2206         or a positive integer
2207     </comment>
2208 </entry>
2209 <entry id="connndrain_timeout">
2210     <internal token="text"/>
2211     <external opt="required" type="uint32_t"/>
2212     <comment>connection timeout for NAT/half-NAT in sec. - 0 means
2213         no forced removal)
2214     </comment>
2215 </entry>
2216 <entry id="nat_timeout">
2217     <internal token="text"/>
2218     <external opt="required" type="uint32_t"/>
2219     <comment>nat entry timeout for NAT/half-NAT in sec - 0 means
2220         default value(see man page)
2221     </comment>
2222 </entry>
2223 <entry id="persist_timeout">
2224     <internal token="text"/>
2225     <external opt="required" type="uint32_t"/>
2226     <comment>session persistence mapping in sec - 0 means no
2227         persistence
2228     </comment>
2229 </entry>
2230 <entry id="server_group">
2231     <internal token="text"/>
2232     <external opt="required" type="char *"/>
2233     <comment>server group name</comment>
2234 </entry>
2235 <entry id="rule_name">
2236     <internal token="text"/>

```

```

2237     <external opt="required" type="char **"/>
2238     <comment>rule name</comment>
2239   </entry>
2240   <entry id="return">
2241     <internal token="return"/>
2242     <external opt="none"/>
2243   </entry>
2244 </event>

2246 <!-- generic ILB rule event -->

2248 <event id="AUE_generic_ILB_rule" type="generic" omit="always">
2249   <entry id="subject">
2250     <internal token="subject"/>
2251     <external opt="none"/>
2252   </entry>
2253   <entry id="auth_used">
2254     <internal token="uauth"/>
2255     <external opt="required" type="char **"/>
2256     <comment>authorization used</comment>
2257   </entry>
2258   <entry id="rule_name">
2259     <internal token="text"/>
2260     <external opt="required" type="char **"/>
2261     <comment>rule name - "all" means all rules</comment>
2262   </entry>
2263   <entry id="return">
2264     <internal token="return"/>
2265     <external opt="none"/>
2266   </entry>
2267 </event>

2269 <!-- instances of the ILB generic rule event. -->
2270 <event id="AUE_ilb_delete_rule" instance_of="AUE_generic_ILB_rule"
2271   header="0" idNo="123">
2272   <title>Delete Integrated Loadbalancer rule</title>
2273   <program>/usr/sbin/ilbadm</program>
2274   <see>ilbadm(1m)</see>
2275 </event>

2277 <event id="AUE_ilb_disable_rule" instance_of="AUE_generic_ILB_rule"
2278   header="0" idNo="124">
2279   <title>Disable Integrated Loadbalancer rule</title>
2280   <program>/usr/sbin/ilbadm</program>
2281   <see>ilbadm(1m)</see>
2282 </event>

2284 <event id="AUE_ilb_enable_rule" instance_of="AUE_generic_ILB_rule"
2285   header="0" idNo="125">
2286   <title>Enable Integrated Loadbalancer rule</title>
2287   <program>/usr/sbin/ilbadm</program>
2288   <see>ilbadm(1m)</see>
2289 </event>

2291 <event id="AUE_ilb_add_server" header="0" idNo="126" omit="JNI">
2292   <title>Add server to Integrated Loadbalancer</title>
2293   <program>/usr/sbin/ilbadm</program>
2294   <see>ilbadm(1m)</see>
2295   <entry id="subject">
2296     <internal token="subject"/>
2297     <external opt="none"/>
2298   </entry>
2299   <entry id="auth_used">
2300     <internal token="uauth"/>
2301     <external opt="required" type="char **"/>
2302     <comment>authorization used</comment>

```

```

2303   </entry>
2304   <entry id="server_ipaddress_type,server_ipaddress">
2305     <internal token="in_remote"/>
2306     <external opt="required" type="int32_t,uint32_t[4]"/>
2307     <comment>IP address</comment>
2308   </entry>
2309   <entry id="server_id">
2310     <internal token="text"/>
2311     <external opt="optional" type="char **"/>
2312     <comment>serverid that corresponds IP address - empty
2313       if authorization fails, user specified IP address
2314       is invalid or server cannot be added because
2315       server group is full
2316     </comment>
2317   </entry>
2318   <entry id="server_group">
2319     <internal token="text"/>
2320     <external opt="required" type="char **"/>
2321     <comment>server group name</comment>
2322   </entry>
2323   <entry id="server_minport">
2324     <internal token="iport"/>
2325     <external opt="optional" type="uint16_t" />
2326     <comment>server's minimum value in port range - empty
2327       means default value (see man page)
2328     </comment>
2329   </entry>
2330   <entry id="server_maxport">
2331     <internal token="iport"/>
2332     <external opt="optional" type="uint16_t" />
2333     <comment>server's maximum value in port range - empty
2334       means default value(see man page)
2335     </comment>
2336   </entry>
2337   <entry id="return">
2338     <internal token="return"/>
2339     <external opt="none"/>
2340   </entry>
2341 </event>

2343 <event id="AUE_ilb_disable_server" header="0" idNo="127" omit="JNI">
2344   <title>Disable server to Integrated Loadbalancer</title>
2345   <program>/usr/sbin/ilbadm</program>
2346   <see>ilbadm(1m)</see>
2347   <entry id="subject">
2348     <internal token="subject"/>
2349     <external opt="none"/>
2350   </entry>
2351   <entry id="auth_used">
2352     <internal token="uauth"/>
2353     <external opt="required" type="char **"/>
2354     <comment>authorization used</comment>
2355   </entry>
2356   <entry id="server_id">
2357     <internal token="text"/>
2358     <external opt="required" type="char **"/>
2359     <comment>serverid</comment>
2360   </entry>
2361   <entry id="server_ipaddress_type,server_ipaddress">
2362     <internal token="in_remote"/>
2363     <external opt="optional" type="int32_t,uint32_t[4]"/>
2364     <comment>IPaddr corresponding to the serverid - empty
2365       if authorization fails, or user specified serverid
2366       is nonexistent
2367     </comment>
2368   </entry>

```

```

2369     <entry id="return">
2370         <internal token="return"/>
2371         <external opt="none"/>
2372     </entry>
2373 </event>

2375 <event id="AUE_ilb_enable_server" header="0" idNo="128" omit="JNI">
2376     <title>Enable server to Integrated Loadbalancer</title>
2377     <program>/usr/sbin/ilbadm</program>
2378     <see>ilbadm(1m)</see>
2379     <entry id="subject">
2380         <internal token="subject"/>
2381         <external opt="none"/>
2382     </entry>
2383     <entry id="auth_used">
2384         <internal token="uauth"/>
2385         <external opt="required" type="char *"/>
2386         <comment>authorization used</comment>
2387     </entry>
2388     <entry id="server_id">
2389         <internal token="text"/>
2390         <external opt="required" type="char *"/>
2391         <comment>serverid</comment>
2392     </entry>
2393     <entry id="server_ipaddress_type,server_ipaddress">
2394         <internal token="in_remote"/>
2395         <external opt="optional" type="int32_t,uint32_t[4]"/>
2396         <comment>IPaddr corresponding to the serverid - empty
2397             if authorization fails, or user specified serverid
2398             is nonexistent
2399         </comment>
2400     </entry>
2401     <entry id="return">
2402         <internal token="return"/>
2403         <external opt="none"/>
2404     </entry>
2405 </event>

2407 <event id="AUE_ilb_remove_server" header="0" idNo="129" omit="JNI">
2408     <title>Remove server from Integrated Loadbalancer</title>
2409     <program>/usr/sbin/ilbadm</program>
2410     <see>ilbadm(1m)</see>
2411     <entry id="subject">
2412         <internal token="subject"/>
2413         <external opt="none"/>
2414     </entry>
2415     <entry id="auth_used">
2416         <internal token="uauth"/>
2417         <external opt="required" type="char *"/>
2418         <comment>authorization used</comment>
2419     </entry>
2420     <entry id="server_id">
2421         <internal token="text"/>
2422         <external opt="required" type="char *"/>
2423         <comment>serverid</comment>
2424     </entry>
2425     <entry id="server_group">
2426         <internal token="text"/>
2427         <external opt="required" type="char *"/>
2428         <comment>server group name</comment>
2429     </entry>
2430     <entry id="server_ipaddress_type,server_ipaddress">
2431         <internal token="in_remote"/>
2432         <external opt="optional" type="int32_t,uint32_t[4]"/>
2433         <comment>IPaddr corresponding to serverid - empty
2434             if authorization fails or user specified serverid

```

```

2435         serverid is nonexistent
2436     </comment>
2437 </entry>
2438     <entry id="return">
2439         <internal token="return"/>
2440         <external opt="none"/>
2441     </entry>
2442 </event>

2444 <event id="AUE_ilb_create_servergroup" header="0" idNo="130" omit="JNI">
2445     <title>Create server group for Integrated Loadbalancer</title>
2446     <program>/usr/sbin/ilbadm</program>
2447     <see>ilbadm(1m)</see>
2448     <entry id="subject">
2449         <internal token="subject"/>
2450         <external opt="none"/>
2451     </entry>
2452     <entry id="auth_used">
2453         <internal token="uauth"/>
2454         <external opt="required" type="char *"/>
2455         <comment>authorization used</comment>
2456     </entry>
2457     <entry id="server_group">
2458         <internal token="text"/>
2459         <external opt="required" type="char *"/>
2460         <comment>server group name</comment>
2461     </entry>
2462     <entry id="return">
2463         <internal token="return"/>
2464         <external opt="none"/>
2465     </entry>
2466 </event>

2468 <event id="AUE_ilb_delete_servergroup" header="0" idNo="131" omit="JNI">
2469     <title>Delete server group from Integrated Loadbalancer</title>
2470     <program>/usr/sbin/ilbadm</program>
2471     <see>ilbadm(1m)</see>
2472     <entry id="subject">
2473         <internal token="subject"/>
2474         <external opt="none"/>
2475     </entry>
2476     <entry id="auth_used">
2477         <internal token="uauth"/>
2478         <external opt="required" type="char *"/>
2479         <comment>authorization used</comment>
2480     </entry>
2481     <entry id="server_group">
2482         <internal token="text"/>
2483         <external opt="required" type="char *"/>
2484         <comment>server group name</comment>
2485     </entry>
2486     <entry id="return">
2487         <internal token="return"/>
2488         <external opt="none"/>
2489     </entry>
2490 </event>

2492 <event id="AUE_nwam_enable" header="0" idNo="132" omit="JNI">
2493     <entry id="subject">
2494         <internal token="subject"/>
2495         <external opt="none"/>
2496     </entry>
2497     <entry id="profile_type">
2498         <internal token="text"/>
2499         <external opt="required" type="char *"/>
2500         <comment>Type of profile being enabled</comment>

```

```

2501     </entry>
2502     <entry id="profile_name">
2503         <internal token="text"/>
2504         <external opt="required" type="char *"/>
2505         <comment>Name of profile being enabled</comment>
2506     </entry>
2507     <entry id="return">
2508         <internal token="return"/>
2509         <external opt="none"/>
2510     </entry>
2511 </event>

2513 <event id="AUE_nwam_disable" header="0" idNo="133" omit="JNI">
2514     <entry id="subject">
2515         <internal token="subject"/>
2516         <external opt="none"/>
2517     </entry>
2518     <entry id="profile_type">
2519         <internal token="text"/>
2520         <external opt="required" type="char *"/>
2521         <comment>Type of profile being disabled</comment>
2522     </entry>
2523     <entry id="profile_name">
2524         <internal token="text"/>
2525         <external opt="required" type="char *"/>
2526         <comment>Name of profile being disabled</comment>
2527     </entry>
2528     <entry id="return">
2529         <internal token="return"/>
2530         <external opt="none"/>
2531     </entry>
2532 </event>

2534 <event id="AUE_netcfg_update" header="0" idNo="134" omit="JNI">
2535     <entry id="subject">
2536         <internal token="subject"/>
2537         <external opt="none"/>
2538     </entry>
2539     <entry id="parent_file">
2540         <internal token="text"/>
2541         <external opt="required" type="char *"/>
2542         <comment>Back-end data file being updated</comment>
2543     </entry>
2544     <entry id="object_name">
2545         <internal token="text"/>
2546         <external opt="required" type="char *"/>
2547         <comment>Name of object being updated</comment>
2548     </entry>
2549     <entry id="return">
2550         <internal token="return"/>
2551         <external opt="none"/>
2552     </entry>
2553 </event>

2555 <event id="AUE_netcfg_remove" header="0" idNo="135" omit="JNI">
2556     <entry id="subject">
2557         <internal token="subject"/>
2558         <external opt="none"/>
2559     </entry>
2560     <entry id="parent_file">
2561         <internal token="text"/>
2562         <external opt="required" type="char *"/>
2563         <comment>Back-end data file being modified</comment>
2564     </entry>
2565     <entry id="object_name">
2566         <internal token="text"/>

```

```

2567         <external opt="required" type="char *"/>
2568         <comment>Name of object being removed</comment>
2569     </entry>
2570     <entry id="return">
2571         <internal token="return"/>
2572         <external opt="none"/>
2573     </entry>
2574 </event>

2576 <!-- add new events here with the next higher idNo -->
2577 <!-- Highest idNo is 135, so next is 136, then fix this comment -->
2578 <!-- end of C Only events -->

2580 <!--
2581     token definitions are partially implemented. All they do for now
2582     is create a list of defined token names. In the future they may
2583     become a way of describing token structure.
2584 -->

2586 <token id="acl">
2587 </token>
2588 <token id="arbitrary">
2589 </token>
2590 <token id="arg">
2591 </token>
2592 <token id="attr">
2593 </token>
2594 <token id="command">
2595 </token>
2596 <token id="command_alt">
2597 </token>
2598 <token id="date">
2599 </token>
2600 <token id="exec_args">
2601 </token>
2602 <token id="exec_env">
2603 </token>
2604 <token id="exit">
2605 </token>
2606 <token id="file">
2607 </token>
2608 <token id="fmri">
2609 </token>
2610 <token id="groups">
2611 </token>
2612 <token id="secflags">
2613 </token>
2614 #endif /* !codereview */
2615 <!--
2616     the iport token take a single argument of type uint16_t
2617     if there are any other tokens following it that have arguments
2618     the last of the iport tokens in the event description must
2619     be followed by a dummy iport token that is optional.
2620     This is to ensure proper structure alignment across all
2621     compilers and architectures.
2622 -->
2623 <token id="iport">
2624 </token>
2625 <!-- pseudo token; in_addr and in_port of peer -->
2626 <token id="in_peer">
2627 </token>
2628 <!-- pseudo token; specified in_addr -->
2629 <token id="in_remote">
2630 </token>
2631 <token id="ipc">
2632 </token>

```



```

2633 <token id="ipc_perm">
2634 </token>
2635 <token id="label">
2636 </token>
2637 <token id="newgroups">
2638 </token>
2639 <token id="opaque">
2640 </token>
2641 <token id="path">
2642 </token>
2643 <!-- pseudo token; path list generates 0 or more path tokens -->
2644 <token id="path_list">
2645 </token>
2646 <!--
2647     privilege token is implemented as one of the pseudo tokens
2648     priv_limit, priv_effective, or priv_inherit
2650 <token id="privilege">
2651 </token>
2652 -->
2653 <token id="priv_effective">
2654 </token>
2655 <token id="priv_inherit">
2656 </token>
2657 <token id="priv_limit">
2658 </token>
2659 <token id="process">
2660 </token>
2661 <token id="return">
2662 </token>
2663 <token id="seq">
2664 </token>
2665 <token id="socket">
2666 </token>
2667 <token id="socket-inet">
2668 </token>
2669 <token id="subject">
2670 </token>
2671 <token id="text">
2672 </token>
2673 <token id="tid">
2674 </token>
2675 <token id="uauth">
2676 </token>
2677 <token id="user">
2678 </token>
2679 <token id="zonename">
2680 </token>
2682 <!--
2683 error value list for return values with success/fail code of fail.
2684 These values start at 1000 so praudit can tell the difference
2685 between the libbsm/common/audit_*.c broken error values and
2686 the new adt_error value list. It is public so that praudit
2687 can find it.
2689 praudit outputs "failure" %s for these strings, so there is
2690 no need to use words such as "failed" in the message.
2692 ** Add to the end only to maintain validity across versions of
2693 the audit log. **
2694 -->
2696 <msg_list id="fail_value" header="0" start="1000" public="true">
2697 <msg id="PW_ATTR">Attribute update</msg>
2698 <msg id="PW">Password update</msg>

```

```

2699 <msg id="USERNAME">bad username</msg>
2700 <msg id="AUTH">authorization failed</msg>
2701 <msg id="UID">bad uid</msg>
2702 <msg id="UNKNOWN">unknown failure</msg>
2703 <msg id="EXPIRED">password expired</msg>
2704 <msg id="ACCOUNT_LOCKED">Account is locked</msg>
2705 <msg id="BAD_DIALUP">Bad dial up</msg>
2706 <msg id="BAD_ID">Invalid ID</msg>
2707 <msg id="BAD_PW">Invalid password</msg>
2708 <msg id="CONSOLE">Not on console</msg>
2709 <msg id="MAX_TRIES">Too many failed attempts</msg>
2710 <msg id="PROTOCOL_FAILURE">Protocol failure</msg>
2711 <msg id="EXCLUDED_USER">Excluded user</msg>
2712 <msg id="ANON_USER">No anonymous</msg>
2713 <msg id="BAD_CMD">Invalid command</msg>
2714 <msg id="BAD_TTY">Standard input not a tty line</msg>
2715 <msg id="PROGRAM">Program failure</msg>
2716 <msg id="CHDIR_FAILED">chdir to home directory</msg>
2717 <msg id="INPUT_OVERFLOW">Input line too long.</msg>
2718 <msg id="DEVICE_PERM">login device override</msg>
2719 <msg id="AUTH_BYPASS">authorization bypass</msg>
2720 <msg id="LOGIN_DISABLED">login disabled</msg>
2721 </msg_list>
2723 <!--
2724     The following empty list is used for PAM errors; the "start"
2725     value is used by praudit to know to use the PAM infrastructure
2726     for generating error strings
2727 -->
2728 <msg_list id="fail_pam" header="0" start="2000" public="true">
2729 </msg_list>
2731 <!--
2732 This is still in use by SMC. See AUE_generic_login. When
2733 either SMC is fixed to stop using this, or SMC goes away.
2734 REMOVE this stuff and the corresponding AUE_generic_login
2735 message field.
2737 Message list for the various authentication events, such
2738 as AUE_login and AUE_admin_authenticate. Add new entries
2739 at the end. The order of msg_list entries and the order
2740 of msg entries both affect the names in ad.t.h and the value
2741 of the associated enumerated types.
2743 Each of these messages except NO_MSG is also in the failure_attribute
2744 list; the difference is that the messages below use a text token
2745 in the audit record, while the failure_attribute messages are
2746 associated with the return value of the return token.
2748 This list is deprecated; please don't use text tokens for error
2749 messages.
2750 -->
2752 <msg_list id="login_text" header="0" deprecated="true">
2753 <msg id="NO_MSG"></msg>
2754 <msg id="ACCOUNT_LOCKED">Account is locked</msg>
2755 <msg id="BAD_DIALUP">Bad dial up</msg>
2756 <msg id="BAD_ID">Invalid ID</msg>
2757 <msg id="BAD_PW">Invalid password</msg>
2758 <msg id="CONSOLE">Not on console</msg>
2759 <msg id="MAX_TRIES">Too many failed attempts</msg>
2760 <msg id="PROTOCOL_FAILURE">Protocol failure</msg>
2761 <msg id="EXCLUDED_USER">Excluded user</msg>
2762 <msg id="ANON_USER">No anonymous</msg>
2763 </msg_list>

```

```

2765 <!-- msg list for uadmin(lm) fcn argument (next action, see uadmin(2)) -->
2766 <msg_list id="uadmin_fcn" header="0" start="3000" public="true">
2767 <msg id="AD_HALT">Halt the processor(s)</msg>
2768 <msg id="AD_POWEROFF">Halt the processor(s) and turn off the power</msg>
2769 <msg id="AD_BOOT">Reboot the system using the kernel file</msg>
2770 <msg id="AD_IBOOT">Interactive reboot</msg>
2771 <msg id="AD_SUSPEND_TO_DISK">Save the system state to the state file</ms
2772 <msg id="AD_CHECK_SUSPEND_TO_DISK">Check if system supports suspend to d
2773 <msg id="AD_FORCE">Force suspend to disk even when threads of user
2774 applications are not suspendable</msg>
2775 <msg id="AD_SUSPEND_TO_RAM">Save the system state to memory</msg>
2776 <msg id="AD_CHECK_SUSPEND_TO_RAM">Check if system supports suspend to me
2777 <msg id="AD_SBOOT">Single-user reboot</msg>
2778 <msg id="AD_SIBOOT">Single-user interactive reboot</msg>
2779 <msg id="AD_NOSYNC">Do not sync filesystems on next A_DUMP</msg>
2780 <msg id="AD_FASTREBOOT">Reboot bypassing BIOS and boot loader</msg>
2781 <msg id="AD_FASTREBOOT_DRYRUN">Check if system supports reboot bypassing
2782 <msg id="AD_UPDATE_BOOT_CONFIG">Update boot configuration parameters</ms
2783 <msg id="AD_REUSEINIT">Prepare for AD_REUSABLE</msg>
2784 <msg id="AD_REUSABLE">Create reusable statefile</msg>
2785 <msg id="AD_REUSEFINI">Revert to normal CPR mode (not reusable)</msg>
2786 <msg id="AD_FTRACE_START">ftrace start</msg>
2787 <msg id="AD_FTRACE_STOP">ftrace stop</msg>
2788 </msg_list>

2790 <!--
2791 msg list for TPM errors that will be reported by tcstd(8).
2792 This list must match the order of the TPM_E_* error codes defined
2793 in /usr/include/tss/tpm_error.h (SUNWtss package)
2794 -->
2795 <msg_list id="tpm_e" header="0" start="4000" public="true">
2796 <msg id="AUTHFAIL">Authentication failed</msg>
2797 <msg id="BADINDEX">The index to a PCR, DIR or other register is incorrec
2798 <msg id="BAD_PARAMETER">One or more parameter is bad</msg>
2799 <msg id="AUDITFAILURE">auditing of the operation failed.</msg>
2800 <msg id="CLEAR_DISABLED">clear operations now physical access</msg>
2801 <msg id="DEACTIVATED">The TPM is deactivated</msg>
2802 <msg id="DISABLED">The TPM is disabled</msg>
2803 <msg id="DISABLED_CMD">The target command has been disabled</msg>
2804 <msg id="FAIL">The operation failed</msg>
2805 <msg id="BAD_ORDINAL">The ordinal was unknown or inconsistent</msg>
2806 <msg id="INSTALL_DISABLED">The ability to install an owner is disabled</
2807 <msg id="INVALID_KEYHANDLE">The key handle can not be interpreted</msg>
2808 <msg id="KEYNOTFOUND">The key handle points to an invalid key</msg>
2809 <msg id="INAPPROPRIATE_ENC">Unacceptable encryption scheme</msg>
2810 <msg id="MIGRATEFAIL">Migration authorization failed</msg>
2811 <msg id="INVALID_PCR_INFO">PCR information could not be interpreted</msg>
2812 <msg id="NOSPACED">No room to load key.</msg>
2813 <msg id="NOSRK">There is no SRK set</msg>
2814 <msg id="NOTSEALED_BLOB">An encrypted blob is invalid or was
2815 not created by this TPM</msg>
2816 <msg id="OWNER_SET">There is already an Owner </msg>
2817 <msg id="RESOURCES">The TPM has insufficient internal resources</msg>
2818 <msg id="SHORTRANDOM">A random string was too short</msg>
2819 <msg id="SIZE">The TPM does not have the space to perform the operation.
2820 <msg id="WRONGPCRVAL">The named PCR value does not match the current PCR
2821 <msg id="BAD_PARAM_SIZE">The paramSize argument has the incorrect value
2822 <msg id="SHA_THREAD">There is no existing SHA-1 thread.</msg>
2823 <msg id="SHA_ERROR">SHA-1 thread encountered an error.</msg>
2824 <msg id="FAILEDSELFTEST">Self-test has failed and the TPM has shutdown.<
2825 <msg id="AUTH2FAIL">The auth for the second key failed authorization</ms
2826 <msg id="BADTAG">The tag value sent to for a command is invalid</msg>
2827 <msg id="IOERROR">An IO error occurred transmitting information to the T
2828 <msg id="ENCRYPT_ERROR">The encryption process had a problem.</msg>
2829 <msg id="DECRYPT_ERROR">The decryption process did not complete.</msg>
2830 <msg id="INVALID_AUTHHANDLE">An invalid handle was used.</msg>

```

```

2831 <msg id="NO_ENDORSEMENT">The TPM does not a EK installed</msg>
2832 <msg id="INVALID_KEYUSAGE">The usage of a key is not allowed</msg>
2833 <msg id="WRONG_ENTITYTYPE">The submitted entity type is not allowed</msg>
2834 <msg id="INVALID_POSTINIT">The command was received in the wrong sequenc
2835 <msg id="INAPPROPRIATE_SIG">Signed data cannot include additional DER in
2836 <msg id="BAD_KEY_PROPERTY">The key properties are not supported by this
2837 <msg id="BAD_MIGRATION">The migration properties of this key are incorre
2838 <msg id="BAD_SCHEME">Incorrect signature or encryption scheme</msg>
2839 <msg id="BAD_DATASIZE">The size of the data parameter is bad</msg>
2840 <msg id="BAD_MODE">A mode parameter is bad</msg>
2841 <msg id="BAD_PRESENCE">physicalPresence or physicalPresenceLock bits hav
2842 <msg id="BAD_VERSION">The TPM cannot perform this version of the capabil
2843 <msg id="NO_WRAP_TRANSPORT">The TPM does not allow for wrapped transport
2844 <msg id="AUDITFAIL_UNSUCCESSFUL">TPM audit construction failed for faile
2845 <msg id="AUDITFAIL_SUCCESSFUL">TPM audit construction failed for success
2846 <msg id="NOTRESETTABLE">PCR register does not have the resettable attribu
2847 <msg id="NOTLOCAL">PCR register requires locality</msg>
2848 <msg id="BAD_TYPE">Make identity blob not properly typed</msg>
2849 <msg id="INVALID_RESOURCE">Resource type does not match actual resource<
2850 <msg id="NOTFIPS">Command only available when TPM is in FIPS mode</msg>
2851 <msg id="INVALID_FAMILY">The command is attempting to use an invalid fam
2852 <msg id="NO_NV_PERMISSION">The permission to manipulate the NV storage i
2853 <msg id="REQUIRES_SIGN">The operation requires a signed command</msg>
2854 <msg id="KEY_NOTSUPPORTED">Wrong operation to load an NV key</msg>
2855 <msg id="AUTH_CONFLICT">NV_LoadKey blob requires both owner and blob aut
2856 <msg id="AREA_LOCKED">The NV area is locked and not writable</msg>
2857 <msg id="BAD_LOCALITY">The locality is incorrect for the attempted opera
2858 <msg id="READ_ONLY">The NV area is read only and can't be written to</ms
2859 <msg id="PER_NOWRITE">There is no protection on the write to the NV area
2860 <msg id="FAMILYCOUNT">The family count value does not match</msg>
2861 <msg id="WRITE_LOCKED">The NV area has already been written to</msg>
2862 <msg id="BAD_ATTRIBUTES">The NV area attributes conflict</msg>
2863 <msg id="INVALID_STRUCTURE">The tag and version are invalid or inconsist
2864 <msg id="KEY_OWNER_CONTROL">The key evicted by the TPM Owner.</msg>
2865 <msg id="BAD_COUNTER">The counter handle is incorrect</msg>
2866 <msg id="NOT_FULLWRITE">The write is not a complete write of the area</m
2867 <msg id="CONTEXT_GAP">The gap between saved context counts is too large<
2868 <msg id="MAXNVWRITES">Max number of NV writes without owner has been exc
2869 <msg id="NOOPERATOR">No operator AuthData value is set</msg>
2870 <msg id="RESOURCEMISSING">The resource pointed to by context is not load
2871 <msg id="DELEGATE_LOCK">The delegate administration is locked</msg>
2872 <msg id="DELEGATE_FAMILY">Attempt to manage a family other than the dele
2873 <msg id="DELEGATE_ADMIN">Delegation table management not enabled</msg>
2874 <msg id="TRANSPORT_NOTEXCLUSIVE">Command executed outside of exclusive t
2875 <msg id="OWNER_CONTROL">Attempt to context save a owner evict controlled
2876 <msg id="DAA_RESOURCES">DAA command has no resources available to execut
2877 <msg id="DAA_INPUT_DATA0">The consistency check on DAA parameter inputDa
2878 <msg id="DAA_INPUT_DATA1">The consistency check on DAA parameter inputDa
2879 <msg id="DAA_ISSUER_SETTINGS">The consistency check on DAA issuerSetting
2880 <msg id="DAA_TPM_SETTINGS">The consistency check on DAA_tpmSpecific has
2881 <msg id="DAA_STAGE">Atomic process indicated by DAA command is not the e
2882 <msg id="DAA_ISSUER_VALIDITY">Inconsistent issuer validity</msg>
2883 <msg id="DAA_WRONG_W">The consistency check on w has failed.</msg>
2884 <msg id="BAD_HANDLE">The handle is incorrect</msg>
2885 <msg id="BAD_DELEGATE">Delegation is not correct</msg>
2886 <msg id="BADCONTEXT">The context blob is invalid</msg>
2887 <msg id="TOOMANYCONTEXTS">Too many contexts held by the TPM</msg>
2888 <msg id="MA_TICKET_SIGNATURE">Migration authority signature validation f
2889 <msg id="MA_DESTINATION">Migration destination not authenticated</msg>
2890 <msg id="MA_SOURCE">Migration source incorrect</msg>
2891 <msg id="MA_AUTHORITY">Incorrect migration authority</msg>
2892 <msg id="PERMANENTEK">Attempt to revoke the EK and the EK is not revocab
2893 <msg id="BAD_SIGNATURE">Bad signature of CMK ticket</msg>
2894 <msg id="NOCONTEXTSPACE">There is no room in the context list for additi
2895 <msg id="RETRY">The TPM is too busy to respond to the command immediatel
2896 <msg id="NEEDS_SELFTEST">SelfTestFull has not been run</msg>

```

```
2897     <msg id="DOING_SELFTEST">The TPM is currently executing a full selftest<  
2898     <msg id="DEFEND_LOCK_RUNNING">TPM is defending against dictionary attack  
2899     <msg id="NO_MSG"></msg>  
2900     <!-- End TPM failure codes -->  
2901     </msg_list>  
2902 </specification>
```

new/usr/src/lib/libc/Makefile.targ

1

```
*****
7204 Thu Jun 30 22:02:06 2016
new/usr/src/lib/libc/Makefile.targ
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
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 #
27 # libc build rules
29 #
30 # This first rule picks up optimized sources for libc variants.
31 #
32 pics%.o: $(OPTIMIZED_LIBCBASE)/gen%.s
33     $(BUILD.s)
34     $(POST_PROCESS_O)
36 pics%.o: $(LIBCBASE)/crt%.c
37     $(COMPILE.c) -o $@ $<
38     $(POST_PROCESS_O)
40 pics%.o: $(LIBCBASE)/crt%.s
41     $(BUILD.s)
42     $(POST_PROCESS_O)
44 pics%.o: $(LIBCBASE)/gen%.c
45     $(COMPILE.c) -o $@ $<
46     $(POST_PROCESS_O)
48 pics%.o: $(LIBCBASE)/gen%.s
49     $(BUILD.s)
50     $(POST_PROCESS_O)
52 pics%.o: $(LIBCBASE)/../$(TARGET_ARCH)/gen%.s
53     $(BUILD.s)
54     $(POST_PROCESS_O)
56 pics/_stack_grow.o: $(LIBCBASE)/../$(TARGET_ARCH)/gen/_stack_grow.s
57     $(BUILD.s) $(LIBCBASE)/../$(TARGET_ARCH)/gen/_stack_grow.s
58     $(POST_PROCESS_O)
```

new/usr/src/lib/libc/Makefile.targ

2

```
60 pics%.o: $(LIBCBASE)/fp%.c
61     $(COMPILE.c) -o $@ $<
62     $(POST_PROCESS_O)
64 pics%.o: $(LIBCBASE)/../$(MACH)/fp%.c
65     $(COMPILE.c) -o $@ $<
66     $(POST_PROCESS_O)
68 pics%.o: $(LIBCBASE)/fp%.s
69     $(BUILD.s)
70     $(POST_PROCESS_O)
72 pics%.o: $(LIBCBASE)/../$(TARGET_ARCH)/fp%.c
73     $(COMPILE.c) -o $@ $<
74     $(POST_PROCESS_O)
76 pics%.o: $(LIBCBASE)/../$(TARGET_ARCH)/fp%.s
77     $(BUILD.s)
78     $(POST_PROCESS_O)
80 pics%.o: $(LIBCBASE)/sys%.c
81     $(COMPILE.c) -o $@ $<
82     $(POST_PROCESS_O)
84 pics%.o: $(LIBCBASE)/../common/sys%.s
85     $(BUILD.s)
86     $(POST_PROCESS_O)
88 pics%.o: $(LIBCBASE)/sys%.s
89     $(BUILD.s)
90     $(POST_PROCESS_O)
92 pics%.o: $(LIBCBASE)/../$(MACH)/sys%.s
93     $(BUILD.s)
94     $(POST_PROCESS_O)
96 pics%.o: $(LIBCBASE)/../$(TARGET_ARCH)/sys%.c
97     $(COMPILE.c) -o $@ $<
98     $(POST_PROCESS_O)
100 pics%.o: $(LIBCBASE)/../$(TARGET_ARCH)/sys%.s
101     $(BUILD.s)
102     $(POST_PROCESS_O)
104 # rules to build large file aware objects (xxx64.o from xxx.s or xxx.c)
106 pics/%64.o: $(LIBCBASE)/../common/sys%.s
107     $(BUILD.s)
108     $(POST_PROCESS_O)
110 pics/%64.o: $(LIBCBASE)/sys%.s
111     $(BUILD.s)
112     $(POST_PROCESS_O)
114 pics/%64.o: $(LIBCBASE)/gen%.c
115     $(COMPILE.c) -o $@ $<
116     $(POST_PROCESS_O)
118 pics/%64.o: $(LIBCBASE)/../port/gen%.c
119     $(COMPILE.c) -o $@ $<
120     $(POST_PROCESS_O)
122 pics/%64.o: $(LIBCBASE)/../port/sys%.c
123     $(COMPILE.c) -o $@ $<
124     $(POST_PROCESS_O)
```

```

126 pics/%64.o: $(LIBCBASE)/../port/print/%.c
127     $(COMPILE.c) -o $@ $<
128     $(POST_PROCESS_O)

130 pics/%64.o: $(LIBCBASE)/../port/regex/%.c
131     $(COMPILE.c) -DM_I18N_MB -DI18N \
132     -I$(LIBCBASE)/../port/regex \
133     -I$(LIBCBASE)/../port/gen -o $@ $<
134     $(POST_PROCESS_O)

136 pics/%64.o: $(LIBCBASE)/../port/stdio/%.c
137     $(COMPILE.c) -o $@ $<
138     $(POST_PROCESS_O)

140 pics/%_w.o: $(LIBCBASE)/../port/stdio/%.c
141     $(COMPILE.c) -o $@ $<
142     $(POST_PROCESS_O)

144 pics/%_w.o: $(LIBCBASE)/../port/print/%.c
145     $(COMPILE.c) -o $@ $<
146     $(POST_PROCESS_O)

148 pics/%_pos.o: $(LIBCBASE)/../port/il8n/%.c
149     $(COMPILE.c) -o $@ $<
150     $(POST_PROCESS_O)

152 pics/%_sbyte.o: $(LIBCBASE)/../port/il8n/%.c
153     $(COMPILE.c) -o $@ $<
154     $(POST_PROCESS_O)

156 pics/%_possbyte.o: $(LIBCBASE)/../port/il8n/%.c
157     $(COMPILE.c) -o $@ $<
158     $(POST_PROCESS_O)

160 pics/%_longlong.o: $(LIBCBASE)/../port/il8n/%.c
161     $(COMPILE.c) -o $@ $<
162     $(POST_PROCESS_O)

164 # libc build rules for objects built from "portable" source in ../port

166 pics/%.o: $(LIBCBASE)/../port/fp/%.c
167     $(COMPILE.c) \
168     -I$(LIBCBASE)/../port/fp -o $@ $<
169     $(POST_PROCESS_O)

171 pics/%.o: $(LIBCBASE)/../port/il8n/%.c
172     $(COMPILE.c) \
173     -I$(LIBCBASE)/../port/il8n -o $@ $<
174     $(POST_PROCESS_O)

176 # gen rules
177 pics/%.o %.o: $(LIBCBASE)/../port/gen/%.c
178     $(COMPILE.c) -o $@ $<
179     $(POST_PROCESS_O)

181 # locale rules
182 pics/%.o %.o: $(LIBCBASE)/../port/locale/%.c
183     $(COMPILE.c) -o $@ $<
184     $(POST_PROCESS_O)

186 # print rules
187 pics/%.o: $(LIBCBASE)/../port/print/%.c
188     $(COMPILE.c) -o $@ $<
189     $(POST_PROCESS_O)

```

```

191 # regex rules
192 pics/%.o: $(LIBCBASE)/../port/regex/%.c
193     $(COMPILE.c) -DM_I18N_MB -DI18N \
194     -I$(LIBCBASE)/../port/regex \
195     -I$(LIBCBASE)/../port/gen -o $@ $<
196     $(POST_PROCESS_O)

198 # stdio rules
199 pics/%.o: $(LIBCBASE)/../port/stdio/%.c
200     $(COMPILE.c) -o $@ $<
201     $(POST_PROCESS_O)

203 # c89 print, stdio rules
204 pics/%_c89.o: $(LIBCBASE)/../port/print/%.c
205     $(COMPILE.c) -o $@ $<
206     $(POST_PROCESS_O)

208 pics/%_c89.o: $(LIBCBASE)/../port/stdio/%.c
209     $(COMPILE.c) -o $@ $<
210     $(POST_PROCESS_O)

212 # aio rules
213 pics/%.o: $(LIBCBASE)/../port/aio/%.c
214     $(COMPILE.c) -o $@ $<
215     $(POST_PROCESS_O)

217 # rt rules
218 pics/%.o: $(LIBCBASE)/../port/rt/%.c
219     $(COMPILE.c) -o $@ $<
220     $(POST_PROCESS_O)

222 # tpool rules
223 pics/%.o: $(LIBCBASE)/../port/tpool/%.c
224     $(COMPILE.c) -o $@ $<
225     $(POST_PROCESS_O)

227 # threads rules
228 pics/%.o: $(LIBCBASE)/../port/threads/%.c
229     $(COMPILE.c) -o $@ $<
230     $(POST_PROCESS_O)

232 pics/%.o: $(LIBCBASE)/threads/%.c
233     $(COMPILE.c) -o $@ $<
234     $(POST_PROCESS_O)

236 pics/%.o: $(LIBCBASE)/threads/%.s
237     $(BUILD.s)
238     $(POST_PROCESS_O)

240 pics/%.o: $(LIBCBASE)/../$(TARGET_ARCH)/threads/%.c
241     $(COMPILE.c) -o $@ $<
242     $(POST_PROCESS_O)

244 pics/%.o: $(LIBCBASE)/../$(TARGET_ARCH)/threads/%.s
245     $(BUILD.s)
246     $(POST_PROCESS_O)

248 pics/%.o: $(LIBCBASE)/../$(TARGET_ARCH)/unwind/%.c
249     $(COMPILE.c) -o $@ $<
250     $(POST_PROCESS_O)

252 pics/%.o: $(LIBCBASE)/../$(TARGET_ARCH)/unwind/%.s
253     $(BUILD.s)
254     $(POST_PROCESS_O)

256 pics/%.o: $(LIBCBASE)/../port/unwind/%.c

```

new/usr/src/lib/libc/Makefile.targ

5

```
257      $(COMPILE.c) -o $@ $<
258      $(POST_PROCESS_O)

260 pics/%.o: $(LIBCBASE)/../$(MACH)/unwind/%.s
261      $(BUILD.s)
262      $(POST_PROCESS_O)

264 pics/%.o: $(LIBCBASE)/../port/sys/%.c
265      $(COMPILE.c) -o $@ $<
266      $(POST_PROCESS_O)

268 pics/%.o: $(LIBCBASE)/../common/common/%.c
269      $(COMPILE.c) -o $@ $<
270      $(POST_PROCESS_O)

272 pics/%.o: $(LIBCBASE)/$(CRTSRCS)/%.s
273      $(BUILD.s)
274      $(POST_PROCESS_O)

276 # $(SRC)/common rules
277 pics/%.o: $(SRC)/common/atomic/$(TARGETMACH)/%.s
278      $(BUILD.s)
279      $(POST_PROCESS_O)

281 $(COMOBSJS:=pics/%): $(SRC)/common/util/$(@F:.o=.c)
282      $(COMPILE.c) -o $@ $(SRC)/common/util/$(@F:.o=.c)
283      $(POST_PROCESS_O)

285 $(XATTROBSJS:=pics/%): $(SRC)/common/xattr/$(@F:.o=.c)
286      $(COMPILE.c) -o $@ $(SRC)/common/xattr/$(@F:.o=.c)
287      $(POST_PROCESS_O)

289 $(DTRACEOBSJS:=pics/%): $(SRC)/common/dtrace/$(@F:.o=.c)
290      $(COMPILE.c) -o $@ $(SRC)/common/dtrace/$(@F:.o=.c)
291      $(POST_PROCESS_O)

293 $(SECFLAGSOBSJS:=pics/%): $(SRC)/common/secflags/$(@F:.o=.c)
294      $(COMPILE.c) -o $@ $(SRC)/common/secflags/$(@F:.o=.c)
295      $(POST_PROCESS_O)

297 #endif /* ! codereview */
298 $(UNICODEOBSJS:=pics/%): $(SRC)/common/unicode/$(@F:.o=.c)
299      $(COMPILE.c) -o $@ $(SRC)/common/unicode/$(@F:.o=.c)
300      $(POST_PROCESS_O)

302 $(CHACHAOBSJS:=pics/%): $(SRC)/common/crypto/chacha/$(@F:.o=.c)
303      $(COMPILE.c) -I$(SRC)/common/crypto/chacha -DKEYSTREAM_ONLY \
304      -o $@ $(SRC)/common/crypto/chacha/$(@F:.o=.c)
305      $(POST_PROCESS_O)

307 # DTrace rules
308 pics/%.o: $(LIBCBASE)/../port/threads/%.d $(THREADSOBSJS:=pics/%)
309      $(COMPILE.d) -C -xlazyload -s $< -o $@ $(THREADSOBSJS:=pics/%)
310      $(POST_PROCESS_O)

312 include $(SRC)/lib/Makefile.targ
```

```

*****
23028 Thu Jun 30 22:02:07 2016
new/usr/src/lib/libc/amd64/Makefile
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
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 (c) 2004, 2010, Oracle and/or its affiliates. All rights reserved.
23 # Copyright 2016 Joyent, Inc.
24 #
25 # Copyright (c) 2013, OmniTI Computer Consulting, Inc. All rights reserved.
26 # Copyright 2013 Garrett D'Amore <garrett@damore.org>
27 # Copyright 2011 Nexenta Systems, Inc. All rights reserved.
28 # Use is subject to license terms.
29 #

31 LIBCBASE= .
32 LIBCDIR= $(SRC)/lib/libc
33 LIBRARY= libc.a
34 LIB_PIC= libc_pic.a
35 VERS= .1
36 CPP= /usr/lib/cpp
37 TARGET_ARCH= amd64

39 # objects are grouped by source directory

41 # local objects
42 STRETS=

44 CRTOBSJ= \
45     cerror.o

47 DYNOBJS=

49 FPOBJS= \
50     fpgetmask.o \
51     fpgetround.o \
52     fpsetmask.o \
53     fpsetround.o \
54     fpstart.o \
55     ieee.o

57 I386FPOBJS= \
58     _D_cplx_div.o \

```

```

59     _D_cplx_div_ix.o \
60     _D_cplx_div_rx.o \
61     _D_cplx_lr_div.o \
62     _D_cplx_lr_div_ix.o \
63     _D_cplx_lr_div_rx.o \
64     _D_cplx_mul.o \
65     _F_cplx_div.o \
66     _F_cplx_div_ix.o \
67     _F_cplx_div_rx.o \
68     _F_cplx_lr_div.o \
69     _F_cplx_lr_div_ix.o \
70     _F_cplx_lr_div_rx.o \
71     _F_cplx_mul.o \
72     _X_cplx_div.o \
73     _X_cplx_div_ix.o \
74     _X_cplx_div_rx.o \
75     _X_cplx_lr_div.o \
76     _X_cplx_lr_div_ix.o \
77     _X_cplx_lr_div_rx.o \
78     _X_cplx_mul.o

80 FPASMOBJS= \
81     __xgetRD.o \
82     _xtoll.o \
83     _xtoull.o \
84     _base_il.o \
85     fpcw.o \
86     fpgetsticky.o \
87     fpsetsticky.o

89 ATOMICOBJS= \
90     atomic.o

92 CHACHAOBJS= \
93     chacha.o

95 KATROBJS= \
96     xattr_common.o
97 COMOBJS= \
98     bcmp.o \
99     bcopy.o \
100    bsearch.o \
101    bzero.o \
102    qsort.o \
103    strtol.o \
104    strtoul.o \
105    strtoll.o \
106    strtoull.o

108 GENOBJS= \
109    _getsp.o \
110    abs.o \
111    alloca.o \
112    arc4random.o \
113    arc4random_uniform.o \
114    attrat.o \
115    byteorder.o \
116    cuexit.o \
117    ecvt.o \
118    endian.o \
119    errlst.o \
120    amd64_data.o \
121    ldivide.o \
122    lock.o \
123    makecvt.o \
124    memccpy.o \

```

new/usr/src/lib/libc/amd64/Makefile

```

125     memchr.o      \
126     memcmp.o     \
127     memcpy.o     \
128     memset.o     \
129     new_list.o   \
130     proc64_id.o  \
131     proc64_support.o \
132     setjmp.o     \
133     siginfolst.o \
134     siglongjmp.o \
135     strcmp.o     \
136     strcpy.o     \
137     strlen.o     \
138     strncmp.o    \
139     strncpy.o    \
140     strnlen.o    \
141     sync_instruction_memory.o

```

```

143 # Preserved solely to ease maintenance of 32-bit and 64-bit library builds
144 # This macro should ALWAYS be empty; native APIs are already 'large file'.
145 COMSYSOBS64=

```

```
147 SYSOBS64=
```

```

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

```

3

new/usr/src/lib/libc/amd64/Makefile

```

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

```

4


```

257     setreid.o           \
258     setrlimit.o        \
259     setuid.o           \
260     sigaltstk.o        \
261     sigprocmsk.o       \
262     sigsendset.o       \
263     sigsuspend.o       \
264     statfs.o           \
265     statvfs.o          \
266     stty.o             \
267     sync.o             \
268     sysconfig.o        \
269     sysfs.o            \
270     sysinfo.o          \
271     syslwp.o           \
272     times.o            \
273     ulimit.o           \
274     umask.o            \
275     umount2.o          \
276     utssys.o           \
277     uucopy.o           \
278     vhangup.o          \
279     waitid.o           \
280     write.o            \
281     writev.o           \
282     yield.o            \
\
284 SYSOBJS=              \
285     __clock_gettime.o  \
286     __getcontext.o    \
287     __uadmin.o        \
288     __lwp_mutex_unlock.o \
289     __stack_grow.o    \
290     door.o            \
291     forkx.o           \
292     forkallx.o        \
293     getcontext.o      \
294     gettimeofday.o    \
295     lwp_private.o     \
296     nuname.o          \
297     syscall.o         \
298     sysi86.o          \
299     tls_get_addr.o    \
300     uadmin.o          \
301     umount.o          \
302     uname.o           \
303     vforkx.o          \
\
305 # Preserved solely to ease maintenance of 32-bit and 64-bit library builds
306 # This macro should ALWAYS be empty; native APIs are already 'large file'.
307 PORTGEN64=
\
309 # objects from source under $(LIBCDIR)/port
310 PORTFP=                \
311     __flt_decim.o      \
312     __flt_rounds.o    \
313     __tbl_10_b.o       \
314     __tbl_10_h.o       \
315     __tbl_10_s.o       \
316     __tbl_2_b.o        \
317     __tbl_2_h.o        \
318     __tbl_2_s.o        \
319     __tbl_fdq.o        \
320     __tbl_tens.o       \
321     __x_power.o        \
322     __base_sup.o       \

```

```

323     aconvert.o         \
324     decimal_bin.o     \
325     double_decim.o    \
326     econvert.o        \
327     fconvert.o        \
328     file_decim.o      \
329     finite.o          \
330     fp_data.o         \
331     func_decim.o      \
332     gconvert.o        \
333     hex_bin.o         \
334     ieee_globals.o    \
335     pack_float.o      \
336     sigfpe.o          \
337     string_decim.o    \
\
339 PORTGEN=              \
340     _env_data.o       \
341     _xftw.o           \
342     a64l.o            \
343     abort.o           \
344     addsev.o          \
345     ascii_strcasecmp.o \
346     ascii_strncasecmp.o \
347     assert.o          \
348     atof.o            \
349     atoi.o            \
350     atol.o            \
351     atoll.o           \
352     attropen.o        \
353     atexit.o          \
354     atfork.o          \
355     basename.o        \
356     calloc.o          \
357     catgets.o         \
358     catopen.o         \
359     cfgetispeed.o     \
360     cfgetospeed.o    \
361     cfree.o           \
362     cfsetispeed.o     \
363     cfsetospeed.o    \
364     cftime.o          \
365     clock.o           \
366     closedir.o        \
367     closefrom.o       \
368     confstr.o         \
369     crypt.o           \
370     csetlen.o         \
371     ctime.o           \
372     ctime_r.o         \
373     daemon.o          \
374     deflt.o           \
375     directio.o        \
376     dirname.o         \
377     div.o             \
378     drand48.o         \
379     dup.o             \
380     env_data.o        \
381     err.o             \
382     errno.o           \
383     euclen.o          \
384     event_port.o      \
385     execvp.o          \
386     explicit_bzero.o  \
387     fattach.o         \
388     fdetach.o         \

```

```

389 fdopendir.o \
390  ffs.o \
391  flock.o \
392  fls.o \
393  fmtmsg.o \
394  ftime.o \
395  ftok.o \
396  ftw.o \
397  gcvt.o \
398  getauxv.o \
399  getcwd.o \
400  getdate_err.o \
401  getdtblsize.o \
402  getentropy.o \
403  getenv.o \
404  getexecname.o \
405  getgrnam.o \
406  getgrnam_r.o \
407  gethostid.o \
408  gethostname.o \
409  gethz.o \
410  getisax.o \
411  getloadavg.o \
412  getlogin.o \
413  getmntent.o \
414  getnetgrent.o \
415  get_nprocs.o \
416  getopt.o \
417  getopt_long.o \
418  getpagesize.o \
419  getpw.o \
420  getpwnam.o \
421  getpwnam_r.o \
422  getrusage.o \
423  getspent.o \
424  getspent_r.o \
425  getsubopt.o \
426  gettxt.o \
427  getusershell.o \
428  getut.o \
429  getutx.o \
430  getvfsent.o \
431  getwd.o \
432  getwidth.o \
433  getxby_door.o \
434  gtxt.o \
435  hsearch.o \
436  iconv.o \
437  imaxabs.o \
438  imaxdiv.o \
439  index.o \
440  initgroups.o \
441  insque.o \
442  isaexec.o \
443  isastream.o \
444  isatty.o \
445  killpg.o \
446  klpdlib.o \
447  l64a.o \
448  lckpwwdf.o \
449  lconstants.o \
450  lexp10.o \
451  lfind.o \
452  lfnt.o \
453  lfnt_log.o \
454  lldiv.o \

```

```

455  llog10.o \
456  lltostr.o \
457  lmath.o \
458  localtime.o \
459  lsearch.o \
460  madvise.o \
461  malloc.o \
462  memalign.o \
463  memmem.o \
464  mkdev.o \
465  mkdtemp.o \
466  mkfifo.o \
467  mkstemp.o \
468  mktemp.o \
469  mlock.o \
470  mlockall.o \
471  mon.o \
472  msync.o \
473  munlock.o \
474  munlockall.o \
475  ndbm.o \
476  nftw.o \
477  nlspath_checks.o \
478  nsparse.o \
479  nss_common.o \
480  nss_dbdefs.o \
481  nss_deffinder.o \
482  opendir.o \
483  opt_data.o \
484  perror.o \
485  pfnt.o \
486  pfnt_data.o \
487  pfnt_print.o \
488  pipe.o \
489  plock.o \
490  poll.o \
491  posix_fadvise.o \
492  posix_fallocate.o \
493  posix_madvise.o \
494  posix_memalign.o \
495  priocntl.o \
496  privlib.o \
497  priv_str_xlate.o \
498  psecflags.o \
499  #endif /* ! codereview */
500  psiginfo.o \
501  psignal.o \
502  pt.o \
503  putpwent.o \
504  putspent.o \
505  raise.o \
506  rand.o \
507  random.o \
508  rctlops.o \
509  readdir.o \
510  readdir_r.o \
511  realpath.o \
512  reboot.o \
513  regexpr.o \
514  remove.o \
515  rewinddir.o \
516  rindex.o \
517  scandir.o \
518  seekdir.o \
519  select.o \
520  setlabel.o \

```

```

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

```

```

587      wdata.o \
588      xgetwidth.o \
589      xpg4.o \
590      xpg6.o \
592  PORTPRINT_W= \
593      doprnt_w.o \
595  PORTPRINT= \
596      asprintf.o \
597      doprnt.o \
598      fprintf.o \
599      printf.o \
600      snprintf.o \
601      sprintf.o \
602      vfprintf.o \
603      vprintf.o \
604      vsnprintf.o \
605      vsprintf.o \
606      vwprintf.o \
607      wprintf.o \
609  # Preserved solely to ease maintenance of 32-bit and 64-bit library builds
610  # This macro should ALWAYS be empty; native APIs are already 'large file'.
611  PORTSTDIO64= \
613  PORTSTDIO_W= \
614      doscan_w.o \
616  PORTSTDIO= \
617      __extensions.o \
618      __endopen.o \
619      __filbuf.o \
620      __findbuf.o \
621      __flsbuf.o \
622      __wrtchk.o \
623      clearerr.o \
624      ctermid.o \
625      ctermid_r.o \
626      cuserid.o \
627      data.o \
628      doscan.o \
629      fdopen.o \
630      feof.o \
631      ferrror.o \
632      fgetc.o \
633      fgets.o \
634      fileno.o \
635      flockf.o \
636      flush.o \
637      fopen.o \
638      fpos.o \
639      fputc.o \
640      fputs.o \
641      fread.o \
642      fseek.o \
643      fseeko.o \
644      ftell.o \
645      ftello.o \
646      fwrite.o \
647      getc.o \
648      getchar.o \
649      getline.o \
650      getpass.o \
651      gets.o \
652      getw.o \

```

```

653     mse.o \
654     popen.o \
655     putc.o \
656     putchar.o \
657     puts.o \
658     putw.o \
659     rewind.o \
660     scanf.o \
661     setbuf.o \
662     setbuffer.o \
663     setvbuf.o \
664     system.o \
665     tempnam.o \
666     tmpfile.o \
667     tmpnam_r.o \
668     ungetc.o \
669     vscanf.o \
670     wscanf.o \
671     wscanf.o \

673 PORTI18N= \
674     getwchar.o \
675     putwchar.o \
676     putws.o \
677     strtows.o \
678     wcsnlen.o \
679     wcsstr.o \
680     wcstoimax.o \
681     wcstol.o \
682     wcstoul.o \
683     wcswcs.o \
684     wmemchr.o \
685     wmemcmp.o \
686     wmemcpy.o \
687     wmemmove.o \
688     wmemset.o \
689     wscat.o \
690     wschr.o \
691     wscmp.o \
692     wscpy.o \
693     wscspn.o \
694     wsdup.o \
695     wslen.o \
696     wsncat.o \
697     wsncmp.o \
698     wsncpy.o \
699     wspbkr.o \
700     wsprintf.o \
701     wsrchr.o \
702     wsscanf.o \
703     wsspno.o \
704     wstod.o \
705     wstok.o \
706     wstol.o \
707     wstoll.o \
708     wsxfrm.o \
709     gettext.o \
710     gettext_gnu.o \
711     gettext_real.o \
712     gettext_util.o \
713     plural_parser.o \
714     wdresolve.o \
715     _ctype.o \
716     isascii.o \
717     toascii.o \

```

```

719 PORTI18N_COND= \
720     wcstol_longlong.o \
721     wcstoul_longlong.o \

723 PORTLOCALE= \
724     big5.o \
725     btowc.o \
726     collate.o \
727     collcmp.o \
728     euc.o \
729     fnmatch.o \
730     fgetwc.o \
731     fgetws.o \
732     fix_grouping.o \
733     fputwc.o \
734     fputws.o \
735     fwide.o \
736     gb18030.o \
737     gb2312.o \
738     gbk.o \
739     getdate.o \
740     isdigit.o \
741     iswctype.o \
742     ldapart.o \
743     lmessages.o \
744     lnumeric.o \
745     lmonetary.o \
746     localeconv.o \
747     localeimpl.o \
748     mbftowc.o \
749     mblen.o \
750     mbrlen.o \
751     mbrtowc.o \
752     mbsinit.o \
753     mbsnrtowcs.o \
754     mbsrtowcs.o \
755     mbstowcs.o \
756     mbtowc.o \
757     mskanji.o \
758     nextwctype.o \
759     nl_langinfo.o \
760     none.o \
761     regcomp.o \
762     regfree.o \
763     regerror.o \
764     regexec.o \
765     rune.o \
766     runetype.o \
767     setlocale.o \
768     setrunelocale.o \
769     strcasecmp.o \
770     strcasestr.o \
771     strcoll.o \
772     strfmon.o \
773     strftime.o \
774     strncasecmp.o \
775     strptime.o \
776     strxfrm.o \
777     table.o \
778     timelocal.o \
779     tolower.o \
780     towlower.o \
781     ungetwc.o \
782     utf8.o \
783     wcrtoomb.o \
784     wcscasecmp.o \

```

```

785     wscoll.o      \
786     wcsftime.o   \
787     wcsnrtoombs.o \
788     wcsrtombs.o  \
789     wcswidth.o   \
790     wcstombs.o   \
791     wcsxfrm.o    \
792     wctob.o      \
793     wctomb.o     \
794     wctrans.o    \
795     wctype.o     \
796     wcwidth.o   \
797     wscol.o      \

799 AIOBJS=      \
800     aio.o      \
801     aio_alloc.o \
802     posix_aio.o \

804 RTOBJS=      \
805     clock_timer.o \
806     mqueue.o      \
807     posixobj.o   \
808     sched.o       \
809     sem.o         \
810     shm.o         \
811     sigev_thread.o \

813 SECFLAGSOBJS= \
814     secflags.o  \

816 #endif /* ! codereview */
817 TPOOLOBJS=      \
818     thread_pool.o \

820 THREADSOBJS=   \
821     alloc.o      \
822     assfail.o    \
823     cll_thr.o    \
824     cancel.o     \
825     door_calls.o \
826     tmem.o       \
827     pthr_attr.o  \
828     pthr_barrier.o \
829     pthr_cond.o  \
830     pthr_mutex.o \
831     pthr_rwlock.o \
832     pthread.o    \
833     rwlock.o     \
834     scalls.o     \
835     sema.o       \
836     sigaction.o  \
837     spawn.o      \
838     synch.o      \
839     tdb_agent.o  \
840     thr.o        \
841     thread_interface.o \
842     tls.o        \
843     tsd.o        \

845 THREADSMACHOBJS= \
846     machdep.o    \

848 THREADSASMOBJS= \
849     asm_subr.o   \

```

```

851 UNICODEOBJS=   \
852     u8_textprep.o \
853     uconv.o       \

855 UNWINDMACHOBJS= \
856     call_frame_inst.o \
857     eh_frame.o       \
858     thrp_unwind.o   \
859     unwind.o        \

861 pics/unwind.o:= COPTFLAG64 =

863 UNWINDASMOBJS= \
864     unwind_frame.o \

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

870 PORTSYS=        \
871     _autofssys.o \
872     access.o     \
873     acctctl.o    \
874     bsd_signal.o \
875     chmod.o      \
876     chown.o      \
877     corectl.o    \
878     epoll.o      \
879     exaccts.o    \
880     execl.o      \
881     execle.o     \
882     execv.o      \
883     eventfd.o    \
884     fcntl.o      \
885     getpagesizes.o \
886     getpeerucred.o \
887     inst_sync.o  \
888     issetugid.o  \
889     label.o      \
890     link.o       \
891     lockf.o      \
892     lwp.o        \
893     lwp_cond.o   \
894     lwp_rwlock.o \
895     lwp_sigmask.o \
896     meminfosys.o \
897     mkdir.o      \
898     mknod.o      \
899     msgsys.o     \
900     nfssys.o     \
901     open.o       \
902     pgrp.o       \
903     posix_sigwait.o \
904     ppriv.o      \
905     psetsys.o    \
906     rctlsys.o    \
907     readlink.o   \
908     rename.o     \
909     sbrk.o       \
910     semsys.o     \
911     set_errno.o  \
912     sharefs.o    \
913     shmsys.o     \
914     sidsys.o     \
915     siginterrupt.o \
916     signal.o     \

```

```

917     signalfd.o \
918     sigpending.o \
919     sigstack.o \
920     stat.o \
921     symlink.o \
922     tasksys.o \
923     time.o \
924     time_util.o \
925     timerfd.o \
926     ucontext.o \
927     unlink.o \
928     ustat.o \
929     utimesys.o \
930     zone.o

932 PORTREGEX= \
933     glob.o \
934     regcmp.o \
935     regex.o \
936     wordexp.o

938 VALUES= \
939     values-Xa.o

941 MOSTOBSJ= \
942     $(STRETS) \
943     $(CRTOBSJ) \
944     $(DYNOBJS) \
945     $(FPOBJS) \
946     $(I386FPOBJS) \
947     $(FPASMOBJS) \
948     $(ATOMICOBJS) \
949     $(CHACHAOBJS) \
950     $(XATTROBJS) \
951     $(COMOBJS) \
952     $(GENOBJS) \
953     $(PORTFP) \
954     $(PORTGEN) \
955     $(PORTGEN64) \
956     $(PORTI18N) \
957     $(PORTI18N_COND) \
958     $(PORTLOCALE) \
959     $(PORTPRINT) \
960     $(PORTPRINT_W) \
961     $(PORTREGEX) \
962     $(PORTSTDIO) \
963     $(PORTSTDIO64) \
964     $(PORTSTDIO_W) \
965     $(PORTSYS) \
966     $(PORTSYS64) \
967     $(AIOOBJS) \
968     $(RTOBJS) \
969     $(SECFLAGSOBJS) \
970 #endif /* ! codereview */
971     $(TPOOLBJS) \
972     $(THREADSOBJS) \
973     $(THREADSMACHOBJS) \
974     $(THREADSASMOBJS) \
975     $(UNICODEOBJS) \
976     $(UNWINDMACHOBJS) \
977     $(UNWINDASMOBJS) \
978     $(COMSYSOBJS) \
979     $(SYSOBJS) \
980     $(COMSYSOBJS64) \
981     $(SYSOBJS64) \
982     $(VALUES)

```

```

984 TRACEOBSJ= \
985     plockstat.o

987 # NOTE: libc.so.1 must be linked with the minimal crt1.o and crtn.o
988 # modules whose source is provided in the $(SRC)/lib/common directory.
989 # This must be done because otherwise the Sun C compiler would insert
990 # its own versions of these modules and those versions contain code
991 # to call out to C++ initialization functions. Such C++ initialization
992 # functions can call back into libc before thread initialization is
993 # complete and this leads to segmentation violations and other problems.
994 # Since libc contains no C++ code, linking with the minimal crt1.o and
995 # crtn.o modules is safe and avoids the problems described above.
996 OBJECTS= $(CRTI) $(MOSTOBSJ) $(CRTN)
997 CRTSRCS= ../../common/amd64

999 # include common library definitions
1000 include ../../Makefile.lib
1001 include ../../Makefile.lib.64

1003 CFLAGS64 += $(CTF_FLAGS)

1005 # This is necessary to avoid problems with calling _ex_unwind().
1006 # We probably don't want any inlining anyway.
1007 CFLAGS64 += -xinline=

1009 CERRWARN += -_gcc=-Wno-parentheses
1010 CERRWARN += -_gcc=-Wno-switch
1011 CERRWARN += -_gcc=-Wno-uninitialized
1012 CERRWARN += -_gcc=-Wno-unused-value
1013 CERRWARN += -_gcc=-Wno-unused-label
1014 CERRWARN += -_gcc=-Wno-unused-variable
1015 CERRWARN += -_gcc=-Wno-type-limits
1016 CERRWARN += -_gcc=-Wno-char-subscripts
1017 CERRWARN += -_gcc=-Wno-clobbered
1018 CERRWARN += -_gcc=-Wno-unused-function
1019 CERRWARN += -_gcc=-Wno-address

1021 # Setting THREAD_DEBUG = -DTHREAD_DEBUG (make THREAD_DEBUG=-DTHREAD_DEBUG ...)
1022 # enables ASSERT() checking in the threads portion of the library.
1023 # This is automatically enabled for DEBUG builds, not for non-debug builds.
1024 THREAD_DEBUG =
1025 $(NOT_RELEASE_BUILD)THREAD_DEBUG = -DTHREAD_DEBUG

1027 # Make string literals read-only to save memory
1028 CFLAGS64 += $(XSTRCONST)

1030 ALTPICS= $(TRACEOBSJ:%=pics/%)

1032 $(DYNLIB) := BUILD.SO = $(LD) -o $@ -G $(DYNFLAGS) $(PICS) $(ALTPICS) $(EXTPICS)

1034 MAPFILES = $(LIBCDIR)/port/mapfile-vers

1036 CPPFLAGS= -D_REENTRANT -D$(MACH64) -D__$(MACH64) $(THREAD_DEBUG) \
1037 -I. -I$(LIBCBASE)/inc -I$(LIBCDIR)/inc $(CPPFLAGS.master)
1038 ASFLAGS= $(AS_PICFLAGS) -P -D__STDC__ -D_ASM $(CPPFLAGS) \
1039 $(amd64_AS_XARCH)

1041 # As a favor to the dtrace syscall provider, libc still calls the
1042 # old syscall traps that have been obsoleted by the *at() interfaces.
1043 # Delete this to compile libc using only the new *at() system call traps
1044 CPPFLAGS += -D_RETAIN_OLD_SYSCALLS

1046 # proc64_id.o is built with defines in $(SRC)/uts/intel/sys/x86_archext.h
1047 pics/proc64_id.o := CFLAGS64 += -I$(SRC)/uts/intel

```

```

1049 # Inform the run-time linker about libc specialized initialization
1050 RTLDINFO =      -z rtldinfo=tls_rtldinfo
1051 DYNFLAGS +=     $(RTLDINFO)

1053 # Force libc's internal references to be resolved immediately upon loading
1054 # in order to avoid critical region problems. Since almost all libc symbols
1055 # are marked 'protected' in the mapfiles, this is a minimal set (15 to 20).
1056 DYNFLAGS +=     -znnow

1058 BUILD.s=       $(AS) $(ASFLAGS) $< -o $@

1060 # Override this top level flag so the compiler builds in its native
1061 # C99 mode. This has been enabled to support the complex arithmetic
1062 # added to libc.
1063 C99MODE=       $(C99_ENABLE)

1065 # libc method of building an archive
1066 # The "$(GREP) -v ' L '" part is necessary only until
1067 # lorder is fixed to ignore thread-local variables.
1068 BUILD.AR=      $(RM) $@ ; \
1069               $(AR) q $@ `$(LORDER) $(MOSTOBSJ:%=$(DIR)/%) | $(GREP) -v ' L ' | $(TSOR

1071 # extra files for the clean target
1072 CLEANFILES=    \
1073               $(LIBCDIR)/port/gen/errlst.c \
1074               $(LIBCDIR)/port/gen/new_list.c \
1075               assym.h \
1076               genassym \
1077               crt/_rtld.s \
1078               pics/crti.o \
1079               pics/crtn.o \
1080               $(ALTPICS)

1082 CLOBBERFILES += $(LIB_PIC)

1084 # list of C source for lint
1085 SRCS=          \
1086               $(ATOMICOBJS:%.o=$(SRC)/common/atomic/%.c) \
1087               $(XATTROBJS:%.o=$(SRC)/common/xattr/%.c) \
1088               $(COMOBJS:%.o=$(SRC)/common/util/%.c) \
1089               $(PORTFP:%.o=$(LIBCDIR)/port/fp/%.c) \
1090               $(PORTGEN:%.o=$(LIBCDIR)/port/gen/%.c) \
1091               $(PORTI18N:%.o=$(LIBCDIR)/port/il8n/%.c) \
1092               $(PORTLOCALE:%.o=$(LIBCDIR)/port/locale/%.c) \
1093               $(PORTPRINT:%.o=$(LIBCDIR)/port/print/%.c) \
1094               $(PORTREGEX:%.o=$(LIBCDIR)/port/regex/%.c) \
1095               $(PORTSTDIO:%.o=$(LIBCDIR)/port/stdio/%.c) \
1096               $(PORTSYS:%.o=$(LIBCDIR)/port/sys/%.c) \
1097               $(AIOOBJS:%.o=$(LIBCDIR)/port/aio/%.c) \
1098               $(RTOBJS:%.o=$(LIBCDIR)/port/rt/%.c) \
1099               $(SECFLAGSOBJS:%.o=$(SRC)/common/secflags/%.c) \
1100 #endif /* ! codereview */
1101               $(TPOOLBJS:%.o=$(LIBCDIR)/port/tpool/%.c) \
1102               $(THREADSOBJS:%.o=$(LIBCDIR)/port/threads/%.c) \
1103               $(THREADSMACHOBJS:%.o=threads/%.c) \
1104               $(UNICODEOBJS:%.o=$(SRC)/common/unicode/%.c) \
1105               $(UNWINDMACHOBJS:%.o=unwind/%.c) \
1106               $(FPOBJS:%.o=fp/%.c) \
1107               $(I386FPOBJS:%.o=$(LIBCDIR)/i386/fp/%.c) \
1108               $(LIBCBASE)/gen/ecvt.c \
1109               $(LIBCBASE)/gen/makectxt.c \
1110               $(LIBCBASE)/gen/signfolst.c \
1111               $(LIBCBASE)/gen/siglongjmp.c \
1112               $(LIBCBASE)/gen/sync_instruction_memory.c \
1113               $(LIBCBASE)/sys/uadmin.c

```

```

1115 # conditional assignments
1116 # $(DYNLIB) $(LIB_PIC) := DYNOBJS = _rtbootld.o
1117 $(DYNLIB) := CRTI = crti.o
1118 $(DYNLIB) := CRTN = crtn.o

1120 # Files which need the threads .il inline template
1121 TIL=          \
1122               aio.o \
1123               alloc.o \
1124               assfail.o \
1125               atexit.o \
1126               atfork.o \
1127               cancel.o \
1128               door_calls.o \
1129               err.o \
1130               errno.o \
1131               lwp.o \
1132               ma.o \
1133               machdep.o \
1134               posix_aio.o \
1135               pthr_attr.o \
1136               pthr_barrier.o \
1137               pthr_cond.o \
1138               pthr_mutex.o \
1139               pthr_rwlock.o \
1140               pthread.o \
1141               rand.o \
1142               rwlock.o \
1143               scalls.o \
1144               sched.o \
1145               sema.o \
1146               sigaction.o \
1147               sigev_thread.o \
1148               spawn.o \
1149               stack.o \
1150               synchron.o \
1151               tdb_agent.o \
1152               thr.o \
1153               thread_interface.o \
1154               thread_pool.o \
1155               thrp_unwind.o \
1156               tls.o \
1157               tmem.o \
1158               tsd.o

1160 $(TIL:%=pics/%) := CFLAGS64 += $(LIBCBASE)/threads/amd64.il

1162 # pics/mul64.o := CFLAGS64 += crt/mul64.il

1164 # large-file-aware components that should be built large

1166 #$(COMSYSOBJS64:%=pics/%) := \
1167 # CPPFLAGS += -D_LARGEFILE_SOURCE -D_FILE_OFFSET_BITS=64

1169 #$(SYSOBJS64:%=pics/%) := \
1170 # CPPFLAGS += -D_LARGEFILE_SOURCE -D_FILE_OFFSET_BITS=64

1172 #$(PORTGEN64:%=pics/%) := \
1173 # CPPFLAGS += -D_LARGEFILE_SOURCE -D_FILE_OFFSET_BITS=64

1175 #$(PORTSTDIO64:%=pics/%) := \
1176 # CPPFLAGS += -D_LARGEFILE_SOURCE -D_FILE_OFFSET_BITS=64

1178 #$(PORTSYS64:%=pics/%) := \
1179 # CPPFLAGS += -D_LARGEFILE_SOURCE -D_FILE_OFFSET_BITS=64

```

```

1181 $(PORTSTDIO_W:%=pics/%) := \
1182     CPPFLAGS += -D_WIDE

1184 $(PORTPRINT_W:%=pics/%) := \
1185     CPPFLAGS += -D_WIDE

1187 $(PORTPRINT_C89:%=pics/%) := \
1188     CPPFLAGS += -D_C89_INTMAX32

1190 $(PORTSTDIO_C89:%=pics/%) := \
1191     CPPFLAGS += -D_C89_INTMAX32

1193 $(PORTI18N_COND:%=pics/%) := \
1194     CPPFLAGS += -D_WCS_LONGLONG

1196 pics/arc4random.o := CPPFLAGS += -I$(SRC)/common/crypto/chacha

1198 .KEEP_STATE:

1200 all: $(LIBS) $(LIB_PIC)

1202 lint := CPPFLAGS += -I$(LIBCDIR)/$(MACH)/fp
1203 lint := CPPFLAGS += -D_MSE_INT_H -D_LCONV_C99
1204 lint := LINTFLAGS64 += -mn -erroff=E_SUPPRESSION_DIRECTIVE_UNUSED

1206 lint:
1207     @echo $(LINT.c) ... $(LDLIBS)
1208     @$$(LINT.c) $(SRCS) $(LDLIBS)

1210 $(LINTLIB) := SRCS=$(LIBCDIR)/port/llib-1c
1211 $(LINTLIB) := CPPFLAGS += -D_MSE_INT_H
1212 $(LINTLIB) := LINTFLAGS64=-nvx -m64

1214 # object files that depend on inline template
1215 $(TIL:%=pics/%) : $(LIBCBASE)/threads/amd64.il
1216 # pics/mul64.o: crt/mul64.il

1218 # include common libc targets
1219 include ../Makefile.targ

1221 # We need to strip out all CTF data from the static library
1222 $(LIB_PIC) := DIR = pics
1223 $(LIB_PIC): pics $$$(PICS)
1224     $(BUILD.AR)
1225     $(MCS) -d -n .SUNW_ctf $$@ > /dev/null 2>&1
1226     $(AR) -ts $$@ > /dev/null
1227     $(POST_PROCESS_A)

1229 ASSYMDEP_OBJS= \
1230     _lwp_mutex_unlock.o \
1231     _stack_grow.o \
1232     asm_subr.o \
1233     getcontext.o \
1234     setjmp.o \
1235     tls_get_addr.o \
1236     vforkx.o

1238 $(ASSYMDEP_OBJS:%=pics/%) : assym.h

1240 # assym.h build rules

1242 GENASSYM_C = genassym.c

1244 genassym: $(GENASSYM_C)
1245     $(NATIVECC) $(NATIVE_CFLAGS) -Iinc -I$(LIBCDIR)/inc $(CPPFLAGS.native) \
1246     -o $$@ $(GENASSYM_C)

```

```

1248 OFFSETS = $(LIBCDIR)/$(MACH)/offsets.in

1250 assym.h: $(OFFSETS) genassym
1251     $(OFFSETS_CREATE) <$(OFFSETS) >$$@
1252     ./genassym >>$$@

1254 # derived C source and related explicit dependencies
1255 $(LIBCDIR)/port/gen/errlst.c + \
1256 $(LIBCDIR)/port/gen/new_list.c: $(LIBCDIR)/port/gen/errlist $(LIBCDIR)/port/gen/
1257     cd $(LIBCDIR)/port/gen; pwd; $(AWK) -f errlist.awk < errlist

1259 pics/errlst.o: $(LIBCDIR)/port/gen/errlst.c

1261 pics/new_list.o: $(LIBCDIR)/port/gen/new_list.c

```


new/usr/src/lib/libc/common/sys/brk.s

1

1327 Thu Jun 30 22:02:08 2016

new/usr/src/lib/libc/common/sys/brk.s

7029 want per-process exploit mitigation features (secflags)

7030 want basic address space layout randomization (aslr)

7031 noexec_user_stack should be a secflag

7032 want a means to forbid mappings around NULL.

```
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 2004 Sun Microsystems, Inc. All rights reserved.
23 * Use is subject to license terms.
24 */

26     .file    "brk.s"

28 #include "SYS.h"

30 /*
31 * _brk_unlocked() simply traps into the kernel to set the brk. It
32 * returns 0 if the break was successfully set, or -1 otherwise.
33 * It doesn't enforce any alignment and it doesn't perform any locking.
34 * _brk_unlocked() is only called from brk() and _sbrk_unlocked().
35 */

37     ENTRY_NP(_brk_unlocked)
38     SYSTRAP_RVAL1(brk)
39     SYSCERROR
40     RET
40     RETC
41     SET_SIZE(_brk_unlocked)
_____unchanged_portion_omitted_____
```

new/usr/src/lib/libc/common/sys/psecflagsset.s

1

624 Thu Jun 30 22:02:09 2016

new/usr/src/lib/libc/common/sys/psecflagsset.s

7029 want per-process exploit mitigation features (secflags)

7030 want basic address space layout randomization (aslr)

7031 noexec_user_stack should be a secflag

7032 want a means to forbid mappings around NULL.

```
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 /* Copyright 2015, Richard Lowe */
```

```
14 .file "psecflagsset.s"
```

```
16 #include <sys/asm_linkage.h>
```

```
17 #include "SYS.h"
```

```
18
```

```
19 SYSCALL2_RVAL1(__psecflagsset,psecflags)
```

```
20 RET
```

```
21 SET_SIZE(__psecflagsset)
```

```
22 #endif /* ! codereview */
```

```

*****
24677 Thu Jun 30 22:02:09 2016
new/usr/src/lib/libc/i386/Makefile.com
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
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 (c) 2004, 2010, Oracle and/or its affiliates. All rights reserved.
23 # Copyright 2016 Joyent, Inc.
24 # Copyright (c) 2013, OmniTI Computer Consulting, Inc. All rights reserved.
25 # Copyright 2013 Garrett D'Amore <garrett@damore.org>
26 #
27 # Copyright 2011 Nexenta Systems, Inc. All rights reserved.
28 # Use is subject to license terms.
29 #

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

37 VALUES=     values-Xa.o

39 # objects are grouped by source directory

41 # local objects
42 STRETS=

44 CRTOBS=      \
45      cerror.o \
46      cerror64.o

48 DYNOBJS=     \
49      _rtbootld.o

51 FPOBJS=     \
52      _D_cplx_div.o \
53      _D_cplx_div_ix.o \
54      _D_cplx_div_rx.o \
55      _D_cplx_lr_div.o \
56      _D_cplx_lr_div_ix.o \
57      _D_cplx_lr_div_rx.o \
58      _D_cplx_mul.o \

```

```

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

82 FPASMOBJS=  \
83      __xgetRD.o \
84      _base_il.o \
85      _xtoll.o \
86      _xtoull.o \
87      fpcw.o

89 ATOMICOBJS= \
90      atomic.o

92 CHACHAOBJS= \
93      chacha.o

95 KATROBJS=   \
96      xattr_common.o

98 COMOBJS=    \
99      bcmp.o \
100     bcopy.o \
101     bsearch.o \
102     bzero.o \
103     qsort.o \
104     strtol.o \
105     strtoul.o \
106     strtoll.o \
107     strtoull.o

109 DTRACEOBJS= \
110     dtrace_data.o

112 SECFLAGSOBJS= \
113     secflags.o

115 #endif /* ! codereview */
116 GENOBJS=     \
117     _div64.o \
118     _divdi3.o \
119     _getsp.o \
120     _mul64.o \
121     abs.o \
122     alloca.o \
123     arc4random.o \
124     arc4random_uniform.o \

```

```

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

161 # sysobjs that contain large-file interfaces
162 COMSYSOBS64=
163     fstatvfs64.o    \
164     getdents64.o   \
165     getrlimit64.o  \
166     lseek64.o      \
167     mmap64.o       \
168     pread64.o      \
169     preadv64.o     \
170     pwrite64.o     \
171     pwritev64.o    \
172     setrlimit64.o  \
173     statvfs64.o

175 SYSOBS64=

177 COMSYSOBS=
178     __clock_timer.o \
179     __getloadavg.o  \
180     __rusagesys.o   \
181     __signotify.o   \
182     __sigrt.o       \
183     __time.o        \
184     _lgrp_home_fast.o \
185     _lgrpsys.o      \
186     _nfssys.o       \
187     _portfs.o       \
188     _pset.o         \
189     _rpcsys.o       \
190     _sigaction.o

```

```

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

```

```

257 nice.o \
258 ntp_adjtime.o \
259 ntp_gettime.o \
260 p_online.o \
261 pathconf.o \
262 pause.o \
263 pcsample.o \
264 pipe2.o \
265 pollsys.o \
266 pread.o \
267 preadv.o \
268 priocntlset.o \
269 processor_bind.o \
270 processor_info.o \
271 profil.o \
272 psecflagsset.o \
273 #endif /* ! codereview */
274 putmsg.o \
275 putpmsg.o \
276 pwrite.o \
277 pwritev.o \
278 read.o \
279 readv.o \
280 resolvepath.o \
281 seteguid.o \
282 setgid.o \
283 setgroups.o \
284 setitimer.o \
285 setreid.o \
286 setrlimit.o \
287 setuid.o \
288 sigaltstk.o \
289 sigprocmsk.o \
290 sigsendset.o \
291 sigsuspend.o \
292 statfs.o \
293 statvfs.o \
294 stty.o \
295 sync.o \
296 sysconfig.o \
297 sysfs.o \
298 sysinfo.o \
299 syslwp.o \
300 times.o \
301 ulimit.o \
302 umask.o \
303 umount2.o \
304 utssys.o \
305 uucopy.o \
306 vhangup.o \
307 waitid.o \
308 write.o \
309 writev.o \
310 yield.o \
312 SYSOBJS= \
313 __clock_gettime.o \
314 __getcontext.o \
315 __uadmin.o \
316 __lwp_mutex_unlock.o \
317 __stack_grow.o \
318 door.o \
319 forkx.o \
320 forkallx.o \
321 getcontext.o \
322 gettimeofday.o \

```

```

323 lwp_private.o \
324 nuname.o \
325 ptrace.o \
326 syscall.o \
327 sysi86.o \
328 tls_get_addr.o \
329 uadmin.o \
330 umount.o \
331 uname.o \
332 vforkx.o \
333 xstat.o \
335 # objects under $(LIBCDIR)/port which contain transitional large file interfaces
336 PORTGEN64= \
337 __xftw64.o \
338 attropen64.o \
339 ftw64.o \
340 mkstemp64.o \
341 nftw64.o \
342 tell64.o \
343 truncate64.o \
345 # objects from source under $(LIBCDIR)/port
346 PORTFP= \
347 __flt_decim.o \
348 __flt_rounds.o \
349 __tbl_10_b.o \
350 __tbl_10_h.o \
351 __tbl_10_s.o \
352 __tbl_2_b.o \
353 __tbl_2_h.o \
354 __tbl_2_s.o \
355 __tbl_fdq.o \
356 __tbl_tens.o \
357 __x_power.o \
358 __base_sup.o \
359 aconvert.o \
360 decimal_bin.o \
361 double_decim.o \
362 econvert.o \
363 fconvert.o \
364 file_decim.o \
365 finite.o \
366 fp_data.o \
367 func_decim.o \
368 gconvert.o \
369 hex_bin.o \
370 ieee_globals.o \
371 pack_float.o \
372 sigfpe.o \
373 string_decim.o \
375 PORTGEN= \
376 __env_data.o \
377 __xftw.o \
378 a64l.o \
379 abort.o \
380 addsev.o \
381 ascii_strcasecmp.o \
382 ascii_strncasecmp.o \
383 assert.o \
384 atof.o \
385 atoi.o \
386 atol.o \
387 atoll.o \
388 attrat.o \

```

```

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

```

```

455 getpagesize.o \
456 getpw.o \
457 getpwnam.o \
458 getpwnam_r.o \
459 getrusage.o \
460 getspent.o \
461 getspent_r.o \
462 getsubopt.o \
463 gettxt.o \
464 getusershell.o \
465 getut.o \
466 getutx.o \
467 getvfsent.o \
468 getwd.o \
469 getwidth.o \
470 getxby_door.o \
471 gtxt.o \
472 hsearch.o \
473 iconv.o \
474 imaxabs.o \
475 imaxdiv.o \
476 index.o \
477 initgroups.o \
478 insque.o \
479 isaexec.o \
480 isastream.o \
481 isatty.o \
482 killpg.o \
483 klpdlib.o \
484 l64a.o \
485 lckpwwdf.o \
486 lconstants.o \
487 lexpl0.o \
488 lfind.o \
489 lfmt.o \
490 lfmt_log.o \
491 llabs.o \
492 lldiv.o \
493 llog10.o \
494 lltostr.o \
495 localtime.o \
496 lsearch.o \
497 madvise.o \
498 malloc.o \
499 memalign.o \
500 memmem.o \
501 mkdev.o \
502 mkttemp.o \
503 mkfifo.o \
504 mkstemp.o \
505 mktemp.o \
506 mlock.o \
507 mlockall.o \
508 mon.o \
509 msync.o \
510 munlock.o \
511 munlockall.o \
512 ndbm.o \
513 nftw.o \
514 nlspath_checks.o \
515 nsparse.o \
516 nss_common.o \
517 nss_dbdefs.o \
518 nss_deffinder.o \
519 opendir.o \
520 opt_data.o \

```

```

521 perror.o \
522 pfmt.o \
523 pfmt_data.o \
524 pfmt_print.o \
525 pipe.o \
526 plock.o \
527 poll.o \
528 posix_fadvise.o \
529 posix_fallocate.o \
530 posix_madvise.o \
531 posix_memalign.o \
532 priocntl.o \
533 privlib.o \
534 priv_str_xlate.o \
535 psecflags.o \
536 #endif /* ! codereview */
537 psiginfo.o \
538 psignal.o \
539 pt.o \
540 putpwent.o \
541 putspent.o \
542 raise.o \
543 rand.o \
544 random.o \
545 rctlops.o \
546 readdir.o \
547 readdir_r.o \
548 realpath.o \
549 reboot.o \
550 regexpr.o \
551 remove.o \
552 rewinddir.o \
553 rindex.o \
554 scandir.o \
555 seekdir.o \
556 select.o \
557 select_large_fdset.o \
558 setlabel.o \
559 setpriority.o \
560 settimeofday.o \
561 sh_locks.o \
562 sigflag.o \
563 siglist.o \
564 sigsend.o \
565 sigsetops.o \
566 signal.o \
567 stack.o \
568 stpcpy.o \
569 stpncpy.o \
570 str2sig.o \
571 strcase_ormap.o \
572 strchnul.o \
573 strcspn.o \
574 strdup.o \
575 strerror.o \
576 strlcat.o \
577 strlcpy.o \
578 strndup.o \
579 strpbrk.o \
580 strsep.o \
581 strsignal.o \
582 strspn.o \
583 strstr.o \
584 strtod.o \
585 strtimax.o \
586 strtok.o \

```

```

587 strtok_r.o \
588 strtoumax.o \
589 swab.o \
590 swapctl.o \
591 sysconf.o \
592 syslog.o \
593 tcdrain.o \
594 tcflow.o \
595 tcflush.o \
596 tcgetattr.o \
597 tcgetpgrp.o \
598 tcgetsid.o \
599 tcseendbreak.o \
600 tcsetattr.o \
601 tcsetpgrp.o \
602 tell.o \
603 telldir.o \
604 tfind.o \
605 time_data.o \
606 time_gdata.o \
607 timespec_get.o \
608 tls_data.o \
609 truncate.o \
610 tsdalloc.o \
611 tsearch.o \
612 ttyname.o \
613 ttyslot.o \
614 ualarm.o \
615 ucred.o \
616 valloc.o \
617 vlfmt.o \
618 vpfmt.o \
619 waitpid.o \
620 walkstack.o \
621 wdata.o \
622 xgetwidth.o \
623 xpg4.o \
624 xpg6.o \
626 PORTPRINT_W= \
627 doprnt_w.o \
629 PORTPRINT= \
630 asprintf.o \
631 doprnt.o \
632 fprintf.o \
633 printf.o \
634 snprintf.o \
635 sprintf.o \
636 vfprintf.o \
637 vprintf.o \
638 vsnprintf.o \
639 vsprintf.o \
640 vwprintf.o \
641 wprintf.o \
643 # c89 variants to support 32-bit size of c89 u/intmax_t (32-bit libc only)
644 PORTPRINT_C89= \
645 vfprintf_c89.o \
646 vprintf_c89.o \
647 vsnprintf_c89.o \
648 vsprintf_c89.o \
649 vwprintf_c89.o \
651 PORTSTDIO_C89= \
652 vscanf_c89.o \

```

```

653     vwscanf_c89.o      \
655 # portable stdio objects that contain large file interfaces.
656 # Note: fopen64 is a special case, as we build it small.
657 PORTSTDIO64=         \
658     fopen64.o         \
659     fpos64.o
661 PORTSTDIO_W=        \
662     doscan_w.o
664 PORTSTDIO=          \
665     __extensions.o   \
666     __endopen.o      \
667     _filbuf.o         \
668     _findbuf.o       \
669     _flsbuf.o        \
670     _wrtchk.o        \
671     clearerr.o       \
672     ctermid.o        \
673     ctermid_r.o     \
674     cuserid.o        \
675     data.o           \
676     doscan.o         \
677     fdopen.o         \
678     feof.o           \
679     ferrord.o        \
680     fgetc.o          \
681     fgets.o          \
682     fileno.o         \
683     flockf.o         \
684     flush.o          \
685     fopen.o          \
686     fpos.o           \
687     fputc.o          \
688     fputs.o          \
689     fread.o          \
690     fseek.o          \
691     fseeko.o         \
692     ftell.o          \
693     ftello.o         \
694     fwrite.o         \
695     getc.o           \
696     getchar.o        \
697     getline.o        \
698     getpass.o        \
699     gets.o           \
700     getw.o           \
701     mse.o            \
702     popen.o          \
703     putc.o           \
704     putchar.o        \
705     puts.o           \
706     putw.o           \
707     rewind.o         \
708     scanf.o          \
709     setbuf.o         \
710     setbuffer.o      \
711     setvbuf.o        \
712     system.o         \
713     tempnam.o        \
714     tmpfile.o        \
715     tmpnam_r.o       \
716     ungetc.o         \
717     vscanf.o         \
718     vwscanf.o

```

```

719     wscanf.o
721 PORTI18N=           \
722     getwchar.o       \
723     putwchar.o       \
724     putws.o          \
725     strtows.o        \
726     wcsnlen.o        \
727     wcsstr.o         \
728     wcstoimax.o     \
729     wcstol.o         \
730     wcstoul.o        \
731     wcswcs.o         \
732     wmemchr.o        \
733     wmemcmp.o        \
734     wmemcpy.o        \
735     wmemmove.o      \
736     wmemset.o        \
737     wscat.o          \
738     wchr.o           \
739     wscmp.o          \
740     wscpy.o          \
741     wscspn.o         \
742     wsdup.o          \
743     wslen.o          \
744     wsncat.o         \
745     wsncmp.o         \
746     wsncpy.o         \
747     wspbrk.o         \
748     wprintf.o        \
749     wsrchr.o         \
750     wsscanf.o        \
751     wssp.o           \
752     wstod.o          \
753     wstok.o          \
754     wstol.o          \
755     wstoll.o         \
756     wxfrm.o          \
757     gettext.o        \
758     gettext_gnu.o   \
759     gettext_real.o  \
760     gettext_util.o  \
761     plural_parser.o \
762     wdresolve.o     \
763     _ctype.o         \
764     isascii.o        \
765     toascii.o
767 PORTI18N_COND=      \
768     wcstol_longlong.o \
769     wcstoul_longlong.o
771 PORTLOCALE=         \
772     big5.o           \
773     btowc.o          \
774     collate.o        \
775     collcmp.o        \
776     euc.o            \
777     fnmatch.o        \
778     fgetwc.o         \
779     fgetws.o         \
780     fix_grouping.o   \
781     fputwc.o         \
782     fputws.o         \
783     fwide.o          \
784     gb18030.o

```



```

785 gb2312.o \
786 gbk.o \
787 getdate.o \
788 isdigit.o \
789 iswctype.o \
790 ldpart.o \
791 lmessages.o \
792 lnumeric.o \
793 lmonetary.o \
794 localeconv.o \
795 localeimpl.o \
796 mbftowc.o \
797 mblen.o \
798 mbrlen.o \
799 mbrtowc.o \
800 mbsinit.o \
801 mbsnrtowcs.o \
802 mbsrtowcs.o \
803 mbstowcs.o \
804 mbtowc.o \
805 mskanji.o \
806 nextwctype.o \
807 nl_langinfo.o \
808 none.o \
809 regcomp.o \
810 regfree.o \
811 regerror.o \
812 regexec.o \
813 rune.o \
814 runetype.o \
815 setlocale.o \
816 setrunelocale.o \
817 strcasecmp.o \
818 strcasestr.o \
819 strcoll.o \
820 strfmon.o \
821 strftime.o \
822 strncasecmp.o \
823 strtptime.o \
824 strxfrm.o \
825 table.o \
826 timelocal.o \
827 tolower.o \
828 towlower.o \
829 ungetwc.o \
830 utf8.o \
831 wcrctomb.o \
832 wcscasecmp.o \
833 wcscoll.o \
834 wcsftime.o \
835 wcsnrtombs.o \
836 wcsrtombs.o \
837 wcswidth.o \
838 wcstombs.o \
839 wcsxfrm.o \
840 wctob.o \
841 wctomb.o \
842 wctrans.o \
843 wctype.o \
844 wcwidth.o \
845 wscoll.o \
847 AIOBJS= \
848 aio.o \
849 aio_alloc.o \
850 posix_aio.o

```

```

852 RTOBJS= \
853 clock_timer.o \
854 mqueue.o \
855 posixobj.o \
856 sched.o \
857 sem.o \
858 shm.o \
859 sigev_thread.o \
861 TPOOLBJS= \
862 thread_pool.o \
864 THREADSOBJS= \
865 alloc.o \
866 assfail.o \
867 cancel.o \
868 cli_thr.o \
869 door_calls.o \
870 tmem.o \
871 pthr_attr.o \
872 pthr_barrier.o \
873 pthr_cond.o \
874 pthr_mutex.o \
875 pthr_rwlock.o \
876 pthread.o \
877 rwlock.o \
878 scalls.o \
879 sema.o \
880 sigaction.o \
881 spawn.o \
882 synchron.o \
883 tdb_agent.o \
884 thr.o \
885 thread_interface.o \
886 tls.o \
887 tsd.o \
889 THREADSMACHOBJS= \
890 machdep.o \
892 THREADSASMOBJS= \
893 asm_subr.o \
895 UNICODEOBJS= \
896 u8_textprep.o \
897 uconv.o \
899 UNWINDMACHOBJS= \
900 unwind.o \
902 UNWINDASMOBJS= \
903 unwind_frame.o \
905 # objects that implement the transitional large file API
906 PORTSYS64= \
907 lockf64.o \
908 stat64.o \
910 PORTSYS= \
911 _autofssys.o \
912 access.o \
913 acctctl.o \
914 bsd_signal.o \
915 chmod.o \
916 chown.o

```

```

917 corectl.o \
918 epoll.o \
919 eventfd.o \
920 exaccts.o \
921 execl.o \
922 execl.o \
923 execv.o \
924 fcntl.o \
925 getpagesizes.o \
926 getpeerucred.o \
927 inst_sync.o \
928 issetugid.o \
929 label.o \
930 link.o \
931 lockf.o \
932 lwp.o \
933 lwp_cond.o \
934 lwp_rwlock.o \
935 lwp_sigmask.o \
936 meminfosys.o \
937 mkdir.o \
938 mknod.o \
939 msgsys.o \
940 nfssys.o \
941 open.o \
942 pgrpsys.o \
943 posix_sigwait.o \
944 ppriv.o \
945 psetsys.o \
946 rctlsys.o \
947 readlink.o \
948 rename.o \
949 sbrk.o \
950 semsys.o \
951 set_errno.o \
952 sharefs.o \
953 shmsys.o \
954 sidsys.o \
955 siginterrupt.o \
956 signal.o \
957 signalfd.o \
958 sigpending.o \
959 sigstack.o \
960 stat.o \
961 symlink.o \
962 tasksys.o \
963 time.o \
964 time_util.o \
965 timerfd.o \
966 ucontext.o \
967 unlink.o \
968 ustat.o \
969 utimesys.o \
970 zone.o \
971 \
972 PORTREGEX= \
973 glob.o \
974 regcmp.o \
975 regex.o \
976 wordexp.o \
977 \
978 PORTREGEX64= \
979 glob64.o \
980 \
981 MOSTOBS= \
982 $(STRETS) \

```

```

983 $(CRTOBS) \
984 $(DYNOBJS) \
985 $(FPOBJS) \
986 $(FPASMOBJS) \
987 $(ATOMICOBJS) \
988 $(CHACHAOBJS) \
989 $(XATROBJS) \
990 $(COMOBS) \
991 $(DTRACEOBS) \
992 $(GENOBS) \
993 $(PORTFP) \
994 $(PORTGEN) \
995 $(PORTGEN64) \
996 $(PORTI18N) \
997 $(PORTI18N_COND) \
998 $(PORTLOCALE) \
999 $(PORTPRINT) \
1000 $(PORTPRINT_C89) \
1001 $(PORTPRINT_W) \
1002 $(PORTREGEX) \
1003 $(PORTREGEX64) \
1004 $(PORTSTDIO) \
1005 $(PORTSTDIO64) \
1006 $(PORTSTDIO_C89) \
1007 $(PORTSTDIO_W) \
1008 $(PORTSYS) \
1009 $(PORTSYS64) \
1010 $(AIOOBS) \
1011 $(RTOBS) \
1012 $(SECFLAGSOBS) \
1013 #endif /* ! codereview */
1014 $(TPOOLOBS) \
1015 $(THREADSOBS) \
1016 $(THREADSMACHOBS) \
1017 $(THREADSASMOBS) \
1018 $(UNICODEOBS) \
1019 $(UNWINDMACHOBS) \
1020 $(UNWINDASMOBS) \
1021 $(COMSYSOBS) \
1022 $(SYSOBS) \
1023 $(COMSYSOBS64) \
1024 $(SYSOBS64) \
1025 $(VALUES) \
1026 \
1027 TRACEOBS= \
1028 plockstat.o \
1029 \
1030 # NOTE: libc.so.1 must be linked with the minimal crti.o and crtn.o
1031 # modules whose source is provided in the $(SRC)/lib/common directory.
1032 # This must be done because otherwise the Sun C compiler would insert
1033 # its own versions of these modules and those versions contain code
1034 # to call out to C++ initialization functions. Such C++ initialization
1035 # functions can call back into libc before thread initialization is
1036 # complete and this leads to segmentation violations and other problems.
1037 # Since libc contains no C++ code, linking with the minimal crti.o and
1038 # crtn.o modules is safe and avoids the problems described above.
1039 OBJECTS= $(CRTI) $(MOSTOBS) $(CRTN)
1040 CRTSRCS= ../../common/i386
1041 \
1042 LDPASS_OFF= $(POUND_SIGN) \
1043 \
1044 # include common library definitions
1045 include ../../Makefile.lib \
1046 \
1047 # we need to override the default SONAME here because we might
1048 # be building a variant object (still libc.so.1, but different filename)

```

```

1049 SONAME = libc.so.1

1051 CFLAGS += $(CCVERBOSE) $(CTF_FLAGS)

1053 # This is necessary to avoid problems with calling _ex_unwind().
1054 # We probably don't want any inlining anyway.
1055 XINLINE = -xinline=
1056 CFLAGS += $(XINLINE)

1058 CERRWARN += -_gcc=-Wno-parentheses
1059 CERRWARN += -_gcc=-Wno-switch
1060 CERRWARN += -_gcc=-Wno-uninitialized
1061 CERRWARN += -_gcc=-Wno-unused-value
1062 CERRWARN += -_gcc=-Wno-unused-label
1063 CERRWARN += -_gcc=-Wno-unused-variable
1064 CERRWARN += -_gcc=-Wno-type-limits
1065 CERRWARN += -_gcc=-Wno-char-subscripts
1066 CERRWARN += -_gcc=-Wno-clobbered
1067 CERRWARN += -_gcc=-Wno-unused-function
1068 CERRWARN += -_gcc=-Wno-address

1070 # Setting THREAD_DEBUG = -DTHREAD_DEBUG (make THREAD_DEBUG=-DTHREAD_DEBUG ...)
1071 # enables ASSERT() checking in the threads portion of the library.
1072 # This is automatically enabled for DEBUG builds, not for non-debug builds.
1073 THREAD_DEBUG =
1074 $(NOT_RELEASE_BUILD)THREAD_DEBUG = -DTHREAD_DEBUG

1076 # Make string literals read-only to save memory.
1077 CFLAGS += $(XSTRCONST)

1079 ALTPICIS= $(TRACEOBS:=%=pics/%)

1081 $(DYNLIB) := BUILD.SO = $(LD) -o $@ -G $(DYNFLAGS) $(PICIS) $(ALTPICIS) \
1082     $(EXTPICIS) $(LDLIBS)

1084 MAPFILES = $(LIBCDIR)/port/mapfile-vers

1086 #
1087 # EXTN_CPPFLAGS and EXTN_CFLAGS set in enclosing Makefile
1088 #
1089 CFLAGS += $(EXTN_CFLAGS)
1090 CPPFLAGS= -D_REENTRANT -Di386 $(EXTN_CPPFLAGS) $(THREAD_DEBUG) \
1091     -I$(LIBCBASE)/inc -I$(LIBCDIR)/inc $(CPPFLAGS.master)
1092 ASFLAGS= $(AS_PICFLAGS) -P -D__STDC__ -D_ASM $(CPPFLAGS) $(i386_AS_XARCH)

1094 # As a favor to the dtrace syscall provider, libc still calls the
1095 # old syscall traps that have been obsoleted by the *at() interfaces.
1096 # Delete this to compile libc using only the new *at() system call traps
1097 CPPFLAGS += -D_RETAIN_OLD_SYSCALLS

1099 # Inform the run-time linker about libc specialized initialization
1100 RTLDINFO = -z rtldinfo=tls_rtldinfo
1101 DYNFLAGS += $(RTLDINFO)

1103 # Force libc's internal references to be resolved immediately upon loading
1104 # in order to avoid critical region problems. Since almost all libc symbols
1105 # are marked 'protected' in the mapfiles, this is a minimal set (15 to 20).
1106 DYNFLAGS += -znow

1108 DYNFLAGS += -e __rtboot
1109 DYNFLAGS += $(EXTN_DYNFLAGS)

1111 # Inform the kernel about the initial DTrace area (in case
1112 # libc is being used as the interpreter / runtime linker).
1113 DTRACE_DATA = -zdtrace=dtrace_data
1114 DYNFLAGS += $(DTRACE_DATA)

```

```

1116 # DTrace needs an executable data segment.
1117 MAPFILE.NED=

1119 BUILD.s= $(AS) $(ASFLAGS) $< -o $@

1121 # Override this top level flag so the compiler builds in its native
1122 # C99 mode. This has been enabled to support the complex arithmetic
1123 # added to libc.
1124 C99MODE= $(C99_ENABLE)

1126 # libc method of building an archive
1127 # The "$(GREP) -v ' L '" part is necessary only until
1128 # lorder is fixed to ignore thread-local variables.
1129 BUILD.AR= $(RM) $@ ; \
1130     $(AR) q $@ `$(LORDER) $(MOSTOBS:=%=$(DIR)/%) | $(GREP) -v ' L ' | $(TSOR)

1132 # extra files for the clean target
1133 CLEANFILES= \
1134     $(LIBCDIR)/port/gen/errlst.c \
1135     $(LIBCDIR)/port/gen/new_list.c \
1136     assym.h \
1137     genassym \
1138     crt/_rtld.s \
1139     crt/_rtbootld.s \
1140     pics/_rtbootld.o \
1141     pics/crti.o \
1142     pics/crtn.o \
1143     $(ALTPICIS)

1145 CLOBBERFILES += $(LIB_PIC)

1147 # list of C source for lint
1148 SRCS= \
1149     $(ATOMICOBJS:.%=$(SRC)/common/atomic/%.c) \
1150     $(XATTROBJS:.%=$(SRC)/common/xattr/%.c) \
1151     $(COMOBS:.%=$(SRC)/common/util/%.c) \
1152     $(DTRACEOBS:.%=$(SRC)/common/dtrace/%.c) \
1153     $(PORTFP:.%=$(LIBCDIR)/port/fp/%.c) \
1154     $(PORTGEN:.%=$(LIBCDIR)/port/gen/%.c) \
1155     $(PORTI18N:.%=$(LIBCDIR)/port/il8n/%.c) \
1156     $(PORTLOCALE:.%=$(LIBCDIR)/port/locale/%.c) \
1157     $(PORTPRINT:.%=$(LIBCDIR)/port/print/%.c) \
1158     $(PORTREGEX:.%=$(LIBCDIR)/port/regex/%.c) \
1159     $(PORTSTDIO:.%=$(LIBCDIR)/port/stdio/%.c) \
1160     $(PORTSYS:.%=$(LIBCDIR)/port/sys/%.c) \
1161     $(AIOBJS:.%=$(LIBCDIR)/port/aio/%.c) \
1162     $(RTOBJS:.%=$(LIBCDIR)/port/rt/%.c) \
1163     $(SECFLAGSOBJS:.%=$(SRC)/common/secflags/%.c) \
1164 #endif /* ! codereview */
1165     $(TPOOLBJS:.%=$(LIBCDIR)/port/tpool/%.c) \
1166     $(THREADSOBJS:.%=$(LIBCDIR)/port/threads/%.c) \
1167     $(THREADSMACHOBJS:.%=$(LIBCDIR)/$(MACH)/threads/%.c) \
1168     $(UNICODEOBS:.%=$(SRC)/common/unicode/%.c) \
1169     $(UNWINDMACHOBJS:.%=$(LIBCDIR)/port/unwind/%.c) \
1170     $(FPOBJS:.%=$(LIBCDIR)/$(MACH)/fp/%.c) \
1171     $(LIBCBASE)/gen/ecvt.c \
1172     $(LIBCBASE)/gen/makectxt.c \
1173     $(LIBCBASE)/gen/signfolst.c \
1174     $(LIBCBASE)/gen/siglongjmp.c \
1175     $(LIBCBASE)/gen/strcmp.c \
1176     $(LIBCBASE)/gen/sync_instruction_memory.c \
1177     $(LIBCBASE)/sys/ptrace.c \
1178     $(LIBCBASE)/sys/uadmin.c

1180 # conditional assignments

```

```

1181 $(DYNLIB) := CRTI = crt1.o
1182 $(DYNLIB) := CRTN = crtn.o

1184 # Files which need the threads .il inline template
1185 TIL=
1186     aio.o
1187     alloc.o
1188     assfail.o
1189     atexit.o
1190     atfork.o
1191     cancel.o
1192     door_calls.o
1193     err.o
1194     errno.o
1195     lwp.o
1196     ma.o
1197     machdep.o
1198     posix_aio.o
1199     pthr_attr.o
1200     pthr_barrier.o
1201     pthr_cond.o
1202     pthr_mutex.o
1203     pthr_rwlock.o
1204     pthread.o
1205     rand.o
1206     rwlock.o
1207     scalls.o
1208     sched.o
1209     sema.o
1210     sigaction.o
1211     sigev_thread.o
1212     spawn.o
1213     stack.o
1214     synch.o
1215     tdb_agent.o
1216     thr.o
1217     thread_interface.o
1218     thread_pool.o
1219     tls.o
1220     tsd.o
1221     tmem.o
1222     unwind.o

1224 THREADS_INLINES = $(LIBCBASE)/threads/i386.il
1225 $(TIL:%=pics/%) := CFLAGS += $(THREADS_INLINES)

1227 # pics/mul64.o := CFLAGS += $(LIBCBASE)/crt/mul64.il

1229 # large-file-aware components that should be built large

1231 $(COMSYSOBS64:%=pics/%) := \
1232     CPPFLAGS += -D_LARGEFILE_SOURCE -D_FILE_OFFSET_BITS=64

1234 $(SYSOBS64:%=pics/%) := \
1235     CPPFLAGS += -D_LARGEFILE_SOURCE -D_FILE_OFFSET_BITS=64

1237 $(PORTGEN64:%=pics/%) := \
1238     CPPFLAGS += -D_LARGEFILE_SOURCE -D_FILE_OFFSET_BITS=64

1240 $(PORTREGEX64:%=pics/%) := \
1241     CPPFLAGS += -D_LARGEFILE_SOURCE -D_FILE_OFFSET_BITS=64

1243 $(PORTSTDIO64:%=pics/%) := \
1244     CPPFLAGS += -D_LARGEFILE_SOURCE -D_FILE_OFFSET_BITS=64

1246 $(PORTSYS64:%=pics/%) := \

```

```

1247     CPPFLAGS += -D_LARGEFILE_SOURCE -D_FILE_OFFSET_BITS=64

1249 $(PORTSTDIO_W:%=pics/%) := \
1250     CPPFLAGS += -D_WIDE

1252 $(PORTPRINT_W:%=pics/%) := \
1253     CPPFLAGS += -D_WIDE

1255 $(PORTPRINT_C89:%=pics/%) := \
1256     CPPFLAGS += -D_C89_INTMAX32

1258 $(PORTSTDIO_C89:%=pics/%) := \
1259     CPPFLAGS += -D_C89_INTMAX32

1261 $(PORTI18N_COND:%=pics/%) := \
1262     CPPFLAGS += -D_WCS_LONGLONG

1264 pics/arc4random.o := CPPFLAGS += -I$(SRC)/common/crypto/chacha

1266 .KEEP_STATE:

1268 all: $(LIBS) $(LIB_PIC)

1270 lint := CPPFLAGS += -I$(LIBCDIR)/$(MACH)/fp
1271 lint := CPPFLAGS += -D_MSE_INT_H -D_LCONV_C99
1272 lint := LINTFLAGS += -mn -eroff=E_SUPPRESSION_DIRECTIVE_UNUSED

1274 lint:
1275     @echo $(LINT.c) ...
1276     @$$(LINT.c) $(SRCS) $(LDLIBS)

1278 $(LINTLIB) := SRCS=$(LIBCDIR)/port/llib-1c
1279 $(LINTLIB) := CPPFLAGS += -D_MSE_INT_H
1280 $(LINTLIB) := LINTFLAGS=-nvx

1282 # object files that depend on inline template
1283 $(TIL:%=pics/%) := $(LIBCBASE)/threads/i386.il
1284 # pics/mul64.o := $(LIBCBASE)/crt/mul64.il

1286 # include common libc targets
1287 include $(LIBCDIR)/Makefile.targ

1289 # We need to strip out all CTF and DOF data from the static library
1290 $(LIB_PIC) := DIR = pics
1291 $(LIB_PIC): pics $$ (PICS)
1292     $(BUILD.AR)
1293     $(MCS) -d -n .SUNW_ctf $@ > /dev/null 2>&1
1294     $(MCS) -d -n .SUNW_dof $@ > /dev/null 2>&1
1295     $(AR) -ts $@ > /dev/null
1296     $(POST_PROCESS_A)

1298 $(LIBCBASE)/crt/_rtbootld.s := $(LIBCBASE)/crt/_rtboot.s $(LIBCBASE)/crt/_rtld.c
1299     $(CC) $(CPPFLAGS) $(CTF_FLAGS) -o -s $(C_PICFLAGS) \
1300     $(LIBCBASE)/crt/_rtld.c -o $(LIBCBASE)/crt/_rtld.s
1301     $(CAT) $(LIBCBASE)/crt/_rtboot.s $(LIBCBASE)/crt/_rtld.s > $@
1302     $(RM) $(LIBCBASE)/crt/_rtld.s

1304 # partially built from C source
1305 pics/_rtbootld.o := $(LIBCBASE)/crt/_rtbootld.s
1306     $(AS) $(ASFLAGS) $(LIBCBASE)/crt/_rtbootld.s -o $@
1307     $(CTFCONVERT_O)

1309 ASSYMDEP_OBJS=
1310     _lwp_mutex_unlock.o
1311     _stack_grow.o
1312     getcontext.o

```

```
1313     setjmp.o           \
1314     tls_get_addr.o    \
1315     vforkx.o

1317 $(ASSYMDEP_OBJS:%=pics/%) := CPPFLAGS += -I.

1319 $(ASSYMDEP_OBJS:%=pics/%) : assym.h

1321 # assym.h build rules

1323 GENASSYM_C = $(LIBCDIR)/$(MACH)/genassym.c

1325 genassym: $(GENASSYM_C)
1326     $(NATIVECC) $(NATIVE_CFLAGS) -I$(LIBCBASE)/inc -I$(LIBCDIR)/inc \
1327     -D__EXTENSIONS__ $(CPPFLAGS.native) -o $@ $(GENASSYM_C)

1329 OFFSETS = $(LIBCDIR)/$(MACH)/offsets.in

1331 assym.h: $(OFFSETS) genassym
1332     $(OFFSETS_CREATE) <$(OFFSETS) >$@
1333     ./genassym >>$@

1335 # derived C source and related explicit dependencies
1336 $(LIBCDIR)/port/gen/errlst.c + \
1337 $(LIBCDIR)/port/gen/new_list.c: $(LIBCDIR)/port/gen/errlist $(LIBCDIR)/port/gen/
1338     cd $(LIBCDIR)/port/gen; pwd; $(AWK) -f errlist.awk < errlist

1340 pics/errlst.o: $(LIBCDIR)/port/gen/errlst.c

1342 pics/new_list.o: $(LIBCDIR)/port/gen/new_list.c
```

new/usr/src/lib/libc/port/gen/priv_str_xlate.c

1

```
*****  
10845 Thu Jun 30 22:02:10 2016  
new/usr/src/lib/libc/port/gen/priv_str_xlate.c  
7029 want per-process exploit mitigation features (secflags)  
7030 want basic address space layout randomization (aslr)  
7031 noexec_user_stack should be a secflag  
7032 want a means to forbid mappings around NULL.  
*****  
_____unchanged_portion_omitted_
```

```

*****
2488 Thu Jun 30 22:02:11 2016
new/usr/src/lib/libc/port/gen/psecflags.c
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
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 /* Copyright 2015, Richard Lowe. */
14 #include "lint.h"
16 #include <errno.h>
17 #include <stdio.h>
18 #include <stdlib.h>
19 #include <string.h>
20 #include <strings.h>
22 #include <sys/proc.h>
23 #include <sys/procset.h>
24 #include <sys/syscall.h>
25 #include <sys/secflags.h>
27 extern int __psecflagsset(procset_t *, psecflagwhich_t, secflagdelta_t *);
29 int
30 psecflags(idtype_t idtype, id_t id, psecflagwhich_t which,
31          secflagdelta_t *delta)
32 {
33     procset_t procset;
35     setprocset(&procset, POP_AND, idtype, id, P_ALL, 0);
37     return (__psecflagsset(&procset, which, delta));
38 }
40 int
41 secflags_parse(const secflagset_t *defaults, const char *flags,
42              secflagdelta_t *ret)
43 {
44     char *flag;
45     char *s, *ss;
46     boolean_t current = B_FALSE;
48     /* Guarantee a clean base */
49     bzero(ret, sizeof (*ret));
51     if ((ss = s = strdup(flags)) == NULL)
52         return (-1); /* errno set for us */
54     while ((flag = strsep(&s, ",")) != NULL) {
55         secflag_t sf = 0;
56         boolean_t del = B_FALSE;

```

```

59         if (strcasecmp(flag, "default") == 0) {
60             if (defaults != NULL) {
61                 secflags_union(&ret->psd_add, defaults);
62             } else {
63                 free(ss);
64                 errno = EINVAL;
65                 return (-1);
66             }
67             continue;
68         } else if (strcasecmp(flag, "all") == 0) {
69             secflags_fullset(&ret->psd_add);
70             continue;
71         } else if (strcasecmp(flag, "none") == 0) {
72             secflags_fullset(&ret->psd_rem);
73             continue;
74         } else if (strcasecmp(flag, "current") == 0) {
75             current = B_TRUE;
76             continue;
77         }
79         if ((flag[0] == '-') || (flag[0] == '!')) {
80             flag++;
81             del = B_TRUE;
82         } else if (flag[0] == '+') {
83             flag++;
84         }
86         if ((secflag_by_name(flag, &sf)) != B_TRUE) {
87             free(ss);
88             errno = EINVAL;
89             return (-1);
90         }
92         if (del)
93             secflag_set(&(ret->psd_rem), sf);
94         else
95             secflag_set(&(ret->psd_add), sf);
96     }
98     /*
99     * If we're not using the current flags, this is strict assignment.
100    * Negatives "win".
101    */
102     if (!current) {
103         secflags_copy(&ret->psd_assign, &ret->psd_add);
104         secflags_difference(&ret->psd_assign, &ret->psd_rem);
105         ret->psd_ass_active = B_TRUE;
106         secflags_zero(&ret->psd_add);
107         secflags_zero(&ret->psd_rem);
108     }
110     free(ss);
111     return (0);
112 }
113 #endif /* ! codereview */

```

```

*****
58809 Thu Jun 30 22:02:12 2016
new/usr/src/lib/libc/port/mapfile-vers
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
_____unchanged_portion_omitted_____

2793 # There should never be more than one SUNWprivate version.
2794 # Don't add any more. Add new private symbols to SUNWprivate_1.1

2796 SYMBOL_VERSION SUNWprivate_1.1 {
2797     global:
2798         __Argv           { FLAGS = NODIRECT };
2799         cfree            { FLAGS = NODIRECT };
2800         __cswidth;
2801         __ctype_mask;
2802         __environ_lock  { FLAGS = NODIRECT };
2803         __inf_read;
2804         __inf_written;
2805         __i_size;
2806         __isnanf        { TYPE = FUNCTION; FILTER = libm.so.2 };
2807         __iswrunes;
2808         __libc_threaded;
2809         __lib_version   { FLAGS = NODIRECT };
2810         __logb          { TYPE = FUNCTION; FILTER = libm.so.2 };
2811         __lone          { FLAGS = NODYNSORT };
2812         __lten          { FLAGS = NODYNSORT };
2813         __lzero        { FLAGS = NODYNSORT };
2814         __malloc_lock;
2815         __memcmp;
2816         __memcpy       { FLAGS = NODYNSORT };
2817         __memmove;
2818         __memset;
2819         __modff        { TYPE = FUNCTION; FILTER = libm.so.2 };
2820         __nan_read;
2821         __nan_written;
2822         __nextwctype;
2823         __nis_debug_bind;
2824         __nis_debug_calls;
2825         __nis_debug_file;
2826         __nis_debug_rpc;
2827         __nis_prefsrv;
2828         __nis_preftype;
2829         __nis_server;
2830         __nss_default_finders;
2831         __progname     { FLAGS = NODIRECT };
2832         __smbuf;
2833         __sp;
2834         __strdupa_str  { FLAGS = NODIRECT };
2835         __strdupa_len { FLAGS = NODIRECT };
2836         __tdb_bootstrap;
2837         __threaded;
2838         thr_probe_getfunc_addr;
2839         __trans_lower;
2840         __trans_upper;
2841         __uberdata;
2842         __xpg6        { FLAGS = NODIRECT };

2844 $if _ELF32
2845     __dladdr        { TYPE = FUNCTION; FILTER = /usr/lib/ld.so.1 };
2846     __dladdr1       { TYPE = FUNCTION; FILTER = /usr/lib/ld.so.1 };

```

```

2847     __dlclose       { TYPE = FUNCTION; FILTER = /usr/lib/ld.so.1 };
2848     __dldump        { TYPE = FUNCTION; FILTER = /usr/lib/ld.so.1 };
2849     __dlerror       { TYPE = FUNCTION; FILTER = /usr/lib/ld.so.1 };
2850     __dlinfo        { TYPE = FUNCTION; FILTER = /usr/lib/ld.so.1 };
2851     __dlmopen       { TYPE = FUNCTION; FILTER = /usr/lib/ld.so.1 };
2852     __dlopen        { TYPE = FUNCTION; FILTER = /usr/lib/ld.so.1 };
2853     __dlsym         { TYPE = FUNCTION; FILTER = /usr/lib/ld.so.1 };
2854     __ld_libc       { TYPE = FUNCTION; FILTER = /usr/lib/ld.so.1 };
2855     __sys_errlist;
2856     __sys_errs;
2857     __sys_index;
2858     __sys_nerr      { FLAGS = NODYNSORT };
2859     __sys_num_err;
2860 $elif sparcv9
2861     __dladdr        { TYPE = FUNCTION; FILTER = /usr/lib/sparcv9/ld.so.1 };
2862     __dladdr1       { TYPE = FUNCTION; FILTER = /usr/lib/sparcv9/ld.so.1 };
2863     __dlclose       { TYPE = FUNCTION; FILTER = /usr/lib/sparcv9/ld.so.1 };
2864     __dldump        { TYPE = FUNCTION; FILTER = /usr/lib/sparcv9/ld.so.1 };
2865     __dlerror       { TYPE = FUNCTION; FILTER = /usr/lib/sparcv9/ld.so.1 };
2866     __dlinfo        { TYPE = FUNCTION; FILTER = /usr/lib/sparcv9/ld.so.1 };
2867     __dlmopen       { TYPE = FUNCTION; FILTER = /usr/lib/sparcv9/ld.so.1 };
2868     __dlopen        { TYPE = FUNCTION; FILTER = /usr/lib/sparcv9/ld.so.1 };
2869     __dlsym         { TYPE = FUNCTION; FILTER = /usr/lib/sparcv9/ld.so.1 };
2870     __ld_libc       { TYPE = FUNCTION; FILTER = /usr/lib/sparcv9/ld.so.1 };
2871 $elif amd64
2872     __dladdr        { TYPE = FUNCTION; FILTER = /usr/lib/amd64/ld.so.1 };
2873     __dladdr1       { TYPE = FUNCTION; FILTER = /usr/lib/amd64/ld.so.1 };
2874     __dlamd64getunwind { TYPE = FUNCTION; FILTER = /usr/lib/amd64/ld.so.1 };
2875     __dlclose       { TYPE = FUNCTION; FILTER = /usr/lib/amd64/ld.so.1 };
2876     __dldump        { TYPE = FUNCTION; FILTER = /usr/lib/amd64/ld.so.1 };
2877     __dlerror       { TYPE = FUNCTION; FILTER = /usr/lib/amd64/ld.so.1 };
2878     __dlinfo        { TYPE = FUNCTION; FILTER = /usr/lib/amd64/ld.so.1 };
2879     __dlmopen       { TYPE = FUNCTION; FILTER = /usr/lib/amd64/ld.so.1 };
2880     __dlopen        { TYPE = FUNCTION; FILTER = /usr/lib/amd64/ld.so.1 };
2881     __dlsym         { TYPE = FUNCTION; FILTER = /usr/lib/amd64/ld.so.1 };
2882     __ld_libc       { TYPE = FUNCTION; FILTER = /usr/lib/amd64/ld.so.1 };
2883 $else
2884 $error unknown platform
2885 $endif

2887 $if _sparc
2888     __lyday_to_month;
2889     __mon_lengths;
2890     __yday_to_month;
2891 $endif
2892 $if i386
2893     __sse_hw;
2894 $endif

2896     protected:
2897     acctctl;
2898     allocids;
2899     __assert_c99;
2900     __assert_c99;
2901     __assfail;
2902     attr_count;
2903     attr_to_data_type;
2904     attr_to_name;
2905     attr_to_option;
2906     attr_to_xattr_view;
2907     __autofssys;
2908     __bufsync;
2909     __cladm;
2910     __class_quadruple;
2911     core_get_default_content;
2912     core_get_default_path;

```



```

2913 core_get_global_content;
2914 core_get_global_path;
2915 core_get_options;
2916 core_get_process_content;
2917 core_get_process_path;
2918 core_set_default_content;
2919 core_set_default_path;
2920 core_set_global_content;
2921 core_set_global_path;
2922 core_set_options;
2923 core_set_process_content;
2924 core_set_process_path;
2925 dbm_close_status;
2926 dbm_do_nextkey;
2927 dbm_setdefwrite;
2928 _D_cplx_div;
2929 _D_cplx_div_ix;
2930 _D_cplx_div_rx;
2931 _D_cplx_mul;
2932 defclose_r;
2933 defcntl;
2934 defcntl_r;
2935 defopen;
2936 defopen_r;
2937 defread;
2938 defread_r;
2939 _delete;
2940 _dgettext;
2941 _doprnt;
2942 _doscan;
2943 _errfp;
2944 _errxfp;
2945 exportfs;
2946 _F_cplx_div;
2947 _F_cplx_div_ix;
2948 _F_cplx_div_rx;
2949 _F_cplx_mul;
2950 __fgetwc_xpg5;
2951 __fgetws_xpg5;
2952 _findbuf;
2953 _findiop;
2954 __fini_daemon_priv;
2955 _finite;
2956 _forkl { FLAGS = NODYSORT };
2957 _forkall { FLAGS = NODYSORT };
2958 _fpclass;
2959 _fpgetmask;
2960 _fpgetround;
2961 _fpgetsticky;
2962 _fprintf;
2963 _fpsetmask;
2964 _fpsetround;
2965 _fpsetsticky;
2966 __fputwc_xpg5;
2967 __fputws_xpg5;
2968 _ftw;
2969 _gcvt;
2970 _getarg;
2971 __getcontext;
2972 _getdents;
2973 _get_exit_frame_monitor;
2974 _getfp;
2975 _getgroupsbymember;
2976 _getlogin_r;
2977 getrandom;
2978 _getsp;

```

```

2979 __gettsp;
2980 getvmusage;
2981 __getwchar_xpg5;
2982 __getwc_xpg5;
2983 gtty;
2984 __idmap_flush_kcache;
2985 __idmap_reg;
2986 __idmap_unreg;
2987 __init_daemon_priv;
2988 __init_suid_priv;
2989 _insert;
2990 inst_sync;
2991 _iswctype;
2992 klpd_create;
2993 klpd_getpath;
2994 klpd_getport;
2995 klpd_getured;
2996 klpd_register;
2997 klpd_register_id;
2998 klpd_unregister;
2999 klpd_unregister_id;
3000 __lgrp_home_fast { FLAGS = NODYSORT };
3001 __lgrpsys;
3002 _lltostr;
3003 __lock_clear;
3004 __lock_try;
3005 __ltzset;
3006 lwp_self;
3007 makeut;
3008 makeutx;
3009 __mbftowc;
3010 mcfiller;
3011 mntopt;
3012 modctl;
3013 modutx;
3014 msgctl64;
3015 __multi_innetgr;
3016 __mutex_destroy { FLAGS = NODYSORT };
3017 mutex_enter;
3018 mutex_exit;
3019 mutex_held;
3020 __mutex_init { FLAGS = NODYSORT };
3021 __mutex_unlock { FLAGS = NODYSORT };
3022 name_to_attr;
3023 nfs_getfh;
3024 nfssvc;
3025 _nfssys;
3026 __nis_get_environment;
3027 __nss_db_state_destr;
3028 nss_default_key2str;
3029 nss_delete;
3030 nss_endent;
3031 nss_getent;
3032 __nss_initf_group;
3033 __nss_initf_netgroup;
3034 __nss_initf_passwd;
3035 __nss_initf_shadow;
3036 nss_packed_arg_init;
3037 nss_packed_context_init;
3038 nss_packed_getkey;
3039 nss_packed_set_status;
3040 nss_search;
3041 nss_setent;
3042 __nss_XbyY_fgets;
3043 __nsw_extended_action_v1;
3044 __nsw_freeconfig_v1;

```

```

3045     __nsw_getconfig_v1;
3046     __nthreads;
3047     __openatrrdirat;
3048     option_to_attr;
3049     __priv_bracket;
3050     __priv_relinquish;
3051     psecflags;
3052 #endif /* ! codereview */
3053     pset_assign_forced;
3054     pset_bind_lwp;
3055     _psignal;
3056     pthread_attr_getdaemonstate_np;
3057     pthread_attr_setdaemonstate_np;
3058     _pthread_setcleanupinit;
3059     __putwchar_xpg5;
3060     __putwc_xpg5;
3061     rctlctl;
3062     rctlctl;
3063     _realbufend;
3064     _resume;
3065     _resume_ret;
3066     _rpcsys;
3067     _sbrk_grow_aligned;
3068     scrwidth;
3069     secflag_by_name;
3070     secflag_clear;
3071     secflags_copy;
3072     secflags_difference;
3073     secflags_fullset;
3074     secflags_intersection;
3075     secflags_isempty;
3076     secflag_isset;
3077     secflags_issubset;
3078     secflags_issuperset;
3079     secflag_set;
3080     secflag_to_bit;
3081     secflag_to_str;
3082     secflags_union;
3083     psecflags_validate_delta;
3084     secflags_zero;
3085     psecflags_default;
3086     secflags_parse;
3087     secflags_to_str;
3088     psecflags_validate;
3089 #endif /* ! codereview */
3090     semctl164;
3091     _semctl164;
3092     set_setcontext_enforcement;
3093     _setbufend;
3094     __set_errno;
3095     setprojctl;
3096     _setregid;
3097     _setreuid;
3098     setsigacthandler;
3099     shmctl164;
3100     _shmctl164;
3101     sigflag;
3102     _signal;
3103     _sigoff;
3104     _sigon;
3105     _so_accept;
3106     _so_bind;
3107     _sockconfig;
3108     _so_connect;
3109     _so_getpeername;
3110     _so_getsockname;

```

```

3111     _so_getsockopt;
3112     _so_listen;
3113     _so_recv;
3114     _so_recvfrom;
3115     _so_recvmsg;
3116     _so_send;
3117     _so_sendmsg;
3118     _so_sendto;
3119     _so_setsockopt;
3120     _so_shutdown;
3121     _so_socket;
3122     _so_socketpair;
3123     str2group;
3124     str2passwd;
3125     str2spwd;
3126     __strptime_dontzero;
3127     stty;
3128     syscall;
3129     _sysconfig;
3130     __systemcall;
3131     thr_continue_allmutators;
3132     _thr_continue_allmutators;
3133     thr_continue_mutator;
3134     _thr_continue_mutator;
3135     thr_getstate;
3136     _thr_getstate;
3137     thr_mutators_barrier;
3138     _thr_mutators_barrier;
3139     thr_probe_setup;
3140     _thr_schedctl;
3141     thr_setmutator;
3142     _thr_setmutator;
3143     thr_setstate;
3144     _thr_setstate;
3145     thr_sighndlrinfo;
3146     _thr_sighndlrinfo;
3147     _thr_slot_offset;
3148     thr_suspend_allmutators;
3149     _thr_suspend_allmutators;
3150     thr_suspend_mutator;
3151     _thr_suspend_mutator;
3152     thr_wait_mutator;
3153     _thr_wait_mutator;
3154     __tls_get_addr;
3155     _tmem_get_base;
3156     _tmem_get_nentries;
3157     _tmem_set_cleanup;
3158     tpool_create;
3159     tpool_dispatch;
3160     tpool_destroy;
3161     tpool_wait;
3162     tpool_suspend;
3163     tpool_suspended;
3164     tpool_resume;
3165     tpool_member;
3166     _ttypename_dev;
3167     _ucred_alloc;
3168     ucred_getamask;
3169     _ucred_getamask;
3170     ucred_getasid;
3171     _ucred_getasid;
3172     ucred_getatid;
3173     _ucred_getatid;
3174     ucred_getauid;
3175     _ucred_getauid;
3176     _ulltostr;

```

```

3177     _uncached_getgrgid_r;
3178     _uncached_getgrnam_r;
3179     _uncached_getpwnam_r;
3180     _uncached_getpwuid_r;
3181     __ungetwc_xpg5;
3182     _unordered;
3183     utssys;
3184     _verrpf;
3185     _verrxfp;
3186     _vwarnfp;
3187     _vwarnxfp;
3188     _warnfp;
3189     _warnxfp;
3190     __wcsftime_xpg5;
3191     __wcstok_xpg5;
3192     wdbindf;
3193     wdchkind;
3194     wddelim;
3195     _wrtchk;
3196     _xflsbuf;
3197     _xgetwidth;
3198     zone_add_datalink;
3199     zone_boot;
3200     zone_check_datalink;
3201     zone_create;
3202     zone_destroy;
3203     zone_enter;
3204     zone_getattr;
3205     zone_get_id;
3206     zone_list;
3207     zone_list_datalink;
3208     zonept;
3209     zone_remove_datalink;
3210     zone_setattr;
3211     zone_shutdown;
3212     zone_version;

3214 $if _ELF32
3215     __divdi3;
3216     _file_set;
3217     _fprintf_c89;
3218     _fscanf_c89;
3219     _fwprintf_c89;
3220     _fwscanf_c89;
3221     _imaxabs_c89;
3222     _imaxdiv_c89;
3223     __moddi3;
3224     _printf_c89;
3225     _scanf_c89;
3226     _snprintf_c89;
3227     _sprintf_c89;
3228     _sscanf_c89;
3229     _strtoimax_c89;
3230     _strtoumax_c89;
3231     _swprintf_c89;
3232     _swscanf_c89;
3233     __uidivi3;
3234     _umoddi3;
3235     _vfprintf_c89;
3236     _vfscanf_c89;
3237     _vfwprintf_c89;
3238     _vfwscanf_c89;
3239     _vprintf_c89;
3240     _vscanf_c89;
3241     _vsnprintf_c89;
3242     _vsprintf_c89;

```

```

3243     _vsscanf_c89;
3244     _vswprintf_c89;
3245     _vswscanf_c89;
3246     _vwprintf_c89;
3247     _vwscanf_c89;
3248     _wcstoimax_c89;
3249     _wcstoumax_c89;
3250     _wprintf_c89;
3251     _wscanf_c89;
3252 $endif

3254 $if _sparc
3255     _cerror;
3256     install_utrap;
3257     _install_utrap;
3258     nop;
3259     _Q_cplx_div;
3260     _Q_cplx_div_ix;
3261     _Q_cplx_div_rx;
3262     _Q_cplx_lr_div;
3263     _Q_cplx_lr_div_ix;
3264     _Q_cplx_lr_div_rx;
3265     _Q_cplx_lr_mul;
3266     _Q_cplx_mul;
3267     _QgetRD;
3268     _xregs_clrptr;
3269 $endif

3271 $if sparc32
3272     __ashldi3;
3273     __ashrdi3;
3274     _cerror64;
3275     __cmpdi2;
3276     __floatdidf;
3277     __floatdisf;
3278     __floatundidf;
3279     __floatundisf;
3280     __lshrdi3;
3281     __muldi3;
3282     __ucmpdi2;
3283 $endif

3285 $if _x86
3286     _D_cplx_lr_div;
3287     _D_cplx_lr_div_ix;
3288     _D_cplx_lr_div_rx;
3289     _F_cplx_lr_div;
3290     _F_cplx_lr_div_ix;
3291     _F_cplx_lr_div_rx;
3292     _fltrounds;
3293     sysi86;
3294     _sysi86;
3295     _X_cplx_div;
3296     _X_cplx_div_ix;
3297     _X_cplx_div_rx;
3298     _X_cplx_lr_div;
3299     _X_cplx_lr_div_ix;
3300     _X_cplx_lr_div_rx;
3301     _X_cplx_mul;
3302     _xgetRD;
3303     _xtol;
3304     _xtoll;
3305     _xtoul;
3306     _xtoull;
3307 $endif

```

```

3309 $if i386
3310     __divrem64;
3311     __tls_get_addr;
3312     __udivrem64;
3313 $endif

3315 # The following functions should not be exported from libc,
3316 # but /lib/libm.so.2, some older versions of the Studio
3317 # compiler/debugger components, and some ancient programs
3318 # found in /usr/dist reference them. When we no longer
3319 # care about these old and broken binary objects, these
3320 # symbols should be deleted.
3321     _brk                { FLAGS = NODYNSORT };
3322     _cond_broadcast     { FLAGS = NODYNSORT };
3323     _cond_init          { FLAGS = NODYNSORT };
3324     _cond_signal        { FLAGS = NODYNSORT };
3325     _cond_wait          { FLAGS = NODYNSORT };
3326     _ecvt               { FLAGS = NODYNSORT };
3327     _fcvt               { FLAGS = NODYNSORT };
3328     _getc_unlocked      { FLAGS = NODYNSORT };
3329     _llseek             { FLAGS = NODYNSORT };
3330     _pthread_attr_getdetachstate { FLAGS = NODYNSORT };
3331     _pthread_attr_getinheritsched { FLAGS = NODYNSORT };
3332     _pthread_attr_getschedparam { FLAGS = NODYNSORT };
3333     _pthread_attr_getschedpolicy { FLAGS = NODYNSORT };
3334     _pthread_attr_getscope { FLAGS = NODYNSORT };
3335     _pthread_attr_getstackaddr { FLAGS = NODYNSORT };
3336     _pthread_attr_getstacksize { FLAGS = NODYNSORT };
3337     _pthread_attr_init  { FLAGS = NODYNSORT };
3338     _pthread_condattr_getpshared { FLAGS = NODYNSORT };
3339     _pthread_condattr_init { FLAGS = NODYNSORT };
3340     _pthread_cond_init  { FLAGS = NODYNSORT };
3341     _pthread_create     { FLAGS = NODYNSORT };
3342     _pthread_getschedparam { FLAGS = NODYNSORT };
3343     _pthread_join       { FLAGS = NODYNSORT };
3344     _pthread_key_create { FLAGS = NODYNSORT };
3345     _pthread_mutexattr_getprioceiling { FLAGS = NODYNSORT };
3346     _pthread_mutexattr_getprotocol { FLAGS = NODYNSORT };
3347     _pthread_mutexattr_getpshared { FLAGS = NODYNSORT };
3348     _pthread_mutexattr_init { FLAGS = NODYNSORT };
3349     _pthread_mutex_getprioceiling { FLAGS = NODYNSORT };
3350     _pthread_mutex_init { FLAGS = NODYNSORT };
3351     _pthread_sigmask    { FLAGS = NODYNSORT };
3352     _rwlock_init        { FLAGS = NODYNSORT };
3353     _rw_rdlock          { FLAGS = NODYNSORT };
3354     _rw_unlock          { FLAGS = NODYNSORT };
3355     _rw_wrlck           { FLAGS = NODYNSORT };
3356     _sbrk_unlocked     { FLAGS = NODYNSORT };
3357     _select             { FLAGS = NODYNSORT };
3358     _sema_init          { FLAGS = NODYNSORT };
3359     _sema_post          { FLAGS = NODYNSORT };
3360     _sema_trywait      { FLAGS = NODYNSORT };
3361     _sema_wait          { FLAGS = NODYNSORT };
3362     _sysfs              { FLAGS = NODYNSORT };
3363     _thr_create         { FLAGS = NODYNSORT };
3364     _thr_exit           { FLAGS = NODYNSORT };
3365     _thr_getprio        { FLAGS = NODYNSORT };
3366     _thr_getspecific    { FLAGS = NODYNSORT };
3367     _thr_join           { FLAGS = NODYNSORT };
3368     _thr_keycreate      { FLAGS = NODYNSORT };
3369     _thr_kill           { FLAGS = NODYNSORT };
3370     _thr_main           { FLAGS = NODYNSORT };
3371     _thr_self           { FLAGS = NODYNSORT };
3372     _thr_setspecific    { FLAGS = NODYNSORT };
3373     _thr_sigsetmask     { FLAGS = NODYNSORT };
3374     _thr_stksegment     { FLAGS = NODYNSORT };

```

```

3375     _ungetc_unlocked   { FLAGS = NODYNSORT };

3377     local:
3378         __imax_lldiv    { FLAGS = NODYNSORT };
3379         _ti_thr_self    { FLAGS = NODYNSORT };
3380         *;

3382 $if lf64
3383     __seekdir64        { FLAGS = NODYNSORT };
3384     _telldir64         { FLAGS = NODYNSORT };
3385 $endif

3387 $if _sparc
3388     __cerror           { FLAGS = NODYNSORT };
3389 $endif

3391 $if sparc32
3392     __cerror64         { FLAGS = NODYNSORT };
3393 $endif

3395 $if sparcv9
3396     __cleanup          { FLAGS = NODYNSORT };
3397 $endif

3399 $if i386
3400     __syscall6         { FLAGS = NODYNSORT };
3401     __systemcall6     { FLAGS = NODYNSORT };
3402 $endif

3404 $if amd64
3405     __tls_get_addr     { FLAGS = NODYNSORT };
3406 $endif
};

```

new/usr/src/lib/libc/port/sys/sbrk.c

1

```
*****
4697 Thu Jun 30 22:02:13 2016
new/usr/src/lib/libc/port/sys/sbrk.c
Code review comments from jeffpc
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */

22 /*
23  * Copyright 2008 Sun Microsystems, Inc. All rights reserved.
24  * Use is subject to license terms.
25 */

27 #pragma ident      "%Z%M% %I%      %E% SMI"

27 #pragma weak _sbrk = sbrk
28 #pragma weak _brk = brk

30 #include "lint.h"
31 #include <synch.h>
32 #include <errno.h>
33 #include <sys/isa_defs.h>
34 #include <sys/types.h>
35 #include <sys/sysmacros.h>
36 #include <inttypes.h>
37 #include <unistd.h>
38 #include "mtlib.h"
39 #include "libc.h"

41 void * _nd = NULL;
43 extern int _end;
44 void * _nd = &_end;
42 mutex_t __sbrk_lock = DEFAULTMUTEX;

44 extern intptr_t _brk_unlocked(void *);
45 void * _sbrk_unlocked(intptr_t);
47 extern int _brk_unlocked(void *);
48 extern void * _sbrk_unlocked(intptr_t);

47 /*
48  * The break must always be at least 8-byte aligned
49  */
50 #if (_MAX_ALIGNMENT < 8)
51 #define ALIGNSZ      8
```

new/usr/src/lib/libc/port/sys/sbrk.c

2

```
52 #else
53 #define ALIGNSZ      _MAX_ALIGNMENT
54 #endif

56 #define BRKALIGN(x)      (caddr_t)P2ROUNDUP((uintptr_t)(x), ALIGNSZ)

58 void *
59 sbrk(intptr_t addend)
60 {
61     void *result;

63     if (!primary_link_map) {
64         errno = ENOTSUP;
65         return ((void *)-1);
66     }
67     lmutex_lock(&__sbrk_lock);
68     result = _sbrk_unlocked(addend);
69     lmutex_unlock(&__sbrk_lock);

71     return (result);
72 }

74 /*
75  * _sbrk_unlocked() aligns the old break, adds the addend, aligns
76  * the new break, and calls _brk_unlocked() to set the new break.
77  * We must align the old break because _nd may begin life misaligned.
78  * The addend can be either positive or negative, so there are two
79  * overflow/underflow edge conditions to reject:
80  *
81  * - the addend is negative and brk + addend < 0.
82  * - the addend is positive and brk + addend > ULONG_MAX
83  */
84 void *
85 _sbrk_unlocked(intptr_t addend)
86 {
87     char *old_brk;
88     char *new_brk;

90     if (_nd == NULL) {
91         _nd = (void *)_brk_unlocked(0);
92     }

94     old_brk = BRKALIGN(_nd);
95     new_brk = BRKALIGN(old_brk + addend);
96     char *old_brk = BRKALIGN(_nd);
97     char *new_brk = BRKALIGN(old_brk + addend);

99     if ((addend > 0 && new_brk < old_brk) ||
100         (addend < 0 && new_brk > old_brk)) {
101         errno = ENOMEM;
102         return ((void *)-1);
103     }
104     if (_brk_unlocked(new_brk) != 0)
105         return ((void *)-1);
106     _nd = new_brk;
107     return (old_brk);
108 }

108 /*
109  * _sbrk_grow_aligned() aligns the old break to a low_align boundry,
110  * adds min_size, aligns to a high_align boundry, and calls _brk_unlocked()
111  * to set the new break. The low_align-aligned value is returned, and
112  * the actual space allocated is returned through actual_size.
113  *
114  * Unlike sbrk(2), _sbrk_grow_aligned takes an unsigned size, and does
115  * not allow shrinking the heap.
```

```

116 */
117 void *
118 _sbrk_grow_aligned(size_t min_size, size_t low_align, size_t high_align,
119                 size_t *actual_size)
120 {
121     uintptr_t old_brk;
122     uintptr_t ret_brk;
123     uintptr_t high_brk;
124     uintptr_t new_brk;
125     intptr_t brk_result;
126     int brk_result;
127
128     if (!primary_link_map) {
129         errno = ENOTSUP;
130         return ((void *)-1);
131     }
132     if ((low_align & (low_align - 1)) != 0 ||
133         (high_align & (high_align - 1)) != 0) {
134         errno = EINVAL;
135         return ((void *)-1);
136     }
137     low_align = MAX(low_align, ALIGNSZ);
138     high_align = MAX(high_align, ALIGNSZ);
139
140     lmutex_lock(&__sbrk_lock);
141
142     if (_nd == NULL)
143         _nd = (void *)_brk_unlocked(0);
144
145 #endif /* ! codereview */
146     old_brk = (uintptr_t)BRKALIGN(_nd);
147     ret_brk = P2ROUNDUP(old_brk, low_align);
148     high_brk = ret_brk + min_size;
149     new_brk = P2ROUNDUP(high_brk, high_align);
150
151     /*
152      * Check for overflow
153      */
154     if (ret_brk < old_brk || high_brk < ret_brk || new_brk < high_brk) {
155         lmutex_unlock(&__sbrk_lock);
156         errno = ENOMEM;
157         return ((void *)-1);
158     }
159
160     if ((brk_result = _brk_unlocked((void *)new_brk)) == 0)
161         _nd = (void *)new_brk;
162     lmutex_unlock(&__sbrk_lock);
163
164     if (brk_result != 0)
165         return ((void *)-1);
166
167     if (actual_size != NULL)
168         *actual_size = (new_brk - ret_brk);
169     return ((void *)ret_brk);
170 }
171
172 int
173 brk(void *new_brk)
174 {
175     intptr_t result;
176
177     /*
178      * brk(2) will return the current brk if given an argument of 0, so we
179      * need to fail it here
180      */
181     if (new_brk == 0) {

```

```

181         errno = ENOMEM;
182         return (-1);
183     }
184     return result;
185
186     if (!primary_link_map) {
187         errno = ENOTSUP;
188         return (-1);
189     }
190     /*
191      * Need to align this here; _brk_unlocked won't do it for us.
192      */
193     new_brk = BRKALIGN(new_brk);
194
195     lmutex_lock(&__sbrk_lock);
196     if ((result = _brk_unlocked(new_brk)) == 0)
197         _nd = new_brk;
198     lmutex_unlock(&__sbrk_lock);
199
200     return (result);
201 }
202
203 _____unchanged_portion_omitted_____

```

new/usr/src/lib/libc/req.flg

1

1103 Thu Jun 30 22:02:14 2016

new/usr/src/lib/libc/req.flg

7029 want per-process exploit mitigation features (secflags)

7030 want basic address space layout randomization (aslr)

7031 noexec_user_stack should be a secflag

7032 want a means to forbid mappings around NULL.

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

27 find_files "s.*" usr/src/common/atomic
28 find_files "s.*" usr/src/common/dtrace
29 find_files "s.*" usr/src/common/secflags
30 #endif /* !codereview */
31 find_files "s.*" usr/src/common/util
29 find_files "s.*" usr/src/common/dtrace
32 find_files "s.*" usr/src/lib/common
```

```

*****
26493 Thu Jun 30 22:02:14 2016
new/usr/src/lib/libc/sparc/Makefile.com
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
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 (c) 1989, 2010, Oracle and/or its affiliates. All rights reserved.
23 # Copyright 2016 Joyent, Inc.
24 # Copyright (c) 2013, OmniTI Computer Consulting, Inc. All rights reserved.
25 # Copyright 2011 Garrett D'Amore <garrett@damore.org>
26 #
27 # Copyright 2011 Nexenta Systems, Inc. All rights reserved.
28 # Use is subject to license terms.
29 #

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

37 # objects are grouped by source directory

39 # Symbol capabilities objects.
40 EXTPICS=     \
41             $(LIBCDIR)/capabilities/sun4u/sparc/pics/symcap.o \
42             $(LIBCDIR)/capabilities/sun4u-opl/sparc/pics/symcap.o \
43             $(LIBCDIR)/capabilities/sun4u-us3-hwcap1/sparc/pics/symcap.o \
44             $(LIBCDIR)/capabilities/sun4u-us3-hwcap2/sparc/pics/symcap.o \
45             $(LIBCDIR)/capabilities/sun4v-hwcap1/sparc/pics/symcap.o \
46             $(LIBCDIR)/capabilities/sun4v-hwcap2/sparc/pics/symcap.o

48 # local objects
49 STRETS=     \
50             stret1.o \
51             stret2.o \
52             stret4.o

54 CRTOBS=     \
55             _ftou.o \
56             cerror.o \
57             cerror64.o \
58             hwmuldiv.o

```

```

60 DYNOBJS=    \
61             _rtbootld.o

63 FPOBJS=     \
64             _D_cplx_div.o \
65             _D_cplx_div_ix.o \
66             _D_cplx_div_rx.o \
67             _D_cplx_mul.o \
68             _F_cplx_div.o \
69             _F_cplx_div_ix.o \
70             _F_cplx_div_rx.o \
71             _F_cplx_mul.o \
72             _Q_add.o \
73             _Q_cmp.o \
74             _Q_cmpe.o \
75             _Q_cplx_div.o \
76             _Q_cplx_div_ix.o \
77             _Q_cplx_div_rx.o \
78             _Q_cplx_lr_div.o \
79             _Q_cplx_lr_div_ix.o \
80             _Q_cplx_lr_div_rx.o \
81             _Q_cplx_lr_mul.o \
82             _Q_cplx_mul.o \
83             _Q_div.o \
84             _Q_dtoq.o \
85             _Q_fcc.o \
86             _Q_itoq.o \
87             _Q_lltoq.o \
88             _Q_mul.o \
89             _Q_neg.o \
90             _Q_qtod.o \
91             _Q_qtoi.o \
92             _Q_qtos.o \
93             _Q_qtou.o \
94             _Q_scl.o \
95             _Q_set_except.o \
96             _Q_sqrt.o \
97             _Q_stoq.o \
98             _Q_sub.o \
99             _Q_ulltoq.o \
100            _Q_utoq.o \
101            __quad_mag.o

103 FPASMOBJS= \
104             _Q_get_rp_rd.o \
105             fpgetmask.o \
106             fpgetrnd.o \
107             fpgetsticky.o \
108             fpsetmask.o \
109             fpsetrnd.o \
110             fpsetsticky.o

112 $(__GNUCC)FPASMOBJS += \
113             __quad.o

115 ATOMICOBJS= \
116             atomic.o

118 CHACHAOBJS= \
119             chacha.o

121 XATTROBJS= \
122             xattr_common.o

124 COMOBJS=   \

```



```

125      bcmp.o          \
126      bcopy.o        \
127      bzero.o        \
128      bsearch.o      \
129      memccpy.o      \
130      qsort.o        \
131      strtol.o       \
132      strtoul.o      \
133      strtoll.o      \
134      strtoull.o     \

136 DTRACEOBS=       \
137      dtrace_data.o \

139 SECFLAGSOBJ=     \
140      secflags.o     \

142 #endif /* ! codereview */
143 GENOBS=           \
144      _getsp.o       \
145      _xregs_clrptr.o \
146      abs.o          \
147      alloca.o       \
148      arc4random.o   \
149      arc4random_uniform.o \
150      ascii_strcasecmp.o \
151      byteorder.o    \
152      cuexit.o       \
153      ecvt.o         \
154      endian.o       \
155      errlst.o       \
156      getctxt.o      \
157      ladd.o         \
158      lmul.o         \
159      lock.o         \
160      lshiftl.o      \
161      lsign.o        \
162      lsub.o         \
163      makectxt.o     \
164      memchr.o       \
165      memcmp.o       \
166      new_list.o     \
167      setjmp.o       \
168      siginfolst.o   \
169      siglongjmp.o   \
170      smt_pause.o    \
171      sparc_data.o   \
172      strchr.o       \
173      strcmp.o       \
174      strlcpy.o      \
175      strncmp.o      \
176      strncpy.o      \
177      strlen.o       \
178      swapctxt.o    \
179      sync_instruction_memory.o \

181 # sysobjs that contain large-file interfaces
182 COMSYSOBS64=     \
183      fstatvfs64.o  \
184      getdents64.o  \
185      getrlimit64.o \
186      lseek64.o     \
187      mmap64.o      \
188      pread64.o     \
189      preadv64.o    \
190      pwrite64.o    \

```

```

191      pwrite64.o    \
192      setrlimit64.o \
193      statvfs64.o  \

195 SYSOBS64=

197 COMSYSOBS=       \
198      __clock_timer.o \
199      __getloadavg.o  \
200      __rusagesys.o  \
201      __signotify.o  \
202      __sigrt.o      \
203      __time.o       \
204      __lgrp_home_fast.o \
205      __lgrpsys.o    \
206      __nfssys.o     \
207      __portfs.o     \
208      __pset.o       \
209      __rpcsys.o     \
210      __sigaction.o  \
211      __so_accept.o  \
212      __so_bind.o    \
213      __so_connect.o \
214      __so_getpeername.o \
215      __so_getsockname.o \
216      __so_getsockopt.o \
217      __so_listen.o  \
218      __so_recv.o    \
219      __so_recvfrom.o \
220      __so_recvmsg.o \
221      __so_send.o    \
222      __so_sendmsg.o \
223      __so_sendto.o  \
224      __so_setsockopt.o \
225      __so_shutdown.o \
226      __so_socket.o  \
227      __so_socketpair.o \
228      __sockconfig.o \
229      acct.o         \
230      acl.o          \
231      adjtime.o      \
232      alarm.o        \
233      brk.o          \
234      chdir.o        \
235      chroot.o       \
236      cladm.o        \
237      close.o        \
238      execve.o       \
239      exit.o         \
240      facd.o         \
241      fchdir.o       \
242      fchroot.o      \
243      fdsync.o       \
244      fpathconf.o    \
245      fstatfs.o      \
246      fstatvfs.o     \
247      getcpuid.o     \
248      getdents.o     \
249      getegid.o      \
250      geteuid.o      \
251      getgid.o       \
252      getgroups.o    \
253      gethrtime.o    \
254      getitimer.o    \
255      getmsg.o       \
256      getpid.o       \

```

```

257 getpmsg.o \
258 getppid.o \
259 getrandom.o \
260 getrlimit.o \
261 getuid.o \
262 gtty.o \
263 install_utrap.o \
264 ioctl.o \
265 kaio.o \
266 kill.o \
267 llseek.o \
268 lseek.o \
269 memcntl.o \
270 mincore.o \
271 mmap.o \
272 mmapobjsys.o \
273 modctl.o \
274 mount.o \
275 mprotect.o \
276 munmap.o \
277 nice.o \
278 ntp_adjtime.o \
279 ntp_gettime.o \
280 p_online.o \
281 pathconf.o \
282 pause.o \
283 pcsample.o \
284 pipe2.o \
285 pollsys.o \
286 pread.o \
287 preadv.o \
288 prioctlset.o \
289 processor_bind.o \
290 processor_info.o \
291 profil.o \
292 psecflagsset.o \
293 #endif /* ! codereview */
294 putmsg.o \
295 putpmsg.o \
296 pwrite.o \
297 pwritev.o \
298 read.o \
299 readv.o \
300 resolvepath.o \
301 seteguid.o \
302 setgid.o \
303 setgroups.o \
304 setitimer.o \
305 setreid.o \
306 setrlimit.o \
307 setuid.o \
308 sigaltstk.o \
309 sigprocmsk.o \
310 sigsendset.o \
311 sigsuspend.o \
312 statfs.o \
313 statvfs.o \
314 stty.o \
315 sync.o \
316 sysconfig.o \
317 sysfs.o \
318 sysinfo.o \
319 syslwp.o \
320 times.o \
321 ulimit.o \
322 umask.o \

```

```

323 umount2.o \
324 utssys.o \
325 uucopy.o \
326 vhangup.o \
327 waitid.o \
328 write.o \
329 writev.o \
330 yield.o \
332 SYSOBSJ= \
333 __clock_gettime.o \
334 __getcontext.o \
335 __lwp_mutex_unlock.o \
336 __stack_grow.o \
337 __uadmin.o \
338 door.o \
339 forkx.o \
340 forkallx.o \
341 gettimeofday.o \
342 ptrace.o \
343 syscall.o \
344 tls_get_addr.o \
345 uadmin.o \
346 umount.o \
347 uname.o \
348 vforkx.o \
350 # objects under $(LIBCDIR)/port which contain transitional large file interfaces
351 PORTGEN64= \
352 __xftw64.o \
353 attropen64.o \
354 ftw64.o \
355 mkstemp64.o \
356 nftw64.o \
357 tell64.o \
358 truncate64.o \
360 # objects from source under $(LIBCDIR)/port
361 PORTFP= \
362 __flt_decim.o \
363 __flt_rounds.o \
364 __tbl_10_b.o \
365 __tbl_10_h.o \
366 __tbl_10_s.o \
367 __tbl_2_b.o \
368 __tbl_2_h.o \
369 __tbl_2_s.o \
370 __tbl_fdq.o \
371 __tbl_tens.o \
372 __x_power.o \
373 __base_sup.o \
374 aconvert.o \
375 decimal_bin.o \
376 double_decim.o \
377 econvert.o \
378 fconvert.o \
379 file_decim.o \
380 finite.o \
381 fp_data.o \
382 func_decim.o \
383 gconvert.o \
384 hex_bin.o \
385 ieee_globals.o \
386 pack_float.o \
387 sigfpe.o \
388 string_decim.o \

```

```

389      ashldi3.o      \
390      ashrdi3.o      \
391      cmpdi2.o       \
392      divdi3.o       \
393      floatdidf.o   \
394      floatdisf.o   \
395      floatundidf.o \
396      floatundisf.o \
397      lshr3.o        \
398      moddi3.o       \
399      muldi3.o       \
400      qdivrem.o      \
401      ucmpdi2.o     \
402      udivdi3.o     \
403      umoddi3.o     \
404
405  PORTGEN=          \
406      _env_data.o   \
407      _ftoll.o      \
408      _ftoull.o     \
409      _xftw.o       \
410      a64l.o        \
411      abort.o       \
412      addsev.o      \
413      ascii_strncasecmp.o \
414      assert.o      \
415      atof.o        \
416      atoi.o        \
417      atol.o        \
418      atoll.o       \
419      attrat.o      \
420      attropen.o    \
421      atexit.o      \
422      atfork.o      \
423      basename.o    \
424      calloc.o      \
425      catgets.o     \
426      catopen.o     \
427      cfgetispeed.o \
428      cfgetospeed.o \
429      cfree.o       \
430      cfsetispeed.o \
431      cfsetospeed.o \
432      cftime.o      \
433      clock.o       \
434      closedir.o    \
435      closefrom.o   \
436      confstr.o     \
437      crypt.o       \
438      csetlen.o     \
439      ctime.o       \
440      ctime_r.o     \
441      daemon.o     \
442      deflt.o       \
443      directio.o    \
444      dirname.o     \
445      div.o         \
446      drand48.o    \
447      dup.o         \
448      env_data.o    \
449      err.o         \
450      errno.o       \
451      euclen.o      \
452      event_port.o  \
453      execvp.o      \
454      explicit_bzero.o \

```

```

455      fattach.o     \
456      fdetach.o     \
457      fdopendir.o   \
458      ffs.o         \
459      flock.o       \
460      fls.o         \
461      fmtmsg.o      \
462      ftime.o       \
463      ftok.o        \
464      ftw.o         \
465      gcvt.o        \
466      getauxv.o     \
467      getcwd.o      \
468      getdate_err.o \
469      getdtblsize.o \
470      getentropy.o  \
471      getenv.o      \
472      getexecname.o \
473      getgrnam.o    \
474      getgrnam_r.o  \
475      gethostid.o   \
476      gethostname.o \
477      gethz.o       \
478      getisax.o     \
479      getloadavg.o  \
480      getlogin.o    \
481      getmntent.o   \
482      getnetgrent.o \
483      get_nprocs.o  \
484      getopt.o      \
485      getopt_long.o \
486      getpagesize.o \
487      getpw.o       \
488      getpwnam.o    \
489      getpwnam_r.o  \
490      getrusage.o   \
491      getspent.o    \
492      getspent_r.o  \
493      getsubopt.o   \
494      gettxt.o      \
495      getusershell.o \
496      getut.o       \
497      getutx.o      \
498      getvfsent.o   \
499      getwd.o       \
500      getwidth.o    \
501      getxby_door.o \
502      gtxt.o        \
503      hsearch.o     \
504      iconv.o       \
505      imaxabs.o     \
506      imaxdiv.o     \
507      index.o       \
508      initgroups.o  \
509      insque.o       \
510      isaexec.o     \
511      isastream.o   \
512      isatty.o      \
513      killpg.o      \
514      klpdlib.o     \
515      l64a.o        \
516      lckpwwd.o     \
517      lconstants.o \
518      ldivide.o     \
519      lexp10.o      \
520      lfind.o       \

```

```

521     lfmt.o           \
522     lfmt_log.o      \
523     llabs.o         \
524     lldiv.o         \
525     llog10.o        \
526     lltostr.o       \
527     localtime.o    \
528     lsearch.o       \
529     madvise.o       \
530     malloc.o        \
531     memalign.o      \
532     memmem.o        \
533     mkdev.o         \
534     mkdtemp.o       \
535     mkfifo.o        \
536     mkstemp.o       \
537     mktemp.o        \
538     mlock.o         \
539     mlockall.o     \
540     mon.o           \
541     msync.o         \
542     munlock.o       \
543     munlockall.o   \
544     ndbm.o          \
545     nftw.o          \
546     nlspath_checks.o \
547     nsparse.o       \
548     nss_common.o    \
549     nss_dbdefs.o    \
550     nss_deffinder.o \
551     opendir.o       \
552     opt_data.o      \
553     perror.o        \
554     pfmt.o          \
555     pfmt_data.o     \
556     pfmt_print.o    \
557     pipe.o          \
558     plock.o         \
559     poll.o          \
560     posix_fadvise.o \
561     posix_fallocate.o \
562     posix_madvise.o \
563     posix_memalign.o \
564     priocntl.o      \
565     privlib.o       \
566     priv_str_xlate.o \
567     psecflags.o     \
568 #endif /* ! codereview */
569     psiginfo.o       \
570     psignal.o        \
571     pt.o             \
572     putpwent.o       \
573     putspent.o       \
574     raise.o          \
575     rand.o           \
576     random.o         \
577     rctlops.o        \
578     readdir.o        \
579     readdir_r.o     \
580     realpath.o      \
581     reboot.o        \
582     regexpr.o        \
583     remove.o         \
584     rewinddir.o     \
585     rindex.o         \
586     scandir.o       \

```

```

587     seekdir.o       \
588     select.o        \
589     select_large_fdset.o \
590     setlabel.o      \
591     setpriority.o   \
592     settimeofday.o  \
593     sh_locks.o      \
594     sigflag.o       \
595     siglist.o       \
596     sigsend.o       \
597     sigsetops.o     \
598     signal.o        \
599     stack.o         \
600     stpcpy.o        \
601     stpncpy.o       \
602     str2sig.o       \
603     strcase_ormap.o \
604     strcat.o        \
605     strchrnul.o     \
606     strcspn.o       \
607     strdup.o        \
608     strerror.o      \
609     strlcat.o       \
610     strncat.o       \
611     strndup.o       \
612     strpbrk.o       \
613     strrchr.o       \
614     strsep.o        \
615     strsignal.o     \
616     strspn.o        \
617     strstr.o        \
618     strtod.o        \
619     strtointmax.o   \
620     strtok.o        \
621     strtok_r.o      \
622     strtoumax.o    \
623     swab.o          \
624     swapctl.o       \
625     sysconf.o       \
626     syslog.o        \
627     tcdrain.o       \
628     tcflow.o        \
629     tcflush.o       \
630     tcgetattr.o     \
631     tcgetpgrp.o     \
632     tcgetsid.o      \
633     tcsetattr.o     \
634     tcsetpgrp.o     \
635     tell.o          \
636     telldir.o       \
637     tfind.o         \
638     time_data.o     \
639     time_gdata.o    \
640     timespec_get.o  \
641     tls_data.o      \
642     truncate.o      \
643     tsdalloc.o      \
644     tsearch.o       \
645     ttyname.o       \
646     ttyslot.o       \
647     ualarm.o        \
648     ucred.o         \
649     valloc.o        \
650     vlfmt.o         \
651     vpfmt.o         \
652

```

```

653      waitpid.o      \
654      walkstack.o   \
655      wdata.o        \
656      xgetwidth.o   \
657      xpg4.o         \
658      xpg6.o         \

660 PORTPRINT_W=      \
661      doprnt_w.o    \

663 PORTPRINT=        \
664      asprintf.o    \
665      doprnt.o      \
666      fprintf.o     \
667      printf.o      \
668      snprintf.o    \
669      sprintf.o     \
670      vfprintf.o    \
671      vprintf.o     \
672      vsnprintf.o   \
673      vsprintf.o    \
674      vwprintf.o    \
675      wprintf.o     \

677 # c89 variants to support 32-bit size of c89 u/intmax_t (32-bit libc only)
678 PORTPRINT_C89=    \
679      vfprintf_c89.o \
680      vprintf_c89.o  \
681      vsnprintf_c89.o \
682      vsprintf_c89.o \
683      vwprintf_c89.o \

685 PORTSTDIO_C89=   \
686      vscanf_c89.o  \
687      vwscanf_c89.o \

689 # portable stdio objects that contain large file interfaces.
690 # Note: fopen64 is a special case, as we build it small.
691 PORTSTDIO64=     \
692      fopen64.o     \
693      fpos64.o      \

695 PORTSTDIO_W=     \
696      doscan_w.o    \

698 PORTSTDIO=       \
699      __extensions.o \
700      _endopen.o    \
701      _filbuf.o     \
702      _findbuf.o    \
703      _flsbuf.o     \
704      _wrtchk.o     \
705      clearerr.o    \
706      ctermid.o     \
707      ctermid_r.o   \
708      cuserid.o     \
709      data.o        \
710      doscan.o      \
711      fdopen.o      \
712      feof.o        \
713      ferrord.o     \
714      fgetc.o       \
715      fgets.o       \
716      fileno.o      \
717      flockf.o      \
718      flush.o       \

```

```

719      fopen.o       \
720      fpos.o        \
721      fputc.o       \
722      fputs.o       \
723      fread.o       \
724      fseek.o       \
725      fseeko.o      \
726      ftell.o       \
727      ftello.o      \
728      fwrite.o      \
729      getc.o        \
730      getchar.o     \
731      getline.o     \
732      getpass.o     \
733      gets.o        \
734      getw.o        \
735      popen.o       \
736      putc.o        \
737      putchar.o     \
738      puts.o        \
739      putw.o        \
740      rewind.o      \
741      scanf.o       \
742      setbuf.o      \
743      setbuffer.o   \
744      setvbuf.o     \
745      system.o      \
746      tmpnam.o      \
747      tmpfile.o     \
748      tmpnam_r.o    \
749      ungetc.o      \
750      mse.o         \
751      vscanf.o      \
752      vwscanf.o     \
753      wscanf.o      \

755 PORTI18N=        \
756      getwchar.o    \
757      putwchar.o    \
758      putws.o       \
759      strtows.o     \
760      wcsnlen.o     \
761      wcstoimax.o   \
762      wcstol.o      \
763      wcstoul.o     \
764      wcswcs.o      \
765      wscat.o       \
766      wscr.o        \
767      wscmp.o       \
768      wscpy.o       \
769      wscspn.o      \
770      wsdup.o       \
771      wslen.o       \
772      wscat.o       \
773      wscmp.o       \
774      wscpy.o       \
775      wspbkr.o      \
776      wsprintf.o    \
777      wsrchr.o      \
778      wsscanf.o     \
779      wssp.o        \
780      wstod.o       \
781      wstok.o       \
782      wstol.o       \
783      wstoll.o      \
784      wsxfrm.o      \

```

```

785 wmemchr.o \
786 wmemcmp.o \
787 wmemcpy.o \
788 wmemmove.o \
789 wmemset.o \
790 wcstr.o \
791 gettext.o \
792 gettext_real.o \
793 gettext_util.o \
794 gettext_gnu.o \
795 plural_parser.o \
796 wdresolve.o \
797 _ctype.o \
798 isascii.o \
799 toascii.o

801 PORTI18N_COND= \
802 wcstol_longlong.o \
803 wcstoul_longlong.o

805 PORTLOCALE= \
806 big5.o \
807 btowc.o \
808 collate.o \
809 collcmp.o \
810 euc.o \
811 fnmatch.o \
812 fgetwc.o \
813 fgetws.o \
814 fix_grouping.o \
815 fputwc.o \
816 fputws.o \
817 fwide.o \
818 gb18030.o \
819 gb2312.o \
820 gbk.o \
821 getdate.o \
822 isdigit.o \
823 iswctype.o \
824 ldpart.o \
825 lmessages.o \
826 lnumeric.o \
827 lmonetary.o \
828 localeimpl.o \
829 localeconv.o \
830 mbftowc.o \
831 mblen.o \
832 mbrlen.o \
833 mbrtowc.o \
834 mbsinit.o \
835 mbsnrtowcs.o \
836 mbsrtowcs.o \
837 mbstowcs.o \
838 mbtowc.o \
839 mskanji.o \
840 nextwctype.o \
841 nl_langinfo.o \
842 none.o \
843 regcomp.o \
844 regfree.o \
845 regerror.o \
846 regexec.o \
847 rune.o \
848 runetype.o \
849 setlocale.o \
850 setrunelocale.o

```

```

851 strcasecmp.o \
852 strcasestr.o \
853 strcoll.o \
854 strfmon.o \
855 strftime.o \
856 strncasecmp.o \
857 strptime.o \
858 strxfrm.o \
859 table.o \
860 timelocal.o \
861 tolower.o \
862 towlower.o \
863 ungetwc.o \
864 utf8.o \
865 wctomb.o \
866 wcsasecmp.o \
867 wscoll.o \
868 wcsftime.o \
869 wcsnrtombs.o \
870 wcsrtombs.o \
871 wcstombs.o \
872 wcswidth.o \
873 wcsxfrm.o \
874 wctob.o \
875 wctomb.o \
876 wctrans.o \
877 wctype.o \
878 wcwidth.o \
879 wscol.o

881 AIOOBS= \
882 aio.o \
883 aio_alloc.o \
884 posix_aio.o

886 RTOBS= \
887 clock_timer.o \
888 mqueue.o \
889 pos4obj.o \
890 sched.o \
891 sem.o \
892 shm.o \
893 sigev_thread.o

895 TPOOLBS= \
896 thread_pool.o

898 THREADSOBS= \
899 alloc.o \
900 assfail.o \
901 cancel.o \
902 c11_thr.o \
903 door_calls.o \
904 tmem.o \
905 pthr_attr.o \
906 pthr_barrier.o \
907 pthr_cond.o \
908 pthr_mutex.o \
909 pthr_rwlock.o \
910 pthread.o \
911 rwlock.o \
912 scalls.o \
913 sema.o \
914 sigaction.o \
915 spawn.o \
916 synch.o

```

```

917     tdb_agent.o      \
918     thr.o            \
919     thread_interface.o \
920     tls.o            \
921     tsd.o            \

923 THREADSMACHOBJS=   \
924     machdep.o       \

926 THREADSASMOBJS=    \
927     asm_subr.o      \

929 UNICODOBJS=        \
930     u8_textprep.o   \
931     uconv.o         \

933 UNWINDMACHOBJS=    \
934     unwind.o        \

936 UNWINDASMOBJS=     \
937     unwind_frame.o \

939 # objects that implement the transitional large file API
940 PORTSYS64=          \
941     lockf64.o       \
942     stat64.o        \

944 PORTSYS=           \
945     _autofssys.o    \
946     access.o        \
947     acctctl.o       \
948    bsd_signal.o     \
949     chmod.o         \
950     chown.o         \
951     corectl.o       \
952     epoll.o         \
953     eventfd.o       \
954     exacctsyst.o    \
955     execl.o         \
956     execl.o         \
957     execv.o         \
958     fcntl.o         \
959     getpagesizes.o  \
960     getpeerucred.o  \
961     inst_sync.o     \
962     issetugid.o     \
963     label.o         \
964     link.o          \
965     lockf.o         \
966     lwp.o           \
967     lwp_cond.o      \
968     lwp_rwlock.o    \
969     lwp_sigmask.o   \
970     meminfosys.o   \
971     mkdir.o         \
972     mknod.o         \
973     msgsys.o        \
974     nfssys.o        \
975     open.o          \
976     pgrpstys.o      \
977     posix_sigwait.o \
978     ppriv.o         \
979     psetstys.o      \
980     rctlstys.o      \
981     readlink.o      \
982     rename.o        \

```

```

983     sbrk.o          \
984     semsys.o        \
985     set_errno.o     \
986     sharefs.o       \
987     shmsys.o        \
988     sidsys.o        \
989     siginterrupt.o  \
990     signal.o        \
991     signalfd.o      \
992     sigpending.o    \
993     sigstack.o      \
994     stat.o          \
995     symlink.o       \
996     taskstys.o      \
997     time.o          \
998     time_util.o     \
999     timerfd.o       \
1000    ucontext.o      \
1001    unlink.o        \
1002    ustat.o         \
1003    utimesys.o     \
1004    zone.o          \

1006 PORTREGEX=        \
1007     glob.o          \
1008     regcmp.o        \
1009     regex.o         \
1010     wordexp.o       \

1012 PORTREGEX64=      \
1013     glob64.o        \

1015 VALUES= values-Xa.o

1017 MOSTOBJS=         \
1018     $(STRETS)       \
1019     $(CRTOBJS)      \
1020     $(DYNOBJS)      \
1021     $(FPOBJS)       \
1022     $(FPASMOBJS)   \
1023     $(ATOMICOBJS)  \
1024     $(CHACHAOBJS)  \
1025     $(XATTOBJS)    \
1026     $(COMOBJS)     \
1027     $(DTRACEOBJS)  \
1028     $(GENOBJS)     \
1029     $(PRFOBJS)     \
1030     $(PORTFP)      \
1031     $(PORTGEN)     \
1032     $(PORTGEN64)   \
1033     $(PORTI18N)    \
1034     $(PORTI18N_COND) \
1035     $(PORTLOCALE)  \
1036     $(PORTPRINT)   \
1037     $(PORTPRINT_C89) \
1038     $(PORTPRINT_W) \
1039     $(PORTREGEX)   \
1040     $(PORTREGEX64) \
1041     $(PORTSTDIO)   \
1042     $(PORTSTDIO64) \
1043     $(PORTSTDIO_C89) \
1044     $(PORTSTDIO_W) \
1045     $(PORTSYS)     \
1046     $(PORTSYS64)   \
1047     $(AIOBJS)      \
1048     $(RTOBJS)      \

```

```

1049     $(SECFLAGSOBJS)      \
1050 #endif /* ! codereview */
1051     $(TPOOLBJS)          \
1052     $(THREADSOBJS)       \
1053     $(THREADSMACHOBJS)   \
1054     $(THREADSASMOBJS)    \
1055     $(UNICODEOBJS)       \
1056     $(UNWINDMACHOBJS)    \
1057     $(UNWINDASMOBJS)     \
1058     $(COMSYSOBJS)        \
1059     $(SYSOBJS)           \
1060     $(COMSYSOBJS64)      \
1061     $(SYSOBJS64)        \
1062     $(VALUES)

1064 TRACEOBJS=      \
1065     plockstat.o

1067 # NOTE: libc.so.1 must be linked with the minimal crti.o and crtn.o
1068 # modules whose source is provided in the $(SRC)/lib/common directory.
1069 # This must be done because otherwise the Sun C compiler would insert
1070 # its own versions of these modules and those versions contain code
1071 # to call out to C++ initialization functions. Such C++ initialization
1072 # functions can call back into libc before thread initialization is
1073 # complete and this leads to segmentation violations and other problems.
1074 # Since libc contains no C++ code, linking with the minimal crti.o and
1075 # crtn.o modules is safe and avoids the problems described above.
1076 OBJECTS= $(CRTI) $(MOSTOBJS) $(CRTN)
1077 CRTSRCS= .././common/sparc

1079 # include common library definitions
1080 include $(SRC)/lib/Makefile.lib

1082 # we need to override the default SONAME here because we might
1083 # be building a variant object (still libc.so.1, but different filename)
1084 SONAME = libc.so.1

1086 CFLAGS += $(CCVERBOSE)

1088 # This is necessary to avoid problems with calling _ex_unwind().
1089 # We probably don't want any inlining anyway.
1090 CFLAGS += -xinline=

1092 CERRWARN += -_gcc=-Wno-parentheses
1093 CERRWARN += -_gcc=-Wno-switch
1094 CERRWARN += -_gcc=-Wno-uninitialized
1095 CERRWARN += -_gcc=-Wno-unused-value
1096 CERRWARN += -_gcc=-Wno-unused-label
1097 CERRWARN += -_gcc=-Wno-unused-variable
1098 CERRWARN += -_gcc=-Wno-type-limits
1099 CERRWARN += -_gcc=-Wno-char-subscripts
1100 CERRWARN += -_gcc=-Wno-clobbered
1101 CERRWARN += -_gcc=-Wno-unused-function
1102 CERRWARN += -_gcc=-Wno-address

1104 # Setting THREAD_DEBUG = -DTHREAD_DEBUG (make THREAD_DEBUG=-DTHREAD_DEBUG ...)
1105 # enables ASSERT() checking in the threads portion of the library.
1106 # This is automatically enabled for DEBUG builds, not for non-debug builds.
1107 THREAD_DEBUG =
1108 $(NOT_RELEASE_BUILD)THREAD_DEBUG = -DTHREAD_DEBUG

1110 # Make string literals read-only to save memory.
1111 CFLAGS += $(XSTRCONST)

1113 ALTPIC= $(TRACEOBJS:%=pics/%)

```

```

1115 $(DYNLIB) := BUILD.SO = $(LD) -o $@ -G $(DYNFLAGS) $(PIC) $(ALTPIC) $(EXTPIC)

1117 MAPFILES =      $(LIBCDIR)/port/mapfile-vers

1119 CFLAGS +=        $(EXTN_CFLAGS)
1120 CPPFLAGS=        -D_REENTRANT -Dsparc $(EXTN_CPPFLAGS) $(THREAD_DEBUG) \
1121                 -I$(LIBCBASE)/inc -I$(LIBCDIR)/inc $(CPPFLAGS.master)
1122 ASFLAGS=         $(EXTN_ASFLAGS) -K pic -P -D__STDC__ -D_ASM $(CPPFLAGS) $(sparc_

1124 # As a favor to the dtrace syscall provider, libc still calls the
1125 # old syscall traps that have been obsoleted by the *at() interfaces.
1126 # Delete this to compile libc using only the new *at() system call traps
1127 CPPFLAGS += -D_RETAIN_OLD_SYSCALLS

1129 # Inform the run-time linker about libc specialized initialization
1130 RTLDINFO =       -z rtdlinfo=tls_rtdlinfo
1131 DYNFLAGS +=      $(RTLDINFO)

1133 # Force libc's internal references to be resolved immediately upon loading
1134 # in order to avoid critical region problems. Since almost all libc symbols
1135 # are marked 'protected' in the mapfiles, this is a minimal set (15 to 20).
1136 DYNFLAGS +=      -znow

1138 DYNFLAGS +=      -e __rtboot
1139 DYNFLAGS +=      $(EXTN_DYNFLAGS)

1141 # Inform the kernel about the initial DTrace area (in case
1142 # libc is being used as the interpreter / runtime linker).
1143 DTRACE_DATA =    -zdtrace=dtrace_data
1144 DYNFLAGS +=      $(DTRACE_DATA)

1146 # DTrace needs an executable data segment.
1147 MAPFILE.NED=

1149 BUILD.s=        $(AS) $(ASFLAGS) $< -o $@

1151 # Override this top level flag so the compiler builds in its native
1152 # C99 mode. This has been enabled to support the complex arithmetic
1153 # added to libc.
1154 C99MODE=         $(C99_ENABLE)

1156 # libc method of building an archive
1157 # The "$(GREP) -v ' L '" part is necessary only until
1158 # lorder is fixed to ignore thread-local variables.
1159 BUILD.AR= $(RM) $@ ; \
1160           $(AR) q $@ `$(LORDER) $(MOSTOBJS:%=$(DIR)/%) | $(GREP) -v ' L ' | $(TSOR

1162 # extra files for the clean target
1163 CLEANFILES=      \
1164     $(LIBCDIR)/port/gen/errlst.c  \
1165     $(LIBCDIR)/port/gen/new_list.c \
1166     assym.h          \
1167     genassym         \
1168     $(LIBCBASE)/crt/_rtld.s      \
1169     $(LIBCBASE)/crt/_rtbootld.s \
1170     pics/_rtbootld.o  \
1171     pics/crti.o       \
1172     pics/crtn.o       \
1173     $(ALTPIC)

1175 CLOBBERFILES += $(LIB_PIC)

1177 # list of C source for lint
1178 SRCS=            \
1179     $(ATOMICOBJS:%.o=$(SRC)/common/atomic/%.c) \
1180     $(XATTROBJS:%.o=$(SRC)/common/xattr/%.c)   \

```



```

1181 $(COMOBSJ:%.o=$(SRC)/common/util/%.c) \
1182 $(DTRACEOBSJ:%.o=$(SRC)/common/dtrace/%.c) \
1183 $(PORTFP:%.o=$(LIBCDIR)/port/fp/%.c) \
1184 $(PORTGEN:%.o=$(LIBCDIR)/port/gen/%.c) \
1185 $(PORTI18N:%.o=$(LIBCDIR)/port/i18n/%.c) \
1186 $(PORTLOCALE:%.o=$(LIBCDIR)/port/locale/%.c) \
1187 $(PORTPRINT:%.o=$(LIBCDIR)/port/print/%.c) \
1188 $(PORTREGEX:%.o=$(LIBCDIR)/port/regex/%.c) \
1189 $(PORTSTDIO:%.o=$(LIBCDIR)/port/stdio/%.c) \
1190 $(PORTSYS:%.o=$(LIBCDIR)/port/sys/%.c) \
1191 $(AIOOBSJ:%.o=$(LIBCDIR)/port/aio/%.c) \
1192 $(RTOBSJ:%.o=$(LIBCDIR)/port/rt/%.c) \
1193 $(SECFLAGSOBSJ:%.o=$(SRC)/common/secflags/%.c) \
1194 #endif /* ! codereview */
1195 $(TPOOLOBSJ:%.o=$(LIBCDIR)/port/tpool/%.c) \
1196 $(THREADSOBSJ:%.o=$(LIBCDIR)/port/threads/%.c) \
1197 $(THREADSMACHOBSJ:%.o=$(LIBCDIR)/$(MACH)/threads/%.c) \
1198 $(UNICODEOBSJ:%.o=$(SRC)/common/unicode/%.c) \
1199 $(UNWINDMACHOBSJ:%.o=$(LIBCDIR)/port/unwind/%.c) \
1200 $(FPOBSJ:%.o=$(LIBCDIR)/$(MACH)/fp/%.c) \
1201 $(LIBCBASE)/crt/_ftou.c \
1202 $(LIBCBASE)/gen/_xregs_clrptr.c \
1203 $(LIBCBASE)/gen/byteorder.c \
1204 $(LIBCBASE)/gen/ecvt.c \
1205 $(LIBCBASE)/gen/endian.c \
1206 $(LIBCBASE)/gen/getctxt.c \
1207 $(LIBCBASE)/gen/lmul.c \
1208 $(LIBCBASE)/gen/makectxt.c \
1209 $(LIBCBASE)/gen/signfolst.c \
1210 $(LIBCBASE)/gen/siglongjmp.c \
1211 $(LIBCBASE)/gen/swapctxt.c \
1212 $(LIBCBASE)/sys/ptrace.c \
1213 $(LIBCBASE)/sys/uadmin.c

```

```

1215 # conditional assignments
1216 $(DYNLIB) := CRTI = crt1.o
1217 $(DYNLIB) := CRTN = crtn.o

```

```

1219 # Files which need the threads .il inline template
1220 TIL=
1221 aio.o \
1222 alloc.o \
1223 assfail.o \
1224 atexit.o \
1225 atfork.o \
1226 cancel.o \
1227 door_calls.o \
1228 err.o \
1229 errno.o \
1230 getctxt.o \
1231 lwp.o \
1232 ma.o \
1233 machdep.o \
1234 posix_aio.o \
1235 pthr_attr.o \
1236 pthr_barrier.o \
1237 pthr_cond.o \
1238 pthr_mutex.o \
1239 pthr_rwlock.o \
1240 pthread.o \
1241 rand.o \
1242 rwlock.o \
1243 scalls.o \
1244 sched.o \
1245 sema.o \
1246 sigaction.o \

```

```

1247 sigev_thread.o \
1248 spawn.o \
1249 stack.o \
1250 swaptxt.o \
1251 synch.o \
1252 tdb_agent.o \
1253 thr.o \
1254 thread_interface.o \
1255 thread_pool.o \
1256 tls.o \
1257 tsd.o \
1258 unwind.o

1260 $(TIL:%=pics/) := CFLAGS += $(LIBCBASE)/threads/sparc.il

1262 # special kludge for inlines with 'cas':
1263 pics/rwlock.o pics/synch.o pics/lwp.o pics/door_calls.o := \
1264     sparc_CFLAGS += -gcc=-Wa,-xarch=v8plus

1266 # Files in port/fp subdirectory that need base.il inline template
1267 IL=
1268 __flt_decim.o \
1269 decimal_bin.o

1271 $(IL:%=pics/) := CFLAGS += $(LIBCBASE)/fp/base.il

1273 # Files in fp subdirectory which need __quad.il inline template
1274 QIL=
1275 __Q_add.o \
1276 __Q_cmp.o \
1277 __Q_cmpe.o \
1278 __Q_div.o \
1279 __Q_dtoq.o \
1280 __Q_fcc.o \
1281 __Q_mul.o \
1282 __Q_qtod.o \
1283 __Q_qtoi.o \
1284 __Q_qtos.o \
1285 __Q_qtou.o \
1286 __Q_sqrt.o \
1287 __Q_stoq.o \
1288 __Q_sub.o

1290 $(QIL:%=pics/) := CFLAGS += $(LIBCDIR)/$(MACH)/fp/__quad.il
1291 pics/_Q%.o := sparc_COPTFLAG = -xO4 -dalign
1292 pics/__quad%.o := sparc_COPTFLAG = -xO4 -dalign

1294 # large-file-aware components that should be built large

1296 $(COMSYSOBSJ64:%=pics/) := \
1297     CPPFLAGS += -D_LARGEFILE_SOURCE -D_FILE_OFFSET_BITS=64

1299 $(SYSOBSJ64:%=pics/) := \
1300     CPPFLAGS += -D_LARGEFILE_SOURCE -D_FILE_OFFSET_BITS=64

1302 $(PORTGEN64:%=pics/) := \
1303     CPPFLAGS += -D_LARGEFILE_SOURCE -D_FILE_OFFSET_BITS=64

1305 $(PORTREGEX64:%=pics/) := \
1306     CPPFLAGS += -D_LARGEFILE_SOURCE -D_FILE_OFFSET_BITS=64

1308 $(PORTSTDIO64:%=pics/) := \
1309     CPPFLAGS += -D_LARGEFILE_SOURCE -D_FILE_OFFSET_BITS=64

1311 $(PORTSYS64:%=pics/) := \
1312     CPPFLAGS += -D_LARGEFILE_SOURCE -D_FILE_OFFSET_BITS=64

```

```

1314 $(PORTSTDIO_W:%=pics/%) := \
1315     CPPFLAGS += -D_WIDE

1317 $(PORTPRINT_W:%=pics/%) := \
1318     CPPFLAGS += -D_WIDE

1320 # printf/scanf functions to support c89-sized intmax_t variables
1321 $(PORTPRINT_C89:%=pics/%) := \
1322     CPPFLAGS += -D_C89_INTMAX32

1324 $(PORTSTDIO_C89:%=pics/%) := \
1325     CPPFLAGS += -D_C89_INTMAX32

1327 $(PORTI18N_COND:%=pics/%) := \
1328     CPPFLAGS += -D_WCS_LONGLONG

1330 pics/arc4random.o :=     CPPFLAGS += -I$(SRC)/common/crypto/chacha

1332 # Files which need extra optimization
1333 pics/getenv.o := sparc_COPTFLAG = -xO4

1335 .KEEP_STATE:

1337 all: $(LIBS) $(LIB_PIC)

1339 lint :=     CPPFLAGS += -I$(LIBCDIR)/$(MACH)/fp
1340 lint :=     CPPFLAGS += -D_MSE_INT_H -D_LCONV_C99
1341 lint :=     LINTFLAGS += -mn

1343 lint:
1344     @echo $(LINT.c) ... $(LDLIBS)
1345     @$$(LINT.c) $(SRCS) $(LDLIBS)

1347 $(LINTLIB) := SRCS=$(LIBCDIR)/port/liblib-1c
1348 $(LINTLIB) := CPPFLAGS += -D_MSE_INT_H
1349 $(LINTLIB) := LINTFLAGS=-nvx

1351 # object files that depend on inline template
1352 $(TIL:%=pics/%) : $(LIBCBASE)/threads/sparc.il
1353 $(IL:%=pics/%) : $(LIBCBASE)/fp/base.il
1354 $(QIL:%=pics/%) : $(LIBCDIR)/$(MACH)/fp/__quad.il

1356 # include common libc targets
1357 include $(LIBCDIR)/Makefile.targ

1359 # We need to strip out all CTF and DOF data from the static library
1360 $(LIB_PIC) := DIR = pics
1361 $(LIB_PIC): pics $$ (PICS)
1362     $(BUILD.AR)
1363     $(MCS) -d -n .SUNW_ctf $@ > /dev/null 2>&1
1364     $(MCS) -d -n .SUNW_dof $@ > /dev/null 2>&1
1365     $(AR) -ts $@ > /dev/null
1366     $(POST_PROCESS_A)

1368 # special cases
1369 $(STRETS:%=pics/%) : $(LIBCBASE)/crt/stret.s
1370     $(AS) $(ASFLAGS) -DSTRET$(F:stret%.o=) $(LIBCBASE)/crt/stret.s -o $@
1371     $(POST_PROCESS_O)

1373 $(LIBCBASE)/crt/_rtbootld.s: $(LIBCBASE)/crt/_rtboot.s $(LIBCBASE)/crt/_rtld.
1374     $(CC) $(CPPFLAGS) $(CTF_FLAGS) -O -S -K pic \
1375     $(LIBCBASE)/crt/_rtld.c -o $(LIBCBASE)/crt/_rtld.s
1376     $(CAT) $(LIBCBASE)/crt/_rtboot.s $(LIBCBASE)/crt/_rtld.s > $@
1377     $(RM) $(LIBCBASE)/crt/_rtld.s

```

```

1379 # partially built from C source
1380 pics/_rtbootld.o: $(LIBCBASE)/crt/_rtbootld.s
1381     $(AS) $(ASFLAGS) $(LIBCBASE)/crt/_rtbootld.s -o $@
1382     $(CTFCONVERT_O)

1384 ASSYMDEP_OBJS= \
1385     _lwp_mutex_unlock.o \
1386     _stack_grow.o \
1387     asm_subr.o \
1388     setjmp.o \
1389     smt_pause.o \
1390     tls_get_addr.o \
1391     unwind_frame.o \
1392     vforkx.o

1394 $(ASSYMDEP_OBJS:%=pics/%) :=     CPPFLAGS += -I.

1396 $(ASSYMDEP_OBJS:%=pics/%) : assym.h

1398 # assym.h build rules

1400 assym.h := CFLAGS += -g

1402 GENASSYM_C = $(LIBCDIR)/$(MACH)/genassym.c

1404 genassym: $(GENASSYM_C)
1405     $(NATIVECC) $(NATIVE_CFLAGS) -I$(LIBCBASE)/inc -I$(LIBCDIR)/inc \
1406     $(CPPFLAGS.native) -o $@ $(GENASSYM_C)

1408 OFFSETS = $(LIBCDIR)/$(MACH)/offsets.in

1410 assym.h: $(OFFSETS) genassym
1411     $(OFFSETS_CREATE) <$(OFFSETS) > $@
1412     ./genassym >> $@

1414 # derived C source and related explicit dependencies
1415 $(LIBCDIR)/port/gen/errlst.c + \
1416 $(LIBCDIR)/port/gen/new_list.c: $(LIBCDIR)/port/gen/errlist $(LIBCDIR)/port/gen/
1417     cd $(LIBCDIR)/port/gen; pwd; $(AWK) -f errlist.awk < errlist

1419 pics/errlst.o: $(LIBCDIR)/port/gen/errlst.c

1421 pics/new_list.o: $(LIBCDIR)/port/gen/new_list.c

```

```

*****
25047 Thu Jun 30 22:02:15 2016
new/usr/src/lib/libc/sparcv9/Makefile.com
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
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 2016 Gary Mills
23 # Copyright (c) 1989, 2010, Oracle and/or its affiliates. All rights reserved.
24 # Copyright (c) 2015, Joyent, Inc. All rights reserved.
25 # Copyright (c) 2013, OmniTI Computer Consulting, Inc. All rights reserved.
26 # Copyright 2013 Garrett D'Amore <garrett@damore.org>
27 #
28 # Copyright 2011 Nexenta Systems, Inc. All rights reserved.
29 # Use is subject to license terms.
30 #

32 LIBCDIR=      $(SRC)/lib/libc
33 LIB_PIC=      libc_pic.a
34 VERS=        .1
35 CPP=         /usr/lib/cpp
36 TARGET_ARCH= sparc

38 # objects are grouped by source directory

40 # Symbol capabilities objects.
41 EXTPICS=
42 $(LIBCDIR)/capabilities/sun4u/sparcv9/pics/symcap.o \
43 $(LIBCDIR)/capabilities/sun4u-opl/sparcv9/pics/symcap.o \
44 $(LIBCDIR)/capabilities/sun4u-us3-hwcap1/sparcv9/pics/symcap.o \
45 $(LIBCDIR)/capabilities/sun4u-us3-hwcap2/sparcv9/pics/symcap.o \
46 $(LIBCDIR)/capabilities/sun4v-hwcap1/sparcv9/pics/symcap.o \
47 $(LIBCDIR)/capabilities/sun4v-hwcap2/sparcv9/pics/symcap.o

49 # local objects
50 STRETS=

52 CRTOBS=
53 __align_cpy_2.o
54 __align_cpy_4.o
55 __align_cpy_8.o
56 _ftou.o
57 cerror.o

```

```

59 DYNOBJS=

61 FPOBJS=
62 _D_cplx_div.o
63 _D_cplx_div_ix.o
64 _D_cplx_div_rx.o
65 _D_cplx_mul.o
66 _F_cplx_div.o
67 _F_cplx_div_ix.o
68 _F_cplx_div_rx.o
69 _F_cplx_mul.o
70 _Q_add.o
71 _Q_cmp.o
72 _Q_cmpe.o
73 _Q_cplx_div.o
74 _Q_cplx_div_ix.o
75 _Q_cplx_div_rx.o
76 _Q_cplx_lr_div.o
77 _Q_cplx_lr_div_ix.o
78 _Q_cplx_lr_div_rx.o
79 _Q_cplx_lr_mul.o
80 _Q_cplx_mul.o
81 _Q_div.o
82 _Q_dtoq.o
83 _Q_fcc.o
84 _Q_itoq.o
85 _Q_mul.o
86 _Q_neg.o
87 _Q_qtod.o
88 _Q_qtoi.o
89 _Q_qtos.o
90 _Q_qtou.o
91 _Q_scl.o
92 _Q_sqrt.o
93 _Q_stoq.o
94 _Q_sub.o
95 _Q_utoq.o

97 FPOBJS64=
98 _Qp_qtox.o
99 _Qp_qtoux.o
100 _Qp_xtoq.o
101 _Qp_xtoq.o
102 __dtoul.o
103 __ftoul.o

105 FPASMOBJS=
106 _Q_get_rp_rd.o
107 __quad_mag64.o
108 fpgetmask.o
109 fpgetrnd.o
110 fpgetsticky.o
111 fpsetmask.o
112 fpsetrnd.o
113 fpsetsticky.o

115 $(__GNUC)FPASMOBJS +=
116 __quad.o

118 ATOMICOBJS=
119 atomic.o

121 CHACHAOBJS=
122 chacha.o

124 XATTROBJS=

```

```

125      xattr_common.o
127 COMOBJS=
128      bcmp.o
129      bcopy.o
130      bsearch.o
131      bzero.o
132      memccpy.o
133      qsort.o
134      strtol.o
135      strtoul.o
136      strtoll.o
137      strtoull.o

139 GENOBJS=
140      _getsp.o
141      _xregs_clrptr.o
142      abs.o
143      alloca.o
144      arc4random.o
145      arc4random_uniform.o
146      ascii_strcasecmp.o
147      byteorder.o
148      cuexit.o
149      ecvt.o
150      endian.o
151      gettxt.o
152      lock.o
153      makectxt.o
154      memchr.o
155      memcmp.o
156      new_list.o
157      setjmp.o
158      siginfolst.o
159      siglongjmp.o
160      smt_pause.o
161      sparc_data.o
162      strchr.o
163      strcmp.o
164      strlcpy.o
165      strncmp.o
166      strncpy.o
167      strlen.o
168      swapctxt.o
169      sync_instruction_memory.o

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

175 SYSOBJS64=

177 COMSYSOBJS=
178      __clock_timer.o
179      __getloadavg.o
180      __rusagesys.o
181      __signotify.o
182      __sigrt.o
183      __time.o
184      __lgrp_home_fast.o
185      __lgrpsys.o
186      __nfssys.o
187      __portfs.o
188      __pset.o
189      __rpcsys.o
190      __sigaction.o

```

```

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

```

```

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

312 SYSOBJS= \
313 __clock_gettime.o \
314 __getcontext.o \
315 __uadmin.o \
316 __lwp_mutex_unlock.o \
317 __stack_grow.o \
318 door.o \
319 forkx.o \
320 forkallx.o \
321 gettimeofday.o \
322 sparc_utrap_install.o \

```

```

323 syscall.o \
324 tls_get_addr.o \
325 uadmin.o \
326 umount.o \
327 uname.o \
328 vforkx.o \

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

334 # objects from source under $(LIBCDIR)/port
335 PORTFP= \
336 __flt_decim.o \
337 __flt_rounds.o \
338 __tbl_10_b.o \
339 __tbl_10_h.o \
340 __tbl_10_s.o \
341 __tbl_2_b.o \
342 __tbl_2_h.o \
343 __tbl_2_s.o \
344 __tbl_fdq.o \
345 __tbl_tens.o \
346 __x_power.o \
347 __base_sup.o \
348 aconvert.o \
349 decimal_bin.o \
350 double_decim.o \
351 econvert.o \
352 fconvert.o \
353 file_decim.o \
354 finite.o \
355 fp_data.o \
356 func_decim.o \
357 gconvert.o \
358 hex_bin.o \
359 ieee_globals.o \
360 pack_float.o \
361 sigfpe.o \
362 string_decim.o \

364 PORTGEN= \
365 __env_data.o \
366 __xftw.o \
367 a64l.o \
368 abort.o \
369 addsev.o \
370 ascii_strncasecmp.o \
371 assert.o \
372 attrat.o \
373 atof.o \
374 atoi.o \
375 atol.o \
376 atoll.o \
377 attropen.o \
378 atexit.o \
379 atfork.o \
380 basename.o \
381 calloc.o \
382 catgets.o \
383 catopen.o \
384 cfgetispeed.o \
385 cfgetospeed.o \
386 cfree.o \
387 cfsetispeed.o \
388 cfsetospeed.o \

```

```

389 cftime.o \
390 clock.o \
391 closedir.o \
392 closefrom.o \
393 confstr.o \
394 crypt.o \
395 csetlen.o \
396 ctime.o \
397 ctime_r.o \
398 daemon.o \
399 deflt.o \
400 directio.o \
401 dirname.o \
402 div.o \
403 drand48.o \
404 dup.o \
405 env_data.o \
406 err.o \
407 errno.o \
408 euclen.o \
409 event_port.o \
410 execvp.o \
411 explicit_bzero.o \
412 fattach.o \
413 fdetach.o \
414 fdopendir.o \
415 ffs.o \
416 flock.o \
417 fls.o \
418 fmtmsg.o \
419 ftime.o \
420 ftok.o \
421 ftw.o \
422 gcvt.o \
423 getauxv.o \
424 getcwd.o \
425 getdate_err.o \
426 getdtablesize.o \
427 getentropy.o \
428 getenv.o \
429 getexecname.o \
430 getgrnam.o \
431 getgrnam_r.o \
432 gethostid.o \
433 gethostname.o \
434 gethz.o \
435 getisax.o \
436 getloadavg.o \
437 getlogin.o \
438 getmntent.o \
439 getnetgrent.o \
440 get_nprocs.o \
441 getopt.o \
442 getopt_long.o \
443 getpagesize.o \
444 getpw.o \
445 getpwnam.o \
446 getpwnam_r.o \
447 getrusage.o \
448 getspent.o \
449 getspent_r.o \
450 getsubopt.o \
451 gettxt.o \
452 getusershell.o \
453 getut.o \
454 getutx.o \

```

```

455 getvfsent.o \
456 getwd.o \
457 getwidth.o \
458 getxby_door.o \
459 gtxt.o \
460 hsearch.o \
461 iconv.o \
462 imaxabs.o \
463 imaxdiv.o \
464 index.o \
465 initgroups.o \
466 insque.o \
467 isaexec.o \
468 isastream.o \
469 isatty.o \
470 killpg.o \
471 klpdlib.o \
472 l64a.o \
473 lckpwwdf.o \
474 lconstants.o \
475 ldivide.o \
476 lexp10.o \
477 lfind.o \
478 lfmt.o \
479 lfmt_log.o \
480 lldiv.o \
481 llog10.o \
482 lltostr.o \
483 lmath.o \
484 localtime.o \
485 lsearch.o \
486 madvise.o \
487 malloc.o \
488 memalign.o \
489 memmem.o \
490 mkdev.o \
491 mkdtemp.o \
492 mkfifo.o \
493 mkstemp.o \
494 mktemp.o \
495 mlock.o \
496 mlockall.o \
497 mon.o \
498 msync.o \
499 munlock.o \
500 munlockall.o \
501 ndbm.o \
502 nftw.o \
503 nlspath_checks.o \
504 nsparse.o \
505 nss_common.o \
506 nss_dbdefs.o \
507 nss_deffinder.o \
508 opendir.o \
509 opt_data.o \
510 perror.o \
511 pfmt.o \
512 pfmt_data.o \
513 pfmt_print.o \
514 pipe.o \
515 plock.o \
516 poll.o \
517 posix_fadvise.o \
518 posix_fallocate.o \
519 posix_madvise.o \
520 posix_memalign.o \

```

```

521     priocntl.o      \
522     privlib.o      \
523     priv_str_xlate.o \
524     psecflags.o    \
525 #endif /* ! codereview */
526     psiginfo.o     \
527     psignal.o      \
528     pt.o           \
529     putpwent.o     \
530     putsptent.o    \
531     raise.o        \
532     rand.o         \
533     random.o       \
534     rctlops.o      \
535     readdir.o      \
536     readdir_r.o    \
537     realpath.o     \
538     reboot.o       \
539     regexpr.o      \
540     remove.o       \
541     rewinddir.o    \
542     rindex.o       \
543     scandir.o      \
544     seekdir.o      \
545     select.o       \
546     setlabel.o     \
547     setpriority.o  \
548     settimeofday.o \
549     sh_locks.o     \
550     sigflag.o      \
551     siglist.o      \
552     sigsend.o      \
553     sigsetops.o    \
554     signal.o       \
555     stack.o        \
556     stpcpy.o       \
557     stpncpy.o      \
558     str2sig.o      \
559     strcase_ormap.o \
560     strcat.o        \
561     strchrnul.o    \
562     strcspn.o      \
563     strdup.o       \
564     strerror.o     \
565     strlcat.o      \
566     strncat.o      \
567     strndup.o      \
568     strpbrk.o      \
569     strrchr.o      \
570     strsep.o       \
571     strsignal.o    \
572     strspn.o       \
573     strstr.o       \
574     strtod.o       \
575     strtointmax.o  \
576     strtok.o       \
577     strtok_r.o     \
578     strtoumax.o    \
579     swab.o         \
580     swapctl.o      \
581     sysconf.o      \
582     syslog.o       \
583     tcdrain.o      \
584     tcflow.o       \
585     tcflush.o      \
586     tcgetattr.o    \

```

```

587     tcgetpgrp.o    \
588     tcgetsid.o     \
589     tcsendbreak.o  \
590     tcsetattr.o    \
591     tcsetpgrp.o    \
592     tell.o         \
593     telldir.o      \
594     tfind.o        \
595     time_data.o    \
596     time_gdata.o   \
597     timespec_get.o \
598     tls_data.o     \
599     truncate.o     \
600     tsdalloc.o     \
601     tsearch.o      \
602     ttyname.o      \
603     ttyslot.o     \
604     ualarm.o       \
605     ucred.o        \
606     valloc.o       \
607     vlfmt.o        \
608     vprintf.o      \
609     waitpid.o      \
610     walkstack.o    \
611     wdata.o        \
612     xgetwidth.o    \
613     xpg4.o         \
614     xpg6.o         \
616 PORTPRINT_W=     \
617     doprnt_w.o    \
619 PORTPRINT=        \
620     asprintf.o    \
621     doprnt.o      \
622     fprintf.o     \
623     printf.o      \
624     snprintf.o    \
625     sprintf.o     \
626     vfprintf.o    \
627     vprintf.o     \
628     vsnprintf.o   \
629     vsprintf.o    \
630     vwprintf.o    \
631     wprintf.o     \
633 # Preserved solely to ease maintenance of 32-bit and 64-bit library builds
634 # This macro should ALWAYS be empty; native APIs are already 'large file'.
635 PORTSTDIO64=
637 PORTSTDIO_W=      \
638     doscan_w.o    \
640 PORTSTDIO=        \
641     __extensions.o \
642     __endopen.o    \
643     __filbuf.o     \
644     __findbuf.o    \
645     __flsbuf.o     \
646     __wrtchk.o     \
647     clearerr.o     \
648     ctermid.o      \
649     ctermid_r.o    \
650     cuserid.o      \
651     data.o         \
652     doscan.o       \

```

```

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

697 PORTI18N= \
698 getwchar.o \
699 putwchar.o \
700 putws.o \
701 strtows.o \
702 wcsnlen.o \
703 wcstoimax.o \
704 wcstol.o \
705 wcstoul.o \
706 wcswcs.o \
707 wscat.o \
708 wschr.o \
709 wscmp.o \
710 wscopy.o \
711 wscspn.o \
712 wsdup.o \
713 wslen.o \
714 wsnecat.o \
715 wsncmp.o \
716 wsncpy.o \
717 wspbrk.o \
718 wsprintf.o \

```

```

719 wsrchr.o \
720 wsscanf.o \
721 wsspno.o \
722 wstod.o \
723 wstok.o \
724 wstol.o \
725 wstoll.o \
726 wsxfrm.o \
727 wmemchr.o \
728 wmemcmp.o \
729 wmemcpy.o \
730 wmemmove.o \
731 wmemset.o \
732 wcsstr.o \
733 gettext.o \
734 gettext_real.o \
735 gettext_util.o \
736 gettext_gnu.o \
737 plural_parser.o \
738 wdresolve.o \
739 _ctype.o \
740 isascii.o \
741 toascii.o \

743 PORTI18N_COND= \
744 wcstol_longlong.o \
745 wcstoul_longlong.o \

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

```



```

785 regcomp.o \
786 regfree.o \
787 regerror.o \
788 regexec.o \
789 rune.o \
790 runetype.o \
791 setlocale.o \
792 setrunelocale.o \
793 strcasecmp.o \
794 strcasestr.o \
795 strcoll.o \
796 strfmon.o \
797 strftime.o \
798 strncasecmp.o \
799 strptime.o \
800 strxfrm.o \
801 table.o \
802 timelocal.o \
803 tolower.o \
804 towlower.o \
805 ungetwc.o \
806 utf8.o \
807 wcrtdomb.o \
808 wcscasecmp.o \
809 wcscoll.o \
810 wcsftime.o \
811 wcsnrtombs.o \
812 wcsrtombs.o \
813 wcstombs.o \
814 wcswidth.o \
815 wcsxfrm.o \
816 wctob.o \
817 wctomb.o \
818 wctrans.o \
819 wctype.o \
820 wcwidth.o \
821 wscoll.o

823 AIOBJS= \
824 aio.o \
825 aio_alloc.o \
826 posix_aio.o

828 RTOBJS= \
829 clock_timer.o \
830 mqueue.o \
831 pos4obj.o \
832 sched.o \
833 sem.o \
834 shm.o \
835 sigev_thread.o

837 SECFLAGSOBJS= \
838 secflags.o

840 #endif /* ! codereview */
841 TPOOLBJS= \
842 thread_pool.o

844 THREADSOBJS= \
845 alloc.o \
846 assfail.o \
847 cll_thr.o \
848 cancel.o \
849 door_calls.o \
850 tmem.o

```

```

851 pthread_attr.o \
852 pthread_barrier.o \
853 pthread_cond.o \
854 pthread_mutex.o \
855 pthread_rwlock.o \
856 pthread.o \
857 rwlock.o \
858 scalls.o \
859 sema.o \
860 sigaction.o \
861 spawn.o \
862 synch.o \
863 tdb_agent.o \
864 thr.o \
865 thread_interface.o \
866 tls.o \
867 tsd.o

869 THREADSMACHOBJS= \
870 machdep.o

872 THREADSASMOBJS= \
873 asm_subr.o

875 UNICODOBJS= \
876 u8_textprep.o \
877 uconv.o

879 UNWINDMACHOBJS= \
880 unwind.o

882 UNWINDASMOBJS= \
883 unwind_frame.o

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

889 PORTSYS= \
890 _autofssys.o \
891 access.o \
892 acctctl.o \
893 BSD_signal.o \
894 chmod.o \
895 chown.o \
896 corectl.o \
897 eventfd.o \
898 epoll.o \
899 exacctsys.o \
900 execl.o \
901 execl.o \
902 execv.o \
903 fcntl.o \
904 getpagesizes.o \
905 getpeerucred.o \
906 inst_sync.o \
907 issetugid.o \
908 label.o \
909 link.o \
910 lockf.o \
911 lwp.o \
912 lwp_cond.o \
913 lwp_rwlock.o \
914 lwp_sigmask.o \
915 meminfosys.o \
916 mkdir.o

```

```

917      mknod.o      \
918      msgsys.o     \
919      nfssys.o     \
920      open.o       \
921      pgrpsys.o    \
922      posix_sigwait.o \
923      ppriv.o      \
924      psetsys.o    \
925      rctlsys.o    \
926      readlink.o   \
927      rename.o     \
928      sbrk.o       \
929      semsys.o     \
930      set_errno.o  \
931      sharefs.o    \
932      shmsys.o     \
933      sidsys.o     \
934      siginterrupt.o \
935      signal.o     \
936      signalfd.o   \
937      sigpending.o \
938      sigstack.o   \
939      stat.o       \
940      symlink.o    \
941      tasksys.o   \
942      time.o       \
943      time_util.o  \
944      timerfd.o    \
945      ucontext.o   \
946      unlink.o     \
947      ustat.o      \
948      utimesys.o  \
949      zone.o       \

951 PORTREGEX=      \
952      glob.o       \
953      regcmp.o     \
954      regex.o      \
955      wordexp.o    \

957 VALUES= values-Xa.o

959 MOSTOBSJ=      \
960      $(STRETS)   \
961      $(CRTOBSJ)  \
962      $(DYNOBJS)  \
963      $(FPOBSJ)   \
964      $(FPOBSJ64) \
965      $(FPASMOBSJ) \
966      $(ATOMICOBJS) \
967      $(CHACHAOBSJ) \
968      $(XATTROBSJ) \
969      $(COMOBSJ)  \
970      $(GENOBSJ)  \
971      $(PRFOBSJ)  \
972      $(PORTFFP)  \
973      $(PORTGEN)  \
974      $(PORTGEN64) \
975      $(PORTI18N) \
976      $(PORTI18N_COND) \
977      $(PORTLOCALE) \
978      $(PORTPRINT) \
979      $(PORTPRINT_W) \
980      $(PORTREGEX) \
981      $(PORTSTDIO) \
982      $(PORTSTDIO64) \

```

```

983      $(PORTSTDIO_W) \
984      $(PORTSYS)   \
985      $(PORTSYS64) \
986      $(AIOOBSJ)  \
987      $(RTOBSJ)   \
988      $(SECFLAGSOBSJ) \
989      #endif /* ! codereview */
990      $(TPOOLOBSJ) \
991      $(THREADSOBSJ) \
992      $(THREADSMACHOBSJ) \
993      $(THREADSASMOBSJ) \
994      $(UNICODEOBSJ) \
995      $(UNWINDMACHOBSJ) \
996      $(UNWINDASMOBSJ) \
997      $(COMSYSOBSJ) \
998      $(SYSOBSJ)   \
999      $(COMSYSOBSJ64) \
1000     $(SYSOBSJ64) \
1001     $(VALUES)

1003 TRACEOBSJ=      \
1004     plockstat.o

1006 # NOTE: libc.so.1 must be linked with the minimal crt1.o and crtn.o
1007 # modules whose source is provided in the $(SRC)/lib/common directory.
1008 # This must be done because otherwise the Sun C compiler would insert
1009 # its own versions of these modules and those versions contain code
1010 # to call out to C++ initialization functions. Such C++ initialization
1011 # functions can call back into libc before thread initialization is
1012 # complete and this leads to segmentation violations and other problems.
1013 # Since libc contains no C++ code, linking with the minimal crt1.o and
1014 # crtn.o modules is safe and avoids the problems described above.
1015 OBJECTS= $(CRTI) $(MOSTOBSJ) $(CRTN)
1016 CRTSRC= ../common/sparcv9

1018 # include common library definitions
1019 include $(SRC)/lib/Makefile.lib
1020 include $(SRC)/lib/Makefile.lib.64

1022 # we need to override the default SONAME here because we might
1023 # be building a variant object (still libc.so.1, but different filename)
1024 SONAME = libc.so.1

1026 CFLAGS64 += $(CCVERBOSE)

1028 # This is necessary to avoid problems with calling _ex_unwind().
1029 # We probably don't want any inlining anyway.
1030 CFLAGS64 += -xinline=

1032 CERRWARN += -_gcc=-Wno-parentheses
1033 CERRWARN += -_gcc=-Wno-switch
1034 CERRWARN += -_gcc=-Wno-uninitialized
1035 CERRWARN += -_gcc=-Wno-unused-value
1036 CERRWARN += -_gcc=-Wno-unused-label
1037 CERRWARN += -_gcc=-Wno-unused-variable
1038 CERRWARN += -_gcc=-Wno-type-limits
1039 CERRWARN += -_gcc=-Wno-char-subscripts
1040 CERRWARN += -_gcc=-Wno-clobbered
1041 CERRWARN += -_gcc=-Wno-unused-function
1042 CERRWARN += -_gcc=-Wno-address

1044 # Setting THREAD_DEBUG = -DTHREAD_DEBUG (make THREAD_DEBUG=-DTHREAD_DEBUG ...)
1045 # enables ASSERT() checking in the threads portion of the library.
1046 # This is automatically enabled for DEBUG builds, not for non-debug builds.
1047 THREAD_DEBUG =
1048 $(NOT_RELEASE_BUILD)THREAD_DEBUG = -DTHREAD_DEBUG

```

```

1050 # Make string literals read-only to save memory.
1051 CFLAGS64 += $(XSTRCONST)

1053 ALTPIC= $(TRACEOBS:%=pics/)

1055 $(DYNLIB) := BUILD.SO = $(LD) -o $$@ -G $(DYNFLAGS) $(PICS) $(ALTPIC) $(EXTPIC)

1057 MAPFILES = $(LIBCDIR)/port/mapfile-vers

1059 sparcv9_C_PICFLAGS= -K PIC
1060 CFLAGS64 += $(EXTN_CFLAGS)
1061 CPPFLAGS= -D_REENTRANT -Dsparc $(EXTN_CPPFLAGS) $(THREAD_DEBUG) \
1062           -I$(LIBCBASE)/inc -I$(LIBCDIR)/inc $(CPPFLAGS.master)
1063 ASFLAGS= $(EXTN_ASFLAGS) -K PIC -P -D__STDC__ -D__ASM -D__sparcv9 $(CPPFLA
1064          $(sparcv9_AS_XARCH)

1066 # As a favor to the dtrace syscall provider, libc still calls the
1067 # old syscall traps that have been obsoleted by the *at() interfaces.
1068 # Delete this to compile libc using only the new *at() system call traps
1069 CPPFLAGS += -D_RETAIN_OLD_SYSCALLS

1071 # Inform the run-time linker about libc specialized initialization
1072 RTLDINFO = -z rtldinfo=tls_rtldinfo
1073 DYNFLAGS += $(RTLDINFO)

1075 # Force libc's internal references to be resolved immediately upon loading
1076 # in order to avoid critical region problems. Since almost all libc symbols
1077 # are marked 'protected' in the mapfiles, this is a minimal set (15 to 20).
1078 DYNFLAGS += -znov

1080 DYNFLAGS += $(EXTN_DYNFLAGS)

1082 BUILD.s= $(AS) $(ASFLAGS) $< -o $$@

1084 # Override this top level flag so the compiler builds in its native
1085 # C99 mode. This has been enabled to support the complex arithmetic
1086 # added to libc.
1087 C99MODE= $(C99_ENABLE)

1089 # libc method of building an archive
1090 # The "$(GREP) -v ' L '" part is necessary only until
1091 # lorder is fixed to ignore thread-local variables.
1092 BUILD.AR= $(RM) $$@ ; \
1093           $(AR) q $$@ '$(LORDER) $(MOSTOBS:%=$(DIR)/%) | $(GREP) -v ' L ' | $(TSOR

1095 # extra files for the clean target
1096 CLEANFILES= \
1097            $(LIBCDIR)/port/gen/errlst.c \
1098            $(LIBCDIR)/port/gen/new_list.c \
1099            assym.h \
1100            genassym \
1101            pics/crti.o \
1102            pics/crtn.o \
1103            $(ALTPIC)

1105 CLOBBERFILES += $(LIB_PIC)

1107 # list of C source for lint
1108 SRCS= \
1109        $(ATOMICOBJS:%=$(SRC)/common/atomic/%.c) \
1110        $(XATTOBJS:%=$(SRC)/common/xattr/%.c) \
1111        $(COMOBS:%=$(SRC)/common/util/%.c) \
1112        $(PORTFP:%=$(LIBCDIR)/port/fp/%.c) \
1113        $(PORTGEN:%=$(LIBCDIR)/port/gen/%.c) \
1114        $(PORTI18N:%=$(LIBCDIR)/port/i18n/%.c)

```

```

1115        $(PORTLOCALE:%=$(LIBCDIR)/port/locale/%.c) \
1116        $(PORTPRINT:%=$(LIBCDIR)/port/print/%.c) \
1117        $(PORTREGEX:%=$(LIBCDIR)/port/regex/%.c) \
1118        $(PORTSTDIO:%=$(LIBCDIR)/port/stdio/%.c) \
1119        $(PORTSYS:%=$(LIBCDIR)/port/sys/%.c) \
1120        $(AIOBJS:%=$(LIBCDIR)/port/aio/%.c) \
1121        $(RTOBJS:%=$(LIBCDIR)/port/rt/%.c) \
1122        $(SECFLAGSOBJS:%=$(SRC)/common/secflags/%.c) \
1123 #endif /* ! codereview */
1124        $(TPOOLBJS:%=$(LIBCDIR)/port/tpool/%.c) \
1125        $(THREADSOBJS:%=$(LIBCDIR)/port/threads/%.c) \
1126        $(THREADSMACHOBJS:%=$(LIBCDIR)/$(MACH)/threads/%.c) \
1127        $(UNICODEOBJS:%=$(SRC)/common/unicode/%.c) \
1128        $(UNWINDMACHOBJS:%=$(LIBCDIR)/port/unwind/%.c) \
1129        $(FPOBJS:%=$(LIBCDIR)/$(MACH)/fp/%.c) \
1130        $(FPOBJS64:%=$(LIBCBASE)/fp/%.c) \
1131        $(LIBCBASE)/crt/_ftou.c \
1132        $(LIBCBASE)/gen/_xregs_clrprtr.c \
1133        $(LIBCBASE)/gen/byteorder.c \
1134        $(LIBCBASE)/gen/endian.c \
1135        $(LIBCBASE)/gen/ecvt.c \
1136        $(LIBCBASE)/gen/getctxt.c \
1137        $(LIBCBASE)/gen/makectxt.c \
1138        $(LIBCBASE)/gen/siginfofst.c \
1139        $(LIBCBASE)/gen/siglongjmp.c \
1140        $(LIBCBASE)/gen/swapctxt.c

1142 # conditional assignments
1143 $(DYNLIB) := CRTI = crt.i.o
1144 $(DYNLIB) := CRTN = crtn.o

1146 # Files which need the threads .il inline template
1147 TIL= \
1148        aio.o \
1149        alloc.o \
1150        assfail.o \
1151        atexit.o \
1152        atfork.o \
1153        cancel.o \
1154        door_calls.o \
1155        err.o \
1156        errno.o \
1157        getctxt.o \
1158        lwp.o \
1159        ma.o \
1160        machdep.o \
1161        posix_aio.o \
1162        pthr_attr.o \
1163        pthr_barrier.o \
1164        pthr_cond.o \
1165        pthr_mutex.o \
1166        pthr_rwlock.o \
1167        pthread.o \
1168        rand.o \
1169        rwlock.o \
1170        scalls.o \
1171        sched.o \
1172        sema.o \
1173        sigaction.o \
1174        sigev_thread.o \
1175        spawn.o \
1176        stack.o \
1177        swapctxt.o \
1178        synch.o \
1179        tdb_agent.o \
1180        thr.o

```

```

1181     thread_interface.o  \
1182     thread_pool.o      \
1183     tls.o              \
1184     tsd.o              \
1185     unwind.o

1187 $(TIL:%=pics/%) := CFLAGS64 += $(LIBCBASE)/threads/sparcv9.il

1189 # Files in fp, port/fp subdirectories that need base.il inline template
1190 IL=
1191     __flt_decim.o      \
1192     decimal_bin.o

1194 $(IL:%=pics/%) := CFLAGS64 += $(LIBCBASE)/fp/base.il

1196 # Files in fp subdirectory which need __quad.il inline template
1197 QIL=
1198     __Q_add.o          \
1199     __Q_cmp.o          \
1200     __Q_cmpe.o         \
1201     __Q_div.o          \
1202     __Q_dtoq.o         \
1203     __Q_fcc.o          \
1204     __Q_mul.o          \
1205     __Q_qtod.o         \
1206     __Q_qtoi.o         \
1207     __Q_qtos.o         \
1208     __Q_qtou.o        \
1209     __Q_sqrt.o         \
1210     __Q_stoq.o         \
1211     __Q_sub.o          \
1212     __Qp_qtox.o       \
1213     __Qp_qtoux.o

1215 $(QIL:%=pics/%) := CFLAGS64 += $(LIBCDIR)/$(MACH)/fp/__quad.il
1216 pics/_Qp%.o := CFLAGS64 += -I$(LIBCDIR)/$(MACH)/fp
1217 pics/_Q%.o := sparcv9_COPTFLAG = -xO4 -xchip=ultra

1219 # Files in crt subdirectory which need muldiv64.il inline template
1220 #CIL= mul64.o divrem64.o
1221 #$(CIL:%=pics/%) := CFLAGS += $(LIBCBASE)/crt/mul64.il

1223 # large-file-aware components that should be built large

1225 #$(COMSYSOBS64:%=pics/%) := \
1226 #     CPPFLAGS += -D_LARGEFILE_SOURCE -D_FILE_OFFSET_BITS=64

1228 #$(SYSOBS64:%=pics/%) := \
1229 #     CPPFLAGS += -D_LARGEFILE_SOURCE -D_FILE_OFFSET_BITS=64

1231 #$(PORTGEN64:%=pics/%) := \
1232 #     CPPFLAGS += -D_LARGEFILE_SOURCE -D_FILE_OFFSET_BITS=64

1234 #$(PORTSTDIO64:%=pics/%) := \
1235 #     CPPFLAGS += -D_LARGEFILE_SOURCE -D_FILE_OFFSET_BITS=64

1237 #$(PORTSYS64:%=pics/%) := \
1238 #     CPPFLAGS += -D_LARGEFILE_SOURCE -D_FILE_OFFSET_BITS=64

1240 $(PORTSTDIO_W:%=pics/%) := \
1241     CPPFLAGS += -D_WIDE

1243 $(PORTPRINT_W:%=pics/%) := \
1244     CPPFLAGS += -D_WIDE

1246 $(PORTI18N_COND:%=pics/%) := \

```

```

1247     CPPFLAGS += -D_WCS_LONLONG

1249 pics/arc4random.o := CPPFLAGS += -I$(SRC)/common/crypto/chacha

1251 # Files which need extra optimization
1252 pics/getenv.o := sparcv9_COPTFLAG = -xO4

1254 .KEEP_STATE:

1256 all: $(LIBS) $(LIB_PIC)

1258 lint := CPPFLAGS += -I$(LIBCDIR)/$(MACH)/fp
1259 lint := CPPFLAGS += -D_MSE_INT_H -D_LCONV_C99
1260 lint := LINTFLAGS64 += -mn

1262 lint:
1263     @echo $(LINT.c) ... $(LDLIBS)
1264     @$$(LINT.c) $(SRCS) $(LDLIBS)

1266 $(LINTLIB) := SRCS=$(LIBCDIR)/port/lib-lc
1267 $(LINTLIB) := CPPFLAGS += -D_MSE_INT_H
1268 $(LINTLIB) := LINTFLAGS64=-nvx -m64

1270 # object files that depend on inline template
1271 $(TIL:%=pics/%) := $(LIBCBASE)/threads/sparcv9.il
1272 $(IL:%=pics/%) := $(LIBCBASE)/fp/base.il
1273 $(QIL:%=pics/%) := $(LIBCDIR)/$(MACH)/fp/__quad.il
1274 #$(CIL:%=pics/%) := $(LIBCBASE)/crt/muldiv64.il

1276 # include common libc targets
1277 include $(LIBCDIR)/Makefile.targ

1279 # We need to strip out all CTF and DOF data from the static library
1280 $(LIB_PIC) := DIR = pics
1281 $(LIB_PIC): pics $$$$(PICS)
1282     $(BUILD.AR)
1283     $(MCS) -d -n .SUNW_ctf $$@ > /dev/null 2>&1
1284     $(MCS) -d -n .SUNW_dof $$@ > /dev/null 2>&1
1285     $(AR) -ts $$@ > /dev/null
1286     $(POST_PROCESS_A)

1288 # special cases
1289 #$(STRETS:%=pics/%) := crt/stret.s
1290 #     $(AS) $(ASFLAGS) -DSTRET$(@F:stret%.o=) crt/stret.s -o $$@
1291 #     $(POST_PROCESS_O)

1293 #crt/_rtbootld.s: crt/_rtboot.s crt/_rtld.c
1294 #     $(CC) $(CPPFLAGS) -O -s -K pic crt/_rtld.c -o crt/_rtld.s
1295 #     $(CAT) crt/_rtboot.s crt/_rtld.s > $$@
1296 #     $(RM) crt/_rtld.s

1298 ASSYMDEP_OBJS=
1299     _lwp_mutex_unlock.o \
1300     _stack_grow.o      \
1301     asm_subr.o         \
1302     setjmp.o           \
1303     smt_pause.o        \
1304     tls_get_addr.o     \
1305     unwind_frame.o     \
1306     vforkx.o

1308 $(ASSYMDEP_OBJS:%=pics/%) := CPPFLAGS += -I.

1310 $(ASSYMDEP_OBJS:%=pics/%) := assym.h

1312 # assym.h build rules

```

```
1314 assym.h := CFLAGS64 += -g
1316 GENASSYM_C = $(LIBCDIR)/$(MACH)/genassym.c
1318 genassym: $(GENASSYM_C)
1319     $(NATIVECC) $(NATIVE_CFLAGS) -I$(LIBCBASE)/inc -I$(LIBCDIR)/inc \
1320     $(CPPFLAGS.native) -o $@ $(GENASSYM_C)
1322 OFFSETS = $(LIBCDIR)/$(MACH)/offsets.in
1324 assym.h: $(OFFSETS) genassym
1325     $(OFFSETS_CREATE) <$(OFFSETS) >$@
1326     ./genassym >>$@
1328 # derived C source and related explicit dependencies
1329 $(LIBCDIR)/port/gen/new_list.c: $(LIBCDIR)/port/gen/errlist $(LIBCDIR)/port/gen/
1330     cd $(LIBCDIR)/port/gen; pwd; $(AWK) -f errlist.awk < errlist
1332 pics/new_list.o: $(LIBCDIR)/port/gen/new_list.c
```

```

*****
90707 Thu Jun 30 22:02:16 2016
new/usr/src/lib/libproc/common/Pcontrol.c
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */

22 /*
23  * Copyright 2010 Sun Microsystems, Inc. All rights reserved.
24  * Use is subject to license terms.
25  *
26  * Portions Copyright 2007 Chad Mynhier
27  * Copyright 2012 DEY Storage Systems, Inc. All rights reserved.
28  * Copyright (c) 2013 by Delphix. All rights reserved.
29  * Copyright 2015, Joyent, Inc.
30 */

32 #include <assert.h>
33 #include <stdio.h>
34 #include <stdlib.h>
35 #include <unistd.h>
36 #include <ctype.h>
37 #include <fcntl.h>
38 #include <string.h>
39 #include <strings.h>
40 #include <memory.h>
41 #include <errno.h>
42 #include <dirent.h>
43 #include <limits.h>
44 #include <signal.h>
45 #include <atomic.h>
46 #include <zone.h>
47 #include <sys/types.h>
48 #include <sys/uo.h>
49 #include <sys/stat.h>
50 #include <sys/resource.h>
51 #include <sys/param.h>
52 #include <sys/stack.h>
53 #include <sys/fault.h>
54 #include <sys/syscall.h>
55 #include <sys/sysmacros.h>
56 #include <sys/systeminfo.h>
57 #include <sys/secflags.h>
58 #endif /* ! codereview */

```

```

60 #include "libproc.h"
61 #include "Pcontrol.h"
62 #include "Putil.h"
63 #include "P32ton.h"

65 int     _libproc_debug;      /* set non-zero to enable debugging printf's */
66 int     _libproc_no_qsort;   /* set non-zero to inhibit sorting */
67        /* of symbol tables */
68 int     _libproc_incore_elf; /* only use in-core elf data */

70 sigset_t blockable_sigs;    /* signals to block when we need to be safe */
71 static int minfd;          /* minimum file descriptor returned by dupfd(fd, 0) */
72 char     procfs_path[PATH_MAX] = "/proc";

74 /*
75  * Function prototypes for static routines in this module.
76  */
77 static void     deadlock(struct ps_prochandle *);
78 static void     restore_tracing_flags(struct ps_prochandle *);
79 static void     Lfree_internal(struct ps_prochandle *, struct ps_lwphandle *);
80 static prheader_t *read_lfile(struct ps_prochandle *, const char *);

82 /*
83  * Ops vector functions for live processes.
84  */

86 /*ARGSUSED*/
87 static ssize_t
88 Pread_live(struct ps_prochandle *P, void *buf, size_t n, uintptr_t addr,
89            void *data)
90 {
91     return (pread(P->asfd, buf, n, (off_t)addr));
92 }

94 /*ARGSUSED*/
95 static ssize_t
96 Pwrite_live(struct ps_prochandle *P, const void *buf, size_t n, uintptr_t addr,
97             void *data)
98 {
99     return (pwrite(P->asfd, buf, n, (off_t)addr));
100 }

102 /*ARGSUSED*/
103 static int
104 Pread_maps_live(struct ps_prochandle *P, prmap_t **Pmapp, ssize_t *nmapp,
105                void *data)
106 {
107     char mapfile[PATH_MAX];
108     int mapfd;
109     struct stat statb;
110     ssize_t nmap;
111     prmap_t *Pmap = NULL;

113     (void) snprintf(mapfile, sizeof (mapfile), "%s/%d/map",
114                    procfs_path, (int)P->pid);
115     if ((mapfd = open(mapfile, O_RDONLY)) < 0 ||
116         fstat(mapfd, &statb) != 0 ||
117         statb.st_size < sizeof (prmap_t) ||
118         (Pmap = malloc(statb.st_size)) == NULL ||
119         (nmap = pread(mapfd, Pmap, statb.st_size, 0L)) <= 0 ||
120         (nmap /= sizeof (prmap_t)) == 0) {
121         if (Pmap != NULL)
122             free(Pmap);
123         if (mapfd >= 0)
124             (void) close(mapfd);

```

```

125     Preset_maps(P); /* utter failure; destroy tables */
126     return (-1);
127 }
128 (void) close(mapfd);

130 *Pmapp = Pmap;
131 *nmapp = nmap;

133     return (0);
134 }

136 /*ARGSUSED*/
137 static void
138 Pread_aux_live(struct ps_prochandle *P, auxv_t **auxvp, int *nauxp, void *data)
139 {
140     char auxfile[64];
141     int fd;
142     struct stat statb;
143     auxv_t *auxv;
144     ssize_t naux;

146     (void) snprintf(auxfile, sizeof (auxfile), "%s/%d/auxv",
147                    procfs_path, (int)P->pid);
148     if ((fd = open(auxfile, O_RDONLY)) < 0) {
149         dprintf("%s: failed to open %s: %s\n",
150              __func__, auxfile, strerror(errno));
151         return;
152     }

154     if (fstat(fd, &statb) == 0 &&
155         statb.st_size >= sizeof (auxv_t) &&
156         (auxv = malloc(statb.st_size + sizeof (auxv_t))) != NULL) {
157         if ((naux = read(fd, auxv, statb.st_size)) < 0 ||
158             (naux /= sizeof (auxv_t)) < 1) {
159             dprintf("%s: read failed: %s\n",
160                  __func__, strerror(errno));
161             free(auxv);
162         } else {
163             auxv[naux].a_type = AT_NULL;
164             auxv[naux].a_un.a_val = 0L;

166             *auxvp = auxv;
167             *nauxp = (int)naux;
168         }
169     }

171     (void) close(fd);
172 }

174 /*ARGSUSED*/
175 static int
176 Pcred_live(struct ps_prochandle *P, pcred_t *pcrp, int ngroups, void *data)
177 {
178     return (proc_get_cred(P->pid, pcrp, ngroups));
179 }

181 /* ARGSUSED */
182 static int
183 Psecflags_live(struct ps_prochandle *P, prsecflags_t **psf, void *data)
184 {
185     return (proc_get_secflags(P->pid, psf));
186 }

188 #endif /* ! codereview */
189 /*ARGSUSED*/
190 static int

```

```

191 Ppriv_live(struct ps_prochandle *P, prpriv_t **pprv, void *data)
192 {
193     prpriv_t *pp;

195     pp = proc_get_priv(P->pid);
196     if (pp == NULL) {
197         return (-1);
198     }

200     *pprv = pp;
201     return (0);
202 }

204 /*ARGSUSED*/
205 static const psinfo_t *
206 Ppsinfo_live(struct ps_prochandle *P, psinfo_t *psinfo, void *data)
207 {
208     if (proc_get_psinfo(P->pid, psinfo) == -1)
209         return (NULL);

211     return (psinfo);
212 }

214 /*ARGSUSED*/
215 static prheader_t *
216 Plstatus_live(struct ps_prochandle *P, void *data)
217 {
218     return (read_lfile(P, "lstatus"));
219 }

221 /*ARGSUSED*/
222 static prheader_t *
223 Plpsinfo_live(struct ps_prochandle *P, void *data)
224 {
225     return (read_lfile(P, "lpsinfo"));
226 }

228 /*ARGSUSED*/
229 static char *
230 Pplatform_live(struct ps_prochandle *P, char *s, size_t n, void *data)
231 {
232     if (sysinfo(SI_PLATFORM, s, n) == -1)
233         return (NULL);
234     return (s);
235 }

237 /*ARGSUSED*/
238 static int
239 Puname_live(struct ps_prochandle *P, struct utsname *u, void *data)
240 {
241     return (uname(u));
242 }

244 /*ARGSUSED*/
245 static char *
246 Pzonename_live(struct ps_prochandle *P, char *s, size_t n, void *data)
247 {
248     if (getzonenamebyid(P->status.pr_zoneid, s, n) < 0)
249         return (NULL);
250     s[n - 1] = '\0';
251     return (s);
252 }

254 /*
255  * Callback function for Pfindexec(). We return a match if we can stat the
256  * suggested pathname and confirm its device and inode number match our

```

```

257 * previous information about the /proc/<pid>/object/a.out file.
258 */
259 static int
260 stat_exec(const char *path, void *arg)
261 {
262     struct stat64 *stp = arg;
263     struct stat64 st;
264
265     return (stat64(path, &st) == 0 && S_ISREG(st.st_mode) &&
266         stp->st_dev == st.st_dev && stp->st_ino == st.st_ino);
267 }
268
269 /*ARGSUSED*/
270 static char *
271 Pexecname_live(struct ps_prochandle *P, char *buf, size_t buflen, void *data)
272 {
273     char exec_name[PATH_MAX];
274     char cwd[PATH_MAX];
275     char proc_cwd[64];
276     struct stat64 st;
277     int ret;
278
279     /*
280      * Try to get the path information first.
281      */
282     (void) snprintf(exec_name, sizeof(exec_name),
283         "%s/%d/path/a.out", procfs_path, (int)P->pid);
284     if ((ret = readlink(exec_name, buf, buflen - 1)) > 0) {
285         buf[ret] = '\0';
286         (void) Pfindobj(P, buf, buf, buflen);
287         return (buf);
288     }
289
290     /*
291      * Stat the executable file so we can compare Pfindexec's
292      * suggestions to the actual device and inode number.
293      */
294     (void) snprintf(exec_name, sizeof(exec_name),
295         "%s/%d/object/a.out", procfs_path, (int)P->pid);
296
297     if (stat64(exec_name, &st) != 0 || !S_ISREG(st.st_mode))
298         return (NULL);
299
300     /*
301      * Attempt to figure out the current working directory of the
302      * target process. This only works if the target process has
303      * not changed its current directory since it was exec'd.
304      */
305     (void) snprintf(proc_cwd, sizeof(proc_cwd),
306         "%s/%d/path/cwd", procfs_path, (int)P->pid);
307
308     if ((ret = readlink(proc_cwd, cwd, PATH_MAX - 1)) > 0)
309         cwd[ret] = '\0';
310
311     (void) Pfindexec(P, ret > 0 ? cwd : NULL, stat_exec, &st);
312
313     return (NULL);
314 }
315
316 #if defined(__i386) || defined(__amd64)
317 /*ARGSUSED*/
318 static int
319 Pldt_live(struct ps_prochandle *P, struct ssd *pldt, int nldt, void *data)
320 {
321     return (proc_get_ldt(P->pid, pldt, nldt));
322 }

```

```

323 #endif
324
325 static const ps_ops_t P_live_ops = {
326     .pop_pread = Pread_live,
327     .pop_pwrite = Pwrite_live,
328     .pop_read_maps = Pread_maps_live,
329     .pop_read_aux = Pread_aux_live,
330     .pop_cred = Pcred_live,
331     .pop_priv = Ppriv_live,
332     .pop_psinfo = Ppsinfo_live,
333     .pop_lstatus = Plstatus_live,
334     .pop_lpsinfo = Plpsinfo_live,
335     .pop_platform = Pplatform_live,
336     .pop_uname = Puname_live,
337     .pop_zonename = Pzonename_live,
338     .pop_execname = Pexecname_live,
339     .pop_secflags = Psecflags_live,
340 #endif /* !codereview */
341 #if defined(__i386) || defined(__amd64)
342     .pop_ldt = Pldt_live
343 #endif
344 };
345
346 /*
347  * This is the library's .init handler.
348  */
349 #pragma init(_libproc_init)
350 void
351 _libproc_init(void)
352 {
353     _libproc_debug = getenv("LIBPROC_DEBUG") != NULL;
354     _libproc_no_qsort = getenv("LIBPROC_NO_QSORT") != NULL;
355     _libproc_incore_elf = getenv("LIBPROC_INCORE_ELF") != NULL;
356
357     (void) sigfillset(&blockable_sigs);
358     (void) sigdelset(&blockable_sigs, SIGKILL);
359     (void) sigdelset(&blockable_sigs, SIGSTOP);
360 }
361
362 void
363 Pset_procfs_path(const char *path)
364 {
365     (void) snprintf(procfs_path, sizeof(procfs_path), "%s", path);
366 }
367
368 /*
369  * Call set_minfd() once before calling dupfd() several times.
370  * We assume that the application will not reduce its current file
371  * descriptor limit lower than 512 once it has set at least that value.
372  */
373 int
374 set_minfd(void)
375 {
376     static mutex_t minfd_lock = DEFAULTMUTEX;
377     struct rlimit rlim;
378     int fd;
379
380     if ((fd = minfd) < 256) {
381         (void) mutex_lock(&minfd_lock);
382         if ((fd = minfd) < 256) {
383             if (getrlimit(RLIMIT_NOFILE, &rlim) != 0)
384                 rlim.rlim_cur = rlim.rlim_max = 0;
385             if (rlim.rlim_cur >= 512)
386                 fd = 256;
387             else if ((fd = rlim.rlim_cur / 2) < 3)
388                 fd = 3;

```



```

389         membar_producer();
390         minfd = fd;
391     }
392     (void) mutex_unlock(&minfd_lock);
393 }
394     return (fd);
395 }

397 int
398 dupfd(int fd, int dfd)
399 {
400     int mfd;

402     /*
403      * Make fd be greater than 255 (the 32-bit stdio limit),
404      * or at least make it greater than 2 so that the
405      * program will work when spawned by init(1m).
406      * Also, if dfd is non-zero, dup the fd to be dfd.
407      */
408     if ((mfd = minfd) == 0)
409         mfd = set_minfd();
410     if (dfd > 0 || (0 <= fd && fd < mfd)) {
411         if (dfd <= 0)
412             dfd = mfd;
413         dfd = fcntl(fd, F_DUPFD, dfd);
414         (void) close(fd);
415         fd = dfd;
416     }
417     /*
418      * Mark it close-on-exec so any created process doesn't inherit it.
419      */
420     if (fd >= 0)
421         (void) fcntl(fd, F_SETFD, FD_CLOEXEC);
422     return (fd);
423 }

425 /*
426  * Create a new controlled process.
427  * Leave it stopped on successful exit from exec() or execve().
428  * Return an opaque pointer to its process control structure.
429  * Return NULL if process cannot be created (fork()/exec() not successful).
430  */
431 struct ps_prochandle *
432 Pcreate(const char *file,      /* executable file name */
433         char *const *argv,    /* argument vector */
434         char *const *envp,    /* environment */
435         int *perr,           /* pointer to error return code */
436         char *path,         /* if non-null, holds exec path name on return */
437         size_t len)        /* size of the path buffer */
438 {
439     char execpath[PATH_MAX];
440     char procname[PATH_MAX];
441     struct ps_prochandle *P;
442     pid_t pid;
443     int fd;
444     char *fname;
445     int rc;
446     int lasterrno = 0;

448     if (len == 0) /* zero length, no path */
449         path = NULL;
450     if (path != NULL)
451         *path = '\0';

453     if ((P = malloc(sizeof (struct ps_prochandle))) == NULL) {
454         *perr = C_STRANGE;

```

```

455         return (NULL);
456     }

458     if ((pid = fork1()) == -1) {
459         free(P);
460         *perr = C_FORK;
461         return (NULL);
462     }

464     if (pid == 0) { /* child process */
465         id_t id;
466         extern char **environ;

468         /*
469          * If running setuid or setgid, reset credentials to normal.
470          */
471         if ((id = getgid()) != getegid())
472             (void) setgid(id);
473         if ((id = getuid()) != geteuid())
474             (void) setuid(id);

476         Pcreate_callback(P); /* execute callback (see below) */
477         (void) pause(); /* wait for PRSABORT from parent */

479         /*
480          * This is ugly. There is no execvep() function that takes a
481          * path and an environment. We cheat here by replacing the
482          * global 'environ' variable right before we call this.
483          */
484         if (envp)
485             environ = (char **)envp;

487         (void) execvp(file, argv); /* execute the program */
488         _exit(127);
489     }

491     /*
492      * Initialize the process structure.
493      */
494     (void) memset(P, 0, sizeof (*P));
495     (void) mutex_init(&P->proc_lock, USYNC_THREAD, NULL);
496     P->flags |= CREATED;
497     P->state = PS_RUN;
498     P->pid = pid;
499     P->asfd = -1;
500     P->ctlfd = -1;
501     P->statfd = -1;
502     P->agentctlfd = -1;
503     P->agentstatfd = -1;
504     Pinit_ops(&P->ops, &P_live_ops);
505     Pinit_sym(P);

507     /*
508      * Open the /proc/pid files.
509      */
510     (void) snprintf(procname, sizeof (procname), "%s/%d",
511                    procs_path, (int)pid);
512     fname = procname + strlen(procname);
513     (void) set_minfd();

515     /*
516      * Exclusive write open advises others not to interfere.
517      * There is no reason for any of these open()s to fail.
518      */
519     (void) strcpy(fname, "as");
520     if ((fd = open(procname, (O_RDWR|O_EXCL))) < 0 ||

```

```

521     (fd = dupfd(fd, 0)) < 0) {
522         dprintf("Pcreate: failed to open %s: %s\n",
523             procname, strerror(errno));
524         rc = C_STRANGE;
525         goto bad;
526     }
527     P->asfd = fd;

529     (void) strcpy(fname, "status");
530     if ((fd = open(procname, O_RDONLY)) < 0 ||
531         (fd = dupfd(fd, 0)) < 0) {
532         dprintf("Pcreate: failed to open %s: %s\n",
533             procname, strerror(errno));
534         rc = C_STRANGE;
535         goto bad;
536     }
537     P->statfd = fd;

539     (void) strcpy(fname, "ctl");
540     if ((fd = open(procname, O_WRONLY)) < 0 ||
541         (fd = dupfd(fd, 0)) < 0) {
542         dprintf("Pcreate: failed to open %s: %s\n",
543             procname, strerror(errno));
544         rc = C_STRANGE;
545         goto bad;
546     }
547     P->ctlfd = fd;

549     (void) Pstop(P, 0);    /* stop the controlled process */

551     /*
552     * Wait for process to sleep in pause().
553     * If the process has already called pause(), then it should be
554     * stopped (PR_REQUESTED) while asleep in pause and we are done.
555     * Else we set up to catch entry/exit to pause() and set the process
556     * running again, expecting it to stop when it reaches pause().
557     * There is no reason for this to fail other than an interrupt.
558     */
559     (void) Psysentry(P, SYS_pause, 1);
560     (void) Psysexit(P, SYS_pause, 1);
561     for (;;) {
562         if (P->state == PS_STOP &&
563             P->status.pr_lwp.pr_syscall == SYS_pause &&
564             (P->status.pr_lwp.pr_why == PR_REQUESTED ||
565              P->status.pr_lwp.pr_why == PR_SYSENTRY ||
566              P->status.pr_lwp.pr_why == PR_SYSEXIT))
567             break;

569         if (P->state != PS_STOP || /* interrupt or process died */
570             Psetrun(P, 0, 0) != 0) { /* can't restart */
571             if (errno == EINTR || errno == ERESTART)
572                 rc = C_INTR;
573             else {
574                 dprintf("Pcreate: Psetrun failed: %s\n",
575                     strerror(errno));
576                 rc = C_STRANGE;
577             }
578             goto bad;
579         }

581         (void) Pwait(P, 0);
582     }
583     (void) Psysentry(P, SYS_pause, 0);
584     (void) Psysexit(P, SYS_pause, 0);

586     /*

```

```

587     * Kick the process off the pause() and catch
588     * it again on entry to exec() or exit().
589     */
590     (void) Psysentry(P, SYS_exit, 1);
591     (void) Psysentry(P, SYS_execve, 1);
592     if (Psetrun(P, 0, PRSABORT) == -1) {
593         dprintf("Pcreate: Psetrun failed: %s\n", strerror(errno));
594         rc = C_STRANGE;
595         goto bad;
596     }
597     (void) Pwait(P, 0);
598     if (P->state != PS_STOP) {
599         dprintf("Pcreate: Pwait failed: %s\n", strerror(errno));
600         rc = C_STRANGE;
601         goto bad;
602     }

604     /*
605     * Move the process through instances of failed exec()s
606     * to reach the point of stopped on successful exec().
607     */
608     (void) Psysexit(P, SYS_execve, TRUE);

610     while (P->state == PS_STOP &&
611            P->status.pr_lwp.pr_why == PR_SYSENTRY &&
612            P->status.pr_lwp.pr_what == SYS_execve) {
613         /*
614         * Fetch the exec path name now, before we complete
615         * the exec(). We may lose the process and be unable
616         * to get the information later.
617         */
618         (void) Pread_string(P, execpath, sizeof(execpath),
619             (off_t)P->status.pr_lwp.pr_sysarg[0]);
620         if (path != NULL)
621             (void) strncpy(path, execpath, len);
622         /*
623         * Set the process running and wait for
624         * it to stop on exit from the exec().
625         */
626         (void) Psetrun(P, 0, 0);
627         (void) Pwait(P, 0);

629         if (P->state == PS_LOST && /* we lost control */
630             Preopen(P) != 0) { /* and we can't get it back */
631             rc = C_PERM;
632             goto bad;
633         }

635         /*
636         * If the exec() failed, continue the loop, expecting
637         * there to be more attempts to exec(), based on PATH.
638         */
639         if (P->state == PS_STOP &&
640             P->status.pr_lwp.pr_why == PR_SYSEXIT &&
641             P->status.pr_lwp.pr_what == SYS_execve &&
642             (lasterrno = P->status.pr_lwp.pr_errno) != 0) {
643             /*
644             * The exec() failed. Set the process running and
645             * wait for it to stop on entry to the next exec().
646             */
647             (void) Psetrun(P, 0, 0);
648             (void) Pwait(P, 0);

650             continue;
651         }
652         break;

```

```

653     }
655     if (P->state == PS_STOP &&
656         P->status.pr_lwp.pr_why == PR_SYSEXIT &&
657         P->status.pr_lwp.pr_what == SYS_execve &&
658         P->status.pr_lwp.pr_errno == 0) {
659         /*
660          * The process is stopped on successful exec() or execve().
661          * Turn off all tracing flags and return success.
662          */
663         restore_tracing_flags(P);
664 #ifndef _LP64
665         /* We must be a 64-bit process to deal with a 64-bit process */
666         if (P->status.pr_dmodel == PR_MODEL_LP64) {
667             rc = C_LP64;
668             goto bad;
669         }
670 #endif
671         /*
672          * Set run-on-last-close so the controlled process
673          * runs even if we die on a signal.
674          */
675         (void) Psetflags(P, PR_RLC);
676         *perr = 0;
677         return (P);
678     }
680     rc = lasterrno == ENOENT ? C_NOENT : C_NOEXEC;
682 bad:
683     (void) kill(pid, SIGKILL);
684     if (path != NULL && rc != C_PERM && rc != C_LP64)
685         *path = '\0';
686     Pfree(P);
687     *perr = rc;
688     return (NULL);
689 }
691 struct ps_prochandle *
692 Pcreate(
693     const char *file,          /* executable file name */
694     char *const *argv,        /* argument vector */
695     int *perr,                /* pointer to error return code */
696     char *path,               /* if non-null, holds exec path name on return */
697     size_t len)               /* size of the path buffer */
698 {
699     return (Pcreate(file, argv, NULL, perr, path, len));
700 }
702 /*
703  * Return a printable string corresponding to a Pcreate() error return.
704  */
705 const char *
706 Pcreate_error(int error)
707 {
708     const char *str;
710     switch (error) {
711     case C_FORK:
712         str = "cannot fork";
713         break;
714     case C_PERM:
715         str = "file is set-id or unreadable";
716         break;
717     case C_NOEXEC:
718         str = "cannot execute file";

```

```

719         break;
720     case C_INTR:
721         str = "operation interrupted";
722         break;
723     case C_LP64:
724         str = "program is _LP64, self is not";
725         break;
726     case C_STRANGE:
727         str = "unanticipated system error";
728         break;
729     case C_NOENT:
730         str = "cannot find executable file";
731         break;
732     default:
733         str = "unknown error";
734         break;
735     }
737     return (str);
738 }
740 /*
741  * Callback to execute in each child process created with Pcreate() after fork
742  * but before it execs the new process image. By default, we do nothing, but
743  * by calling this function we allow the client program to define its own
744  * version of the function which will interpose on our empty default. This
745  * may be useful for clients that need to modify signal dispositions, terminal
746  * attributes, or process group and session properties for each new victim.
747  */
748 /*ARGSUSED*/
749 void
750 Pcreate_callback(struct ps_prochandle *P)
751 {
752     /* nothing to do here */
753 }
755 /*
756  * Grab an existing process.
757  * Return an opaque pointer to its process control structure.
758  */
759 * pid:          UNIX process ID.
760 * flags:
761 *   PGRAB_RETAIN   Retain tracing flags (default clears all tracing flags).
762 *   PGRAB_FORCE    Grab regardless of whether process is already traced.
763 *   PGRAB_RDONLY   Open the address space file O_RDONLY instead of O_RDWR,
764 *                   and do not open the process control file.
765 *   PGRAB_NOSTOP   Open the process but do not force it to stop.
766 * perr:          pointer to error return code.
767 */
768 struct ps_prochandle *
769 Pgrab(pid_t pid, int flags, int *perr)
770 {
771     struct ps_prochandle *P;
772     int fd, omode;
773     char procname[PATH_MAX];
774     char *fname;
775     int rc = 0;
777     /*
778      * PGRAB_RDONLY means that we do not open the /proc/<pid>/control file,
779      * and so it implies RETAIN and NOSTOP since both require control.
780      */
781     if (flags & PGRAB_RDONLY)
782         flags |= PGRAB_RETAIN | PGRAB_NOSTOP;
784     if ((P = malloc(sizeof (struct ps_prochandle))) == NULL) {

```

```

785         *perr = G_STRANGE;
786         return (NULL);
787     }

789     P->asfd = -1;
790     P->ctlfd = -1;
791     P->statfd = -1;

793 again: /* Come back here if we lose it in the Window of Vulnerability */
794 if (P->ctlfd >= 0)
795     (void) close(P->ctlfd);
796 if (P->asfd >= 0)
797     (void) close(P->asfd);
798 if (P->statfd >= 0)
799     (void) close(P->statfd);
800 (void) memset(P, 0, sizeof (*P));
801 (void) mutex_init(&P->proc_lock, USYNC_THREAD, NULL);
802 P->ctlfd = -1;
803 P->asfd = -1;
804 P->statfd = -1;
805 P->agentctlfd = -1;
806 P->agentstatfd = -1;
807 Pinit_ops(&P->ops, &P_live_ops);
808 Pinit_sym(P);

810 /*
811  * Open the /proc/pid files
812  */
813 (void) snprintf(procname, sizeof (procname), "%s/%d/",
814               procfs_path, (int)pid);
815 fname = procname + strlen(procname);
816 (void) set_minfd();

818 /*
819  * Request exclusive open to avoid grabbing someone else's
820  * process and to prevent others from interfering afterwards.
821  * If this fails and the 'PGRAB_FORCE' flag is set, attempt to
822  * open non-exclusively.
823  */
824 (void) strcpy(fname, "as");
825 omode = (flags & PGRAB_RDONLY) ? O_RDONLY : O_RDWR;

827 if (((fd = open(procname, omode | O_EXCL)) < 0 &&
828      (fd = ((flags & PGRAB_FORCE)? open(procname, omode) : -1)) < 0) ||
829      (fd = dupfd(fd, 0)) < 0) {
830     switch (errno) {
831     case ENOENT:
832         rc = G_NOPROC;
833         break;
834     case EACCES:
835     case EPERM:
836         rc = G_PERM;
837         break;
838     case EMFILE:
839         rc = G_NOFD;
840         break;
841     case EBUSY:
842         if (!(flags & PGRAB_FORCE) || geteuid() != 0) {
843             rc = G_BUSY;
844             break;
845         }
846         /* FALLTHROUGH */
847     default:
848         dprintf("Pgrab: failed to open %s: %s\n",
849               procname, strerror(errno));
850         rc = G_STRANGE;

```

```

851         break;
852     }
853     goto err;
854 }
855 P->asfd = fd;

857 (void) strcpy(fname, "status");
858 if ((fd = open(procname, O_RDONLY)) < 0 ||
859     (fd = dupfd(fd, 0)) < 0) {
860     switch (errno) {
861     case ENOENT:
862         rc = G_NOPROC;
863         break;
864     case EMFILE:
865         rc = G_NOFD;
866         break;
867     default:
868         dprintf("Pgrab: failed to open %s: %s\n",
869               procname, strerror(errno));
870         rc = G_STRANGE;
871         break;
872     }
873     goto err;
874 }
875 P->statfd = fd;

877 if (!(flags & PGRAB_RDONLY)) {
878     (void) strcpy(fname, "ctl");
879     if ((fd = open(procname, O_WRONLY)) < 0 ||
880         (fd = dupfd(fd, 0)) < 0) {
881         switch (errno) {
882         case ENOENT:
883             rc = G_NOPROC;
884             break;
885         case EMFILE:
886             rc = G_NOFD;
887             break;
888         default:
889             dprintf("Pgrab: failed to open %s: %s\n",
890                   procname, strerror(errno));
891             rc = G_STRANGE;
892             break;
893         }
894         goto err;
895     }
896     P->ctlfd = fd;
897 }

899 P->state = PS_RUN;
900 P->pid = pid;

902 /*
903  * We are now in the Window of Vulnerability (WoV). The process may
904  * exec() a setuid/setgid or unreadable object file between the open()
905  * and the PCSTOP. We will get EAGAIN in this case and must start over.
906  * As Pstopstatus will trigger the first read() from a /proc file,
907  * we also need to handle EOVERFLOW here when 32-bit as an indicator
908  * that this process is 64-bit. Finally, if the process has become
909  * a zombie (PS_UNDEAD) while we were trying to grab it, just remain
910  * silent about this and pretend there was no process.
911  */
912 if (Pstopstatus(P, PCNULL, 0) != 0) {
913 #ifdef _LP64
914     if (errno == EOVERFLOW) {
915         rc = G_LP64;
916         goto err;

```

```

917     }
918 #endif
919     if (P->state == PS_LOST) { /* WoV */
920         (void) mutex_destroy(&P->proc_lock);
921         goto again;
922     }
923
924     if (P->state == PS_UNDEAD)
925         rc = G_NOPROC;
926     else
927         rc = G_STRANGE;
928
929     goto err;
930 }
931
932 /*
933  * If the process is a system process, we can't control it even as root
934  */
935 if (P->status.pr_flags & PR_ISSYS) {
936     rc = G_SYS;
937     goto err;
938 }
939 #ifndef _LP64
940 /*
941  * We must be a 64-bit process to deal with a 64-bit process
942  */
943 if (P->status.pr_dmodel == PR_MODEL_LP64) {
944     rc = G_LP64;
945     goto err;
946 }
947 #endif
948
949 /*
950  * Remember the status for use by Prelease().
951  */
952 P->orig_status = P->status; /* structure copy */
953
954 /*
955  * Before stopping the process, make sure we are not grabbing ourselves.
956  * If we are, make sure we are doing it PGRAB_RDONLY.
957  */
958 if (pid == getpid()) {
959     /*
960      * Verify that the process is really ourself:
961      * Set a magic number, read it through the
962      * /proc file and see if the results match.
963      */
964     uint32_t magic1 = 0;
965     uint32_t magic2 = 2;
966
967     errno = 0;
968
969     if (Pread(P, &magic2, sizeof (magic2), (uintptr_t)&magic1)
970         == sizeof (magic2) &&
971         magic2 == 0 &&
972         (magic1 == 0xfeedbeef) &&
973         Pread(P, &magic2, sizeof (magic2), (uintptr_t)&magic1)
974         == sizeof (magic2) &&
975         magic2 == 0xfeedbeef &&
976         !(flags & PGRAB_RDONLY)) {
977         rc = G_SELF;
978         goto err;
979     }
980 }
981
982 /*

```

```

983  * If the process is already stopped or has been directed
984  * to stop via /proc, do not set run-on-last-close.
985  */
986 if (!(P->status.pr_lwp.pr_flags & (PR_ISTOP|PR_DSTOP)) &&
987     !(flags & PGRAB_RDONLY)) {
988     /*
989      * Mark the process run-on-last-close so
990      * it runs even if we die from SIGKILL.
991      */
992     if (Psetflags(P, PR_RLC) != 0) {
993         if (errno == EAGAIN) { /* WoV */
994             (void) mutex_destroy(&P->proc_lock);
995             goto again;
996         }
997         if (errno == ENOENT) /* No complaint about zombies */
998             rc = G_ZOMB;
999         else {
1000             dprintf("Pgrab: failed to set RLC\n");
1001             rc = G_STRANGE;
1002         }
1003         goto err;
1004     }
1005 }
1006
1007 /*
1008  * If a stop directive is pending and the process has not yet stopped,
1009  * then synchronously wait for the stop directive to take effect.
1010  * Limit the time spent waiting for the process to stop by iterating
1011  * at most 10 times. The time-out of 20 ms corresponds to the time
1012  * between sending the stop directive and the process actually stopped
1013  * as measured by DTrace on a slow, busy system. If the process doesn't
1014  * stop voluntarily, clear the PR_DSTOP flag so that the code below
1015  * forces the process to stop.
1016  */
1017 if (!(flags & PGRAB_RDONLY)) {
1018     int niter = 0;
1019     while ((P->status.pr_lwp.pr_flags & (PR_STOPPED|PR_DSTOP)) ==
1020         PR_DSTOP && niter < 10 &&
1021         Pstopstatus(P, PCTWSTOP, 20) != 0) {
1022         niter++;
1023         if (flags & PGRAB_NOSTOP)
1024             break;
1025     }
1026     if (niter == 10 && !(flags & PGRAB_NOSTOP)) {
1027         /* Try it harder down below */
1028         P->status.pr_lwp.pr_flags &= ~PR_DSTOP;
1029     }
1030 }
1031
1032 /*
1033  * If the process is not already stopped or directed to stop
1034  * and PGRAB_NOSTOP was not specified, stop the process now.
1035  */
1036 if (!(P->status.pr_lwp.pr_flags & (PR_ISTOP|PR_DSTOP)) &&
1037     !(flags & PGRAB_NOSTOP)) {
1038     /*
1039      * Stop the process, get its status and signal/syscall masks.
1040      */
1041     if (((P->status.pr_lwp.pr_flags & PR_STOPPED) &&
1042         Pstopstatus(P, PCDSTOP, 0) != 0) ||
1043         Pstopstatus(P, PCSTOP, 2000) != 0) {
1044 #ifndef _LP64
1045         if (errno == EOVERFLOW) {
1046             rc = G_LP64;
1047             goto err;
1048         }

```

```

1049 #endif
1050         if (P->state == PS_LOST) {           /* WoV */
1051             (void) mutex_destroy(&P->proc_lock);
1052             goto again;
1053         }
1054         if ((errno != EINTR && errno != ERESTART) ||
1055             (P->state != PS_STOP &&
1056              !(P->status.pr_flags & PR_DSTOP))) {
1057             if (P->state != PS_RUN && errno != ENOENT) {
1058                 dprintf("Pgrab: failed to PCSTOP\n");
1059                 rc = G_STRANGE;
1060             } else {
1061                 rc = G_ZOMB;
1062             }
1063             goto err;
1064         }
1065     }
1066
1067     /*
1068     * Process should now either be stopped via /proc or there
1069     * should be an outstanding stop directive.
1070     */
1071     if (!(P->status.pr_flags & (PR_ISTOP|PR_DSTOP))) {
1072         dprintf("Pgrab: process is not stopped\n");
1073         rc = G_STRANGE;
1074         goto err;
1075     }
1076 #ifndef _LP64
1077     /*
1078     * Test this again now because the 32-bit victim process may
1079     * have exec'd a 64-bit process in the meantime.
1080     */
1081     if (P->status.pr_dmodel == PR_MODEL_LP64) {
1082         rc = G_LP64;
1083         goto err;
1084     }
1085 #endif
1086 }
1087
1088 /*
1089 * Cancel all tracing flags unless the PGRAB_RETAIN flag is set.
1090 */
1091 if (!(flags & PGRAB_RETAIN)) {
1092     (void) Psysentry(P, 0, FALSE);
1093     (void) Psysexit(P, 0, FALSE);
1094     (void) Psignal(P, 0, FALSE);
1095     (void) Pfault(P, 0, FALSE);
1096     Psync(P);
1097 }
1098
1099 *perr = 0;
1100 return (P);
1101
1102 err:
1103     Pfree(P);
1104     *perr = rc;
1105     return (NULL);
1106 }
1107
1108 /*
1109 * Return a printable string corresponding to a Pgrab() error return.
1110 */
1111 const char *
1112 Pgrab_error(int error)
1113 {
1114     const char *str;

```

```

1116     switch (error) {
1117     case G_NOPROC:
1118         str = "no such process";
1119         break;
1120     case G_NOCORE:
1121         str = "no such core file";
1122         break;
1123     case G_NOPROCCORE:
1124         str = "no such process or core file";
1125         break;
1126     case G_NOEXEC:
1127         str = "cannot find executable file";
1128         break;
1129     case G_ZOMB:
1130         str = "zombie process";
1131         break;
1132     case G_PERM:
1133         str = "permission denied";
1134         break;
1135     case G_BUSY:
1136         str = "process is traced";
1137         break;
1138     case G_SYS:
1139         str = "system process";
1140         break;
1141     case G_SELF:
1142         str = "attempt to grab self";
1143         break;
1144     case G_INTR:
1145         str = "operation interrupted";
1146         break;
1147     case G_LP64:
1148         str = "program is _LP64, self is not";
1149         break;
1150     case G_FORMAT:
1151         str = "file is not an ELF core file";
1152         break;
1153     case G_elf:
1154         str = "libelf error";
1155         break;
1156     case G_NOTE:
1157         str = "core file is corrupt or missing required data";
1158         break;
1159     case G_STRANGE:
1160         str = "unanticipated system error";
1161         break;
1162     case G_ISAINVAL:
1163         str = "wrong ELF machine type";
1164         break;
1165     case G_BADLWPS:
1166         str = "bad lwp specification";
1167         break;
1168     case G_NOFD:
1169         str = "too many open files";
1170         break;
1171     default:
1172         str = "unknown error";
1173         break;
1174     }
1175
1176     return (str);
1177 }
1178
1179 /*
1180 * Free a process control structure.

```

```

1181 * Close the file descriptors but don't do the Prelease logic.
1182 */
1183 void
1184 Pfree(struct ps_prochandle *P)
1185 {
1186     uint_t i;
1187
1188     if (P->ucaddrs != NULL) {
1189         free(P->ucaddrs);
1190         P->ucaddrs = NULL;
1191         P->ucnelems = 0;
1192     }
1193
1194     (void) mutex_lock(&P->proc_lock);
1195     if (P->hashtab != NULL) {
1196         struct ps_lwphandle *L;
1197         for (i = 0; i < HASHSIZE; i++) {
1198             while ((L = P->hashtab[i]) != NULL)
1199                 Lfree_internal(P, L);
1200             free(P->hashtab);
1201         }
1202     }
1203
1204     while (P->num_fd > 0) {
1205         fd_info_t *fip = list_next(&P->fd_head);
1206         list_unlink(fip);
1207         free(fip);
1208         P->num_fd--;
1209     }
1210     (void) mutex_unlock(&P->proc_lock);
1211     (void) mutex_destroy(&P->proc_lock);
1212
1213     if (P->agentctldfd >= 0)
1214         (void) close(P->agentctldfd);
1215     if (P->agentstatfd >= 0)
1216         (void) close(P->agentstatfd);
1217     if (P->ctldfd >= 0)
1218         (void) close(P->ctldfd);
1219     if (P->asfd >= 0)
1220         (void) close(P->asfd);
1221     if (P->statfd >= 0)
1222         (void) close(P->statfd);
1223     Preset_maps(P);
1224     P->ops.pop_fini(P, P->data);
1225
1226     /* clear out the structure as a precaution against reuse */
1227     (void) memset(P, 0, sizeof (*P));
1228     P->ctldfd = -1;
1229     P->asfd = -1;
1230     P->statfd = -1;
1231     P->agentctldfd = -1;
1232     P->agentstatfd = -1;
1233
1234     free(P);
1235 }
1236
1237 /*
1238 * Return the state of the process, one of the PS_* values.
1239 */
1240 int
1241 Pstate(struct ps_prochandle *P)
1242 {
1243     return (P->state);
1244 }
1245
1246 /*

```

```

1247 * Return the open address space file descriptor for the process.
1248 * Clients must not close this file descriptor, not use it
1249 * after the process is freed.
1250 */
1251 int
1252 Pasfd(struct ps_prochandle *P)
1253 {
1254     return (P->asfd);
1255 }
1256
1257 /*
1258 * Return the open control file descriptor for the process.
1259 * Clients must not close this file descriptor, not use it
1260 * after the process is freed.
1261 */
1262 int
1263 Pctldfd(struct ps_prochandle *P)
1264 {
1265     return (P->ctldfd);
1266 }
1267
1268 /*
1269 * Return a pointer to the process psinfo structure.
1270 * Clients should not hold on to this pointer indefinitely.
1271 * It will become invalid on Prelease().
1272 */
1273 const psinfo_t *
1274 Ppsinfo(struct ps_prochandle *P)
1275 {
1276     return (P->ops.pop_psinfo(P, &P->psinfo, P->data));
1277 }
1278
1279 /*
1280 * Return a pointer to the process status structure.
1281 * Clients should not hold on to this pointer indefinitely.
1282 * It will become invalid on Prelease().
1283 */
1284 const pstatus_t *
1285 Pstatus(struct ps_prochandle *P)
1286 {
1287     return (&P->status);
1288 }
1289
1290 static void
1291 Pread_status(struct ps_prochandle *P)
1292 {
1293     P->ops.pop_status(P, &P->status, P->data);
1294 }
1295
1296 /*
1297 * Fill in a pointer to a process credentials structure. The ngroups parameter
1298 * is the number of supplementary group entries allocated in the caller's cred
1299 * structure. It should equal zero or one unless extra space has been
1300 * allocated for the group list by the caller.
1301 */
1302 int
1303 Pcred(struct ps_prochandle *P, pcrd_t *pcrd, int ngroups)
1304 {
1305     return (P->ops.pop_cred(P, pcrd, ngroups, P->data));
1306 }
1307
1308 /* Return an allocated prsecflags_t */
1309 int
1310 Psecflags(struct ps_prochandle *P, prsecflags_t **psf)
1311 {
1312     int ret;

```

```

1314     if ((ret = P->ops.pop_secflags(P, psf, P->data)) == 0) {
1315         if ((*psf)->pr_version != PRSECFLAGS_VERSION_1) {
1316             errno = EINVAL;
1317             return (-1);
1318         }
1319     }
1321     return (ret);
1322 }

1324 void
1325 Psecflags_free(prsecflags_t *psf)
1326 {
1327     free(psf);
1328 }

1330 #endif /* ! codereview */
1331 static prheader_t *
1332 Plstatus(struct ps_prochandle *P)
1333 {
1334     return (P->ops.pop_lstatus(P, P->data));
1335 }

1337 static prheader_t *
1338 Plpsinfo(struct ps_prochandle *P)
1339 {
1340     return (P->ops.pop_lpsinfo(P, P->data));
1341 }

1344 #if defined(__i386) || defined(__amd64)
1345 /*
1346  * Fill in a pointer to a process LDT structure.
1347  * The caller provides a buffer of size 'nldt * sizeof (struct ssd)';
1348  * If pldt == NULL or nldt == 0, we return the number of existing LDT entries.
1349  * Otherwise we return the actual number of LDT entries fetched (<= nldt).
1350  */
1351 int
1352 Pldt(struct ps_prochandle *P, struct ssd *pldt, int nldt)
1353 {
1354     return (P->ops.pop_ldt(P, pldt, nldt, P->data));
1355 }

1356 #endif /* __i386 */

1359 /* ARGSUSED */
1360 void
1361 Ppriv_free(struct ps_prochandle *P, prpriv_t *prv)
1362 {
1363     free(prv);
1364 }

1366 /*
1367  * Return a malloced process privilege structure in *pprv.
1368  */
1369 int
1370 Ppriv(struct ps_prochandle *P, prpriv_t **pprv)
1371 {
1372     return (P->ops.pop_priv(P, pprv, P->data));
1373 }

1375 int
1376 Psetpriv(struct ps_prochandle *P, prpriv_t *pprv)
1377 {
1378     int rc;

```

```

1379     long *ctl;
1380     size_t sz;

1382     if (P->state == PS_DEAD) {
1383         errno = EBADF;
1384         return (-1);
1385     }

1387     sz = PRIV_PRPRIV_SIZE(pprv) + sizeof (long);

1389     sz = ((sz - 1) / sizeof (long) + 1) * sizeof (long);

1391     ctl = malloc(sz);
1392     if (ctl == NULL)
1393         return (-1);

1395     ctl[0] = PCSPRIV;

1397     (void) memcpy(&ctl[1], pprv, PRIV_PRPRIV_SIZE(pprv));

1399     if (write(P->ctlfd, ctl, sz) != sz)
1400         rc = -1;
1401     else
1402         rc = 0;

1404     free(ctl);

1406     return (rc);
1407 }

1409 void *
1410 Pprivinfo(struct ps_prochandle *P)
1411 {
1412     core_info_t *core = P->data;

1414     /* Use default from libc */
1415     if (P->state != PS_DEAD)
1416         return (NULL);

1418     return (core->core_privinfo);
1419 }

1421 /*
1422  * Ensure that all cached state is written to the process.
1423  * The cached state is the LWP's signal mask and registers
1424  * and the process's tracing flags.
1425  */
1426 void
1427 Psync(struct ps_prochandle *P)
1428 {
1429     int ctlfd = (P->agentctlfd >= 0)? P->agentctlfd : P->ctlfd;
1430     long cmd[6];
1431     iovec_t iov[12];
1432     int n = 0;

1434     if (P->flags & SETHOLD) {
1435         cmd[0] = PCSHOLD;
1436         iov[n].iov_base = (caddr_t)&cmd[0];
1437         iov[n+1].iov_len = sizeof (long);
1438         iov[n].iov_base = (caddr_t)&P->status.pr_lwp.pr_lwphold;
1439         iov[n+1].iov_len = sizeof (P->status.pr_lwp.pr_lwphold);
1440     }
1441     if (P->flags & SETREGS) {
1442         cmd[1] = PCSREG;
1443 #ifdef __i386
1444         /* XX64 we should probably restore REG_GS after this */

```



```

1445         if (ctlfd == P->agentctlfd)
1446             P->status.pr_lwp.pr_reg[GS] = 0;
1447 #elif defined(__amd64)
1448     /* XX64 */
1449 #endif
1450     iov[n].iov_base = (caddr_t)&cmd[1];
1451     iov[n+1].iov_len = sizeof (long);
1452     iov[n].iov_base = (caddr_t)&P->status.pr_lwp.pr_reg[0];
1453     iov[n+1].iov_len = sizeof (P->status.pr_lwp.pr_reg);
1454 }
1455 if (P->flags & SETSIG) {
1456     cmd[2] = PCSTRACE;
1457     iov[n].iov_base = (caddr_t)&cmd[2];
1458     iov[n+1].iov_len = sizeof (long);
1459     iov[n].iov_base = (caddr_t)&P->status.pr_sigtrace;
1460     iov[n+1].iov_len = sizeof (P->status.pr_sigtrace);
1461 }
1462 if (P->flags & SETFAULT) {
1463     cmd[3] = PCSFAULT;
1464     iov[n].iov_base = (caddr_t)&cmd[3];
1465     iov[n+1].iov_len = sizeof (long);
1466     iov[n].iov_base = (caddr_t)&P->status.pr_fltrtrace;
1467     iov[n+1].iov_len = sizeof (P->status.pr_fltrtrace);
1468 }
1469 if (P->flags & SETENTRY) {
1470     cmd[4] = PCSEENTRY;
1471     iov[n].iov_base = (caddr_t)&cmd[4];
1472     iov[n+1].iov_len = sizeof (long);
1473     iov[n].iov_base = (caddr_t)&P->status.pr_sysentry;
1474     iov[n+1].iov_len = sizeof (P->status.pr_sysentry);
1475 }
1476 if (P->flags & SETEXIT) {
1477     cmd[5] = PCSEXIT;
1478     iov[n].iov_base = (caddr_t)&cmd[5];
1479     iov[n+1].iov_len = sizeof (long);
1480     iov[n].iov_base = (caddr_t)&P->status.pr_sysexit;
1481     iov[n+1].iov_len = sizeof (P->status.pr_sysexit);
1482 }
1484 if (n == 0 || writev(ctlfd, iov, n) < 0)
1485     return; /* nothing to do or write failed */
1487 P->flags &= ~(SETSIG|SETFAULT|SETENTRY|SETEXIT|SETHOLD|SETREGS);
1488 }
1490 /*
1491  * Reopen the /proc file (after PS_LOST).
1492  */
1493 int
1494 Preopen(struct ps_prochandle *P)
1495 {
1496     int fd;
1497     char procname[PATH_MAX];
1498     char *fname;
1500     if (P->state == PS_DEAD || P->state == PS_IDLE)
1501         return (0);
1503     if (P->agentcnt > 0) {
1504         P->agentcnt = 1;
1505         Pdestroy_agent(P);
1506     }
1508     (void) snprintf(procname, sizeof (procname), "%s/%d/",
1509         procEs_path, (int)P->pid);
1510     fname = procname + strlen(procname);

```

```

1512     (void) strcpy(fname, "as");
1513     if ((fd = open(procname, O_RDWR)) < 0 ||
1514         close(P->asfd) < 0 ||
1515         (fd = dupfd(fd, P->asfd)) != P->asfd) {
1516         dprintf("Preopen: failed to open %s: %s\n",
1517             procname, strerror(errno));
1518         if (fd >= 0)
1519             (void) close(fd);
1520         return (-1);
1521     }
1522     P->asfd = fd;
1524     (void) strcpy(fname, "status");
1525     if ((fd = open(procname, O_RDONLY)) < 0 ||
1526         close(P->statfd) < 0 ||
1527         (fd = dupfd(fd, P->statfd)) != P->statfd) {
1528         dprintf("Preopen: failed to open %s: %s\n",
1529             procname, strerror(errno));
1530         if (fd >= 0)
1531             (void) close(fd);
1532         return (-1);
1533     }
1534     P->statfd = fd;
1536     (void) strcpy(fname, "ctl");
1537     if ((fd = open(procname, O_WRONLY)) < 0 ||
1538         close(P->ctlfd) < 0 ||
1539         (fd = dupfd(fd, P->ctlfd)) != P->ctlfd) {
1540         dprintf("Preopen: failed to open %s: %s\n",
1541             procname, strerror(errno));
1542         if (fd >= 0)
1543             (void) close(fd);
1544         return (-1);
1545     }
1546     P->ctlfd = fd;
1548     /*
1549     * Set the state to PS_RUN and wait for the process to stop so that
1550     * we re-read the status from the new P->statfd. If this fails, Pwait
1551     * will reset the state to PS_LOST and we fail the reopen. Before
1552     * returning, we also forge a bit of P->status to allow the debugger to
1553     * see that we are PS_LOST following a successful exec.
1554     */
1555     P->state = PS_RUN;
1556     if (Pwait(P, 0) == -1) {
1557 #ifdef _ILP32
1558         if (errno == EOVERFLOW)
1559             P->status.pr_dmodel = PR_MODEL_LP64;
1560 #endif
1561         P->status.pr_lwp.pr_why = PR_SYSEXIT;
1562         P->status.pr_lwp.pr_what = SYS_execve;
1563         P->status.pr_lwp.pr_errno = 0;
1564         return (-1);
1565     }
1567     /*
1568     * The process should be stopped on exec (REQUESTED)
1569     * or else should be stopped on exit from exec() (SYSEXIT)
1570     */
1571     if (P->state == PS_STOP &&
1572         (P->status.pr_lwp.pr_why == PR_REQUESTED ||
1573          (P->status.pr_lwp.pr_why == PR_SYSEXIT &&
1574           P->status.pr_lwp.pr_what == SYS_execve))) {
1575         /* fake up stop-on-exit-from-execve */
1576         if (P->status.pr_lwp.pr_why == PR_REQUESTED) {

```

```

1577         P->status.pr_lwp.pr_why = PR_SYSEXIT;
1578         P->status.pr_lwp.pr_what = SYS_execve;
1579         P->status.pr_lwp.pr_errno = 0;
1580     }
1581     } else {
1582         dprintf("Preopen: expected REQUESTED or "
1583             "SYSEXIT(SYS_execve) stop\n");
1584     }
1585
1586     return (0);
1587 }
1588
1589 /*
1590  * Define all settable flags other than the microstate accounting flags.
1591  */
1592 #define ALL_SETTABLE_FLAGS (PR_FORK|PR_RLC|PR_KLC|PR_ASYNC|PR_BPTADJ|PR_PTRACE)
1593
1594 /*
1595  * Restore /proc tracing flags to their original values
1596  * in preparation for releasing the process.
1597  * Also called by Pcreate() to clear all tracing flags.
1598  */
1599 static void
1600 restore_tracing_flags(struct ps_prochandle *P)
1601 {
1602     long flags;
1603     long cmd[4];
1604     iovec_t iov[8];
1605
1606     if (P->flags & CREATED) {
1607         /* we created this process; clear all tracing flags */
1608         preemptset(&P->status.pr_sigtrace);
1609         preemptset(&P->status.pr_fltrtrace);
1610         preemptset(&P->status.pr_sysentry);
1611         preemptset(&P->status.pr_sysexit);
1612         if ((P->status.pr_flags & ALL_SETTABLE_FLAGS) != 0)
1613             (void) Punsetflags(P, ALL_SETTABLE_FLAGS);
1614     } else {
1615         /* we grabbed the process; restore its tracing flags */
1616         P->status.pr_sigtrace = P->orig_status.pr_sigtrace;
1617         P->status.pr_fltrtrace = P->orig_status.pr_fltrtrace;
1618         P->status.pr_sysentry = P->orig_status.pr_sysentry;
1619         P->status.pr_sysexit = P->orig_status.pr_sysexit;
1620         if ((P->status.pr_flags & ALL_SETTABLE_FLAGS) !=
1621             (flags = (P->orig_status.pr_flags & ALL_SETTABLE_FLAGS))) {
1622             (void) Punsetflags(P, ALL_SETTABLE_FLAGS);
1623             if (flags)
1624                 (void) Psetflags(P, flags);
1625         }
1626     }
1627
1628     cmd[0] = PCSTRACE;
1629     iov[0].iov_base = (caddr_t)&cmd[0];
1630     iov[0].iov_len = sizeof (long);
1631     iov[1].iov_base = (caddr_t)&P->status.pr_sigtrace;
1632     iov[1].iov_len = sizeof (P->status.pr_sigtrace);
1633
1634     cmd[1] = PCSFAULT;
1635     iov[2].iov_base = (caddr_t)&cmd[1];
1636     iov[2].iov_len = sizeof (long);
1637     iov[3].iov_base = (caddr_t)&P->status.pr_fltrtrace;
1638     iov[3].iov_len = sizeof (P->status.pr_fltrtrace);
1639
1640     cmd[2] = PCSENTRY;
1641     iov[4].iov_base = (caddr_t)&cmd[2];
1642     iov[4].iov_len = sizeof (long);

```

```

1643     iov[5].iov_base = (caddr_t)&P->status.pr_sysentry;
1644     iov[5].iov_len = sizeof (P->status.pr_sysentry);
1645
1646     cmd[3] = PCSEXIT;
1647     iov[6].iov_base = (caddr_t)&cmd[3];
1648     iov[6].iov_len = sizeof (long);
1649     iov[7].iov_base = (caddr_t)&P->status.pr_sysexit;
1650     iov[7].iov_len = sizeof (P->status.pr_sysexit);
1651
1652     (void) writev(P->ctldfd, iov, 8);
1653
1654     P->flags &= ~(SETSIG|SETFAULT|SETENTRY|SETEXIT);
1655 }
1656
1657 /*
1658  * Release the process. Frees the process control structure.
1659  * flags:
1660  * PRELEASE_CLEAR Clear all tracing flags.
1661  * PRELEASE_RETAIN Retain current tracing flags.
1662  * PRELEASE_HANG Leave the process stopped and abandoned.
1663  * PRELEASE_KILL Terminate the process with SIGKILL.
1664  */
1665 void
1666 Prelease(struct ps_prochandle *P, int flags)
1667 {
1668     if (P->state == PS_DEAD) {
1669         dprintf("Prelease: releasing handle %p PS_DEAD of pid %d\n",
1670             (void *)P, (int)P->pid);
1671         Pfree(P);
1672         return;
1673     }
1674
1675     if (P->state == PS_IDLE) {
1676         file_info_t *fptr = list_next(&P->file_head);
1677         dprintf("Prelease: releasing handle %p PS_IDLE of file %s\n",
1678             (void *)P, fptr->file_pname);
1679         Pfree(P);
1680         return;
1681     }
1682
1683     dprintf("Prelease: releasing handle %p pid %d\n",
1684         (void *)P, (int)P->pid);
1685
1686     if (P->ctldfd == -1) {
1687         Pfree(P);
1688         return;
1689     }
1690
1691     if (P->agentcnt > 0) {
1692         P->agentcnt = 1;
1693         Pdestroy_agent(P);
1694     }
1695
1696     /*
1697      * Attempt to stop the process.
1698      */
1699     P->state = PS_RUN;
1700     (void) Pstop(P, 1000);
1701
1702     if (flags & PRELEASE_KILL) {
1703         if (P->state == PS_STOP)
1704             (void) Psetrun(P, SIGKILL, 0);
1705         (void) kill(P->pid, SIGKILL);
1706         Pfree(P);
1707         return;
1708     }

```

```

1710  /*
1711  * If we lost control, all we can do now is close the files.
1712  * In this case, the last close sets the process running.
1713  */
1714  if (P->state != PS_STOP &&
1715      (P->status.pr_lwp.pr_flags & (PR_ISTOP|PR_DSTOP)) == 0) {
1716      Pfree(P);
1717      return;
1718  }

1720  /*
1721  * We didn't lose control; we do more.
1722  */
1723  Psync(P);

1725  if (flags & PRELEASE_CLEAR)
1726      P->flags |= CREATED;

1728  if (!(flags & PRELEASE_RETAIN))
1729      restore_tracing_flags(P);

1731  if (flags & PRELEASE_HANG) {
1732      /* Leave the process stopped and abandoned */
1733      (void) Punsetflags(P, PR_RLC|PR_KLC);
1734      Pfree(P);
1735      return;
1736  }

1738  /*
1739  * Set the process running if we created it or if it was
1740  * not originally stopped or directed to stop via /proc
1741  * or if we were given the PRELEASE_CLEAR flag.
1742  */
1743  if ((P->flags & CREATED) ||
1744      (P->orig_status.pr_lwp.pr_flags & (PR_ISTOP|PR_DSTOP)) == 0) {
1745      (void) Psetflags(P, PR_RLC);
1746      /*
1747       * We do this repeatedly because the process may have
1748       * more than one LWP stopped on an event of interest.
1749       * This makes sure all of them are set running.
1750       */
1751      do {
1752          if (Psetrun(P, 0, 0) == -1 && errno == EBUSY)
1753              break; /* Agent LWP may be stuck */
1754      } while (Pstopstatus(P, PCNULL, 0) == 0 &&
1755              P->status.pr_lwp.pr_flags & (PR_ISTOP|PR_DSTOP));

1757      if (P->status.pr_lwp.pr_flags & (PR_ISTOP|PR_DSTOP))
1758          dprintf("Prelease: failed to set process running\n");
1759  }

1761  Pfree(P);
1762  }

1764  /* debugging */
1765  void
1766  prldump(const char *caller, lwpstatus_t *lsp)
1767  {
1768      char name[32];
1769      uint32_t bits;

1771      switch (lsp->pr_why) {
1772      case PR_REQUESTED:
1773          dprintf("%s: REQUESTED\n", caller);
1774          break;

```

```

1775      case PR_SIGNALED:
1776          dprintf("%s: SIGNALED %s\n", caller,
1777                proc_signame(lsp->pr_what, name, sizeof (name)));
1778          break;
1779      case PR_FAULTED:
1780          dprintf("%s: FAULTED %s\n", caller,
1781                proc_fltname(lsp->pr_what, name, sizeof (name)));
1782          break;
1783      case PR_SYSENTRY:
1784          dprintf("%s: SYSENTRY %s\n", caller,
1785                proc_sysname(lsp->pr_what, name, sizeof (name)));
1786          break;
1787      case PR_SYSEXIT:
1788          dprintf("%s: SYSEXIT %s\n", caller,
1789                proc_sysname(lsp->pr_what, name, sizeof (name)));
1790          break;
1791      case PR_JOBCONTROL:
1792          dprintf("%s: JOBCONTROL %s\n", caller,
1793                proc_signame(lsp->pr_what, name, sizeof (name)));
1794          break;
1795      case PR_SUSPENDED:
1796          dprintf("%s: SUSPENDED\n", caller);
1797          break;
1798      default:
1799          dprintf("%s: Unknown\n", caller);
1800          break;
1801  }

1803  if (lsp->pr_cursig)
1804      dprintf("%s: p_cursig = %d\n", caller, lsp->pr_cursig);

1806  bits = *((uint32_t *)&lsp->pr_lwppend);
1807  if (bits)
1808      dprintf("%s: pr_lwppend = 0x%.8X\n", caller, bits);
1809  }

1811  /* debugging */
1812  static void
1813  prdump(struct ps_prochandle *P)
1814  {
1815      uint32_t bits;

1817      prldump("Pstopstatus", &P->status.pr_lwp);

1819      bits = *((uint32_t *)&P->status.pr_sigpend);
1820      if (bits)
1821          dprintf("Pstopstatus: pr_sigpend = 0x%.8X\n", bits);
1822  }

1824  /*
1825  * Wait for the specified process to stop or terminate.
1826  * Or, just get the current status (PCNULL).
1827  * Or, direct it to stop and get the current status (PCDSTOP).
1828  * If the agent LWP exists, do these things to the agent,
1829  * else do these things to the process as a whole.
1830  */
1831  int
1832  Pstopstatus(struct ps_prochandle *P,
1833              long request, /* PCNULL, PCDSTOP, PCSTOP, PCWSTOP */
1834              uint_t msec) /* if non-zero, timeout in milliseconds */
1835  {
1836      int ctlfd = (P->agentctlfd >= 0)? P->agentctlfd : P->ctlfd;
1837      long ctl[3];
1838      ssize_t rc;
1839      int err;
1840      int old_state = P->state;

```

```

1842     switch (P->state) {
1843     case PS_RUN:
1844         break;
1845     case PS_STOP:
1846         if (request != PCNULL && request != PCDSTOP)
1847             return (0);
1848         break;
1849     case PS_LOST:
1850         if (request != PCNULL) {
1851             errno = EAGAIN;
1852             return (-1);
1853         }
1854         break;
1855     case PS_UNDEAD:
1856     case PS_DEAD:
1857     case PS_IDLE:
1858         if (request != PCNULL) {
1859             errno = ENOENT;
1860             return (-1);
1861         }
1862         break;
1863     default: /* corrupted state */
1864         dprintf("Pstopstatus: corrupted state: %d\n", P->state);
1865         errno = EINVAL;
1866         return (-1);
1867     }

1869     ctl[0] = PCDSTOP;
1870     ctl[1] = PCTWSTOP;
1871     ctl[2] = (long)msec;
1872     rc = 0;
1873     switch (request) {
1874     case PCSTOP:
1875         rc = write(ctlfd, &ctl[0], 3*sizeof (long));
1876         break;
1877     case PCWSTOP:
1878         rc = write(ctlfd, &ctl[1], 2*sizeof (long));
1879         break;
1880     case PCDSTOP:
1881         rc = write(ctlfd, &ctl[0], 1*sizeof (long));
1882         break;
1883     case PCNULL:
1884         if (P->state == PS_DEAD || P->state == PS_IDLE)
1885             return (0);
1886         break;
1887     default: /* programming error */
1888         errno = EINVAL;
1889         return (-1);
1890     }
1891     err = (rc < 0)? errno : 0;
1892     Psync(P);

1894     if (P->agentstatfd < 0) {
1895         if (pread(P->statfd, &P->status,
1896             sizeof (P->status), (off_t)0) < 0)
1897             err = errno;
1898     } else {
1899         if (pread(P->agentstatfd, &P->status.pr_lwp,
1900             sizeof (P->status.pr_lwp), (off_t)0) < 0)
1901             err = errno;
1902         P->status.pr_flags = P->status.pr_lwp.pr_flags;
1903     }

1905     if (err) {
1906         switch (err) {

```

```

1907     case EINTR: /* user typed ctl-C */
1908     case ERESTART:
1909         dprintf("Pstopstatus: EINTR\n");
1910         break;
1911     case EAGAIN: /* we lost control of the the process */
1912     case EOVERFLOW:
1913         dprintf("Pstopstatus: PS_LOST, errno=%d\n", err);
1914         P->state = PS_LOST;
1915         break;
1916     default: /* check for dead process */
1917         if (_libproc_debug) {
1918             const char *errstr;

1920             switch (request) {
1921             case PCNULL:
1922                 errstr = "Pstopstatus PCNULL"; break;
1923             case PCSTOP:
1924                 errstr = "Pstopstatus PCSTOP"; break;
1925             case PCDSTOP:
1926                 errstr = "Pstopstatus PCDSTOP"; break;
1927             case PCWSTOP:
1928                 errstr = "Pstopstatus PCWSTOP"; break;
1929             default:
1930                 errstr = "Pstopstatus PC???"; break;
1931             }
1932             dprintf("%s: %s\n", errstr, strerror(err));
1933         }
1934         deadcheck(P);
1935         break;
1936     }
1937     if (err != EINTR && err != ERESTART) {
1938         errno = err;
1939         return (-1);
1940     }
1941     }

1943     if (!(P->status.pr_flags & PR_STOPPED)) {
1944         P->state = PS_RUN;
1945         if (request == PCNULL || request == PCDSTOP || msec != 0)
1946             return (0);
1947         dprintf("Pstopstatus: process is not stopped\n");
1948         errno = EPROTO;
1949         return (-1);
1950     }

1952     P->state = PS_STOP;

1954     if (_libproc_debug) /* debugging */
1955         prdump(P);

1957     /*
1958     * If the process was already stopped coming into Pstopstatus(),
1959     * then don't use its PC to set P->sysaddr since it may have been
1960     * changed since the time the process originally stopped.
1961     */
1962     if (old_state == PS_STOP)
1963         return (0);

1965     switch (P->status.pr_lwp.pr_why) {
1966     case PR_SYSENTRY:
1967     case PR_SYSEXIT:
1968         if (Pissyscall_prev(P, P->status.pr_lwp.pr_reg[R_PC],
1969             &P->sysaddr) == 0)
1970             P->sysaddr = P->status.pr_lwp.pr_reg[R_PC];
1971         break;
1972     case PR_REQUESTED:

```

```

1973     case PR_SIGNALED:
1974     case PR_FAULTED:
1975     case PR_JOBCONTROL:
1976     case PR_SUSPENDED:
1977         break;
1978     default:
1979         errno = EPROTO;
1980         return (-1);
1981     }
1982
1983     return (0);
1984 }
1985
1986 /*
1987  * Wait for the process to stop for any reason.
1988  */
1989 int
1990 Pwait(struct ps_prochandle *P, uint_t msec)
1991 {
1992     return (Pstopstatus(P, PCWSTOP, msec));
1993 }
1994
1995 /*
1996  * Direct the process to stop; wait for it to stop.
1997  */
1998 int
1999 Pstop(struct ps_prochandle *P, uint_t msec)
2000 {
2001     return (Pstopstatus(P, PCSTOP, msec));
2002 }
2003
2004 /*
2005  * Direct the process to stop; don't wait.
2006  */
2007 int
2008 Pdstop(struct ps_prochandle *P)
2009 {
2010     return (Pstopstatus(P, PCDSTOP, 0));
2011 }
2012
2013 static void
2014 deadcheck(struct ps_prochandle *P)
2015 {
2016     int fd;
2017     void *buf;
2018     size_t size;
2019
2020     if (P->statfd < 0)
2021         P->state = PS_UNDEAD;
2022     else {
2023         if (P->agentstatfd < 0) {
2024             fd = P->statfd;
2025             buf = &P->status;
2026             size = sizeof (P->status);
2027         } else {
2028             fd = P->agentstatfd;
2029             buf = &P->status.pr_lwp;
2030             size = sizeof (P->status.pr_lwp);
2031         }
2032         while (pread(fd, buf, size, (off_t)0) != size) {
2033             switch (errno) {
2034             default:
2035                 P->state = PS_UNDEAD;
2036                 break;
2037             case EINTR:
2038             case ERESTART:

```

```

2039                 continue;
2040             case EAGAIN:
2041                 P->state = PS_LOST;
2042                 break;
2043             }
2044             break;
2045         }
2046         P->status.pr_flags = P->status.pr_lwp.pr_flags;
2047     }
2048 }
2049
2050 /*
2051  * Get the value of one register from stopped process.
2052  */
2053 int
2054 Pgetareg(struct ps_prochandle *P, int regno, pgreg_t *preg)
2055 {
2056     if (regno < 0 || regno >= NPRGREG) {
2057         errno = EINVAL;
2058         return (-1);
2059     }
2060
2061     if (P->state == PS_IDLE) {
2062         errno = ENODATA;
2063         return (-1);
2064     }
2065
2066     if (P->state != PS_STOP && P->state != PS_DEAD) {
2067         errno = EBUSY;
2068         return (-1);
2069     }
2070
2071     *preg = P->status.pr_lwp.pr_reg[regno];
2072     return (0);
2073 }
2074
2075 /*
2076  * Put value of one register into stopped process.
2077  */
2078 int
2079 Pputareg(struct ps_prochandle *P, int regno, pgreg_t reg)
2080 {
2081     if (regno < 0 || regno >= NPRGREG) {
2082         errno = EINVAL;
2083         return (-1);
2084     }
2085
2086     if (P->state != PS_STOP) {
2087         errno = EBUSY;
2088         return (-1);
2089     }
2090
2091     P->status.pr_lwp.pr_reg[regno] = reg;
2092     P->flags |= SETREGS; /* set registers before continuing */
2093     return (0);
2094 }
2095
2096 int
2097 Psetrun(struct ps_prochandle *P,
2098         int sig, /* signal to pass to process */
2099         int flags) /* PRSTEP|PRSAABORT|PRSTOP|PRCSIG|PRCFAULT */
2100 {
2101     int ctlfd = (P->agentctlfd >= 0) ? P->agentctlfd : P->ctlfd;
2102     int sbits = (PR_DSTOP | PR_ISTOP | PR_ASLEEP);
2103
2104     long ctl[1 + /* PCCFAULT */

```

```

2105     1 + sizeof (siginfo_t)/sizeof (long) + /* PCSSIG/PCCSIG */
2106     2 ]; /* PCRUN */

2108 long *ctlp = ctl;
2109 size_t size;

2111 if (P->state != PS_STOP && (P->status.pr_lwp.pr_flags & sbits) == 0) {
2112     errno = EBUSY;
2113     return (-1);
2114 }

2116 Psync(P); /* flush tracing flags and registers */

2118 if (flags & PRCFAULT) { /* clear current fault */
2119     *ctlp++ = PCCFAULT;
2120     flags &= ~PRCFAULT;
2121 }

2123 if (flags & PRCSIG) { /* clear current signal */
2124     *ctlp++ = PCCSIG;
2125     flags &= ~PRCSIG;
2126 } else if (sig && sig != P->status.pr_lwp.pr_cursig) {
2127     /* make current signal */
2128     siginfo_t *infop;

2130     *ctlp++ = PCSSIG;
2131     infop = (siginfo_t *)ctlp;
2132     (void) memset(infop, 0, sizeof (*infop));
2133     infop->si_signo = sig;
2134     ctlp += sizeof (siginfo_t) / sizeof (long);
2135 }

2137 *ctlp++ = PCRUN;
2138 *ctlp++ = flags;
2139 size = (char *)ctlp - (char *)ctl;

2141 P->info_valid = 0; /* will need to update map and file info */

2143 /*
2144  * If we've cached ucontext-list information while we were stopped,
2145  * free it now.
2146  */
2147 if (P->ucaddrs != NULL) {
2148     free(P->ucaddrs);
2149     P->ucaddrs = NULL;
2150     P->ucnelems = 0;
2151 }

2153 if (write(ctlfd, ctl, size) != size) {
2154     /* If it is dead or lost, return the real status, not PS_RUN */
2155     if (errno == ENOENT || errno == EAGAIN) {
2156         (void) Pstopstatus(P, PCNULL, 0);
2157         return (0);
2158     }
2159     /* If it is not in a jobcontrol stop, issue an error message */
2160     if (errno != EBUSY ||
2161         P->status.pr_lwp.pr_why != PR_JOBCONTROL) {
2162         dprintf("Psetrun: %s\n", strerror(errno));
2163         return (-1);
2164     }
2165     /* Otherwise pretend that the job-stopped process is running */
2166 }

2168 P->state = PS_RUN;
2169 return (0);
2170 }

```

```

2172 ssize_t
2173 Pread(struct ps_prochandle *P,
2174     void *buf, /* caller's buffer */
2175     size_t nbyte, /* number of bytes to read */
2176     uintptr_t address) /* address in process */
2177 {
2178     return (P->ops.pop_pread(P, buf, nbyte, address, P->data));
2179 }

2181 ssize_t
2182 Pread_string(struct ps_prochandle *P,
2183     char *buf, /* caller's buffer */
2184     size_t size, /* upper limit on bytes to read */
2185     uintptr_t addr) /* address in process */
2186 {
2187     enum { STRSZ = 40 };
2188     char string[STRSZ + 1];
2189     ssize_t leng = 0;
2190     int nbyte;

2192     if (size < 2) {
2193         errno = EINVAL;
2194         return (-1);
2195     }

2197     size--; /* ensure trailing null fits in buffer */

2199     *buf = '\0';
2200     string[STRSZ] = '\0';

2202     for (nbyte = STRSZ; nbyte == STRSZ && leng < size; addr += STRSZ) {
2203         if ((nbyte = P->ops.pop_pread(P, string, STRSZ, addr,
2204             P->data)) <= 0) {
2205             buf[leng] = '\0';
2206             return (leng ? leng : -1);
2207         }
2208         if ((nbyte = strlen(string)) > 0) {
2209             if (leng + nbyte > size)
2210                 nbyte = size - leng;
2211             (void) strncpy(buf + leng, string, nbyte);
2212             leng += nbyte;
2213         }
2214     }
2215     buf[leng] = '\0';
2216     return (leng);
2217 }

2219 ssize_t
2220 Pwrite(struct ps_prochandle *P,
2221     const void *buf, /* caller's buffer */
2222     size_t nbyte, /* number of bytes to write */
2223     uintptr_t address) /* address in process */
2224 {
2225     return (P->ops.pop_pwrite(P, buf, nbyte, address, P->data));
2226 }

2228 int
2229 Pclearsig(struct ps_prochandle *P)
2230 {
2231     int ctlfd = (P->agentctlfd >= 0)? P->agentctlfd : P->ctlfd;
2232     long ctl = PCCSIG;

2234     if (write(ctlfd, &ctl, sizeof (ctl)) != sizeof (ctl))
2235         return (-1);
2236     P->status.pr_lwp.pr_cursig = 0;

```

```

2237     return (0);
2238 }

2240 int
2241 Pclearfault(struct ps_prochandle *P)
2242 {
2243     int ctlfd = (P->agentctlfd >= 0)? P->agentctlfd : P->ctlfd;
2244     long ctl = PCCFAULT;

2246     if (write(ctlfd, &ctl, sizeof (ctl)) != sizeof (ctl))
2247         return (-1);
2248     return (0);
2249 }

2251 /*
2252  * Set a breakpoint trap, return original instruction.
2253  */
2254 int
2255 Psetbkpt(struct ps_prochandle *P, uintptr_t address, ulong_t *saved)
2256 {
2257     long ctl[1 + sizeof (priovec_t) / sizeof (long) + /* PCREAD */
2258             1 + sizeof (priovec_t) / sizeof (long)]; /* PCWRITE */
2259     long *ctlp = ctl;
2260     size_t size;
2261     priovec_t *iovp;
2262     instr_t bpt = BPT;
2263     instr_t old;

2265     if (P->state == PS_DEAD || P->state == PS_UNDEAD ||
2266         P->state == PS_IDLE) {
2267         errno = ENOENT;
2268         return (-1);
2269     }

2271     /* fetch the old instruction */
2272     *ctlp++ = PCREAD;
2273     iovp = (priovec_t *)ctlp;
2274     iovp->pio_base = &old;
2275     iovp->pio_len = sizeof (old);
2276     iovp->pio_offset = address;
2277     ctlp += sizeof (priovec_t) / sizeof (long);

2279     /* write the BPT instruction */
2280     *ctlp++ = PCWRITE;
2281     iovp = (priovec_t *)ctlp;
2282     iovp->pio_base = &bpt;
2283     iovp->pio_len = sizeof (bpt);
2284     iovp->pio_offset = address;
2285     ctlp += sizeof (priovec_t) / sizeof (long);

2287     size = (char *)ctlp - (char *)ctl;
2288     if (write(P->ctlfd, ctl, size) != size)
2289         return (-1);

2291     /*
2292      * Fail if there was already a breakpoint there from another debugger
2293      * or DTrace's user-level tracing on x86.
2294      */
2295     if (old == BPT) {
2296         errno = EBUSY;
2297         return (-1);
2298     }

2300     *saved = (ulong_t)old;
2301     return (0);
2302 }

```

```

2304 /*
2305  * Restore original instruction where a breakpoint was set.
2306  */
2307 int
2308 Pdelbkpt(struct ps_prochandle *P, uintptr_t address, ulong_t saved)
2309 {
2310     instr_t old = (instr_t)saved;
2311     instr_t cur;

2313     if (P->state == PS_DEAD || P->state == PS_UNDEAD ||
2314         P->state == PS_IDLE) {
2315         errno = ENOENT;
2316         return (-1);
2317     }

2319     /*
2320      * If the breakpoint instruction we had placed has been overwritten
2321      * with a new instruction, then don't try to replace it with the
2322      * old instruction. Doing do can cause problems with self-modifying
2323      * code -- PLTs for example. If the Pread() fails, we assume that we
2324      * should proceed though most likely the Pwrite() will also fail.
2325      */
2326     if (Pread(P, &cur, sizeof (cur), address) == sizeof (cur) &&
2327         cur != BPT)
2328         return (0);

2330     if (Pwrite(P, &old, sizeof (old), address) != sizeof (old))
2331         return (-1);

2333     return (0);
2334 }

2336 /*
2337  * Common code for Pxecbkpt() and Lxecbkpt().
2338  * Develop the array of requests that will do the job, then
2339  * write them to the specified control file descriptor.
2340  * Return the non-zero errno if the write fails.
2341  */
2342 static int
2343 execute_bkpt(
2344     int ctlfd, /* process or LWP control file descriptor */
2345     const fltset_t *faultset, /* current set of traced faults */
2346     const sigset_t *sigmask, /* current signal mask */
2347     uintptr_t address, /* address of breakpoint */
2348     ulong_t saved) /* the saved instruction */
2349 {
2350     long ctl[
2351         1 + sizeof (sigset_t) / sizeof (long) + /* PCSHOLD */
2352         1 + sizeof (fltset_t) / sizeof (long) + /* PCSFAULT */
2353         1 + sizeof (priovec_t) / sizeof (long) + /* PCWRITE */
2354         2 + /* PCRUN */
2355         1 + /* PCWSTOP */
2356         1 + /* PCCFAULT */
2357         1 + sizeof (priovec_t) / sizeof (long) + /* PCWRITE */
2358         1 + sizeof (fltset_t) / sizeof (long) + /* PCSFAULT */
2359         1 + sizeof (sigset_t) / sizeof (long)]; /* PCSHOLD */
2360     long *ctlp = ctl;
2361     sigset_t unblock;
2362     size_t size;
2363     ssize_t ssize;
2364     priovec_t *iovp;
2365     sigset_t *holdp;
2366     fltset_t *faultp;
2367     instr_t old = (instr_t)saved;
2368     instr_t bpt = BPT;

```

```

2369     int error = 0;

2371     /* block our signals for the duration */
2372     (void) sigprocmask(SIG_BLOCK, &blockable_sigs, &unblock);

2374     /* hold posted signals */
2375     *ctlp++ = PCSHOLD;
2376     holdp = (sigset_t *)ctlp;
2377     prfillset(holdp);
2378     prdelset(holdp, SIGKILL);
2379     prdelset(holdp, SIGSTOP);
2380     ctlp += sizeof (sigset_t) / sizeof (long);

2382     /* force tracing of FLTRACE */
2383     if (!(prismember(faultset, FLTRACE))) {
2384         *ctlp++ = PCSFAULT;
2385         faultp = (fltset_t *)ctlp;
2386         *faultp = *faultset;
2387         praddset(faultp, FLTRACE);
2388         ctlp += sizeof (fltset_t) / sizeof (long);
2389     }

2391     /* restore the old instruction */
2392     *ctlp++ = PCWRITE;
2393     iovp = (priovec_t *)ctlp;
2394     iovp->pio_base = &old;
2395     iovp->pio_len = sizeof (old);
2396     iovp->pio_offset = address;
2397     ctlp += sizeof (priovec_t) / sizeof (long);

2399     /* clear current signal and fault; set running w/ single-step */
2400     *ctlp++ = PCRUN;
2401     *ctlp++ = PRCSIG | PRCFault | PRSTEP;

2403     /* wait for stop, cancel the fault */
2404     *ctlp++ = PCWSTOP;
2405     *ctlp++ = PCCFAULT;

2407     /* restore the breakpoint trap */
2408     *ctlp++ = PCWRITE;
2409     iovp = (priovec_t *)ctlp;
2410     iovp->pio_base = &bpt;
2411     iovp->pio_len = sizeof (bpt);
2412     iovp->pio_offset = address;
2413     ctlp += sizeof (priovec_t) / sizeof (long);

2415     /* restore fault tracing set */
2416     if (!(prismember(faultset, FLTRACE))) {
2417         *ctlp++ = PCSFAULT;
2418         *(fltset_t *)ctlp = *faultset;
2419         ctlp += sizeof (fltset_t) / sizeof (long);
2420     }

2422     /* restore the hold mask */
2423     *ctlp++ = PCSHOLD;
2424     *(sigset_t *)ctlp = *sigmask;
2425     ctlp += sizeof (sigset_t) / sizeof (long);

2427     size = (char *)ctlp - (char *)ctl;
2428     if ((ssize = write(ctlfd, ctl, size)) != size)
2429         error = (ssize == -1)? errno : EINTR;
2430     (void) sigprocmask(SIG_SETMASK, &unblock, NULL);
2431     return (error);
2432 }

2434 /*

```

```

2435     * Step over a breakpoint, i.e., execute the instruction that
2436     * really belongs at the breakpoint location (the current %pc)
2437     * and leave the process stopped at the next instruction.
2438     */
2439     int
2440     Pxecbkpt(struct ps_prochandle *P, ulong_t saved)
2441     {
2442         int ctlfd = (P->agentctlfd >= 0)? P->agentctlfd : P->ctlfd;
2443         int rv, error;

2445         if (P->state != PS_STOP) {
2446             errno = EBUSY;
2447             return (-1);
2448         }

2450         Psync(P);

2452         error = execute_bkpt(ctlfd,
2453             &P->status.pr_fltrace, &P->status.pr_lwp.pr_lwphold,
2454             P->status.pr_lwp.pr_reg[R_PC], saved);
2455         rv = Pstopstatus(P, PCNULL, 0);

2457         if (error != 0) {
2458             if (P->status.pr_lwp.pr_why == PR_JOBCONTROL &&
2459                 error == EBUSY) { /* jobcontrol stop -- back off */
2460                 P->state = PS_RUN;
2461                 return (0);
2462             }
2463             if (error == ENOENT)
2464                 return (0);
2465             errno = error;
2466             return (-1);
2467         }

2469         return (rv);
2470     }

2472     /*
2473     * Install the watchpoint described by wp.
2474     */
2475     int
2476     Psetwapt(struct ps_prochandle *P, const prwatch_t *wp)
2477     {
2478         long ctl[1 + sizeof (prwatch_t) / sizeof (long)];
2479         prwatch_t *cwp = (prwatch_t *)&ctl[1];

2481         if (P->state == PS_DEAD || P->state == PS_UNDEAD ||
2482             P->state == PS_IDLE) {
2483             errno = ENOENT;
2484             return (-1);
2485         }

2487         ctl[0] = PCWATCH;
2488         cwp->pr_vaddr = wp->pr_vaddr;
2489         cwp->pr_size = wp->pr_size;
2490         cwp->pr_wflags = wp->pr_wflags;

2492         if (write(P->ctlfd, ctl, sizeof (ctl)) != sizeof (ctl))
2493             return (-1);

2495         return (0);
2496     }

2498     /*
2499     * Remove the watchpoint described by wp.
2500     */

```



```

2501 int
2502 Pdelwapt(struct ps_prochandle *P, const prwatch_t *wp)
2503 {
2504     long ctl[1 + sizeof (prwatch_t) / sizeof (long)];
2505     prwatch_t *cwp = (prwatch_t *)&ctl[1];

2507     if (P->state == PS_DEAD || P->state == PS_UNDEAD ||
2508         P->state == PS_IDLE) {
2509         errno = ENOENT;
2510         return (-1);
2511     }

2513     ctl[0] = PCWATCH;
2514     cwp->pr_vaddr = wp->pr_vaddr;
2515     cwp->pr_size = wp->pr_size;
2516     cwp->pr_wflags = 0;

2518     if (write(P->ctldfd, ctl, sizeof (ctl)) != sizeof (ctl))
2519         return (-1);

2521     return (0);
2522 }

2524 /*
2525  * Common code for Pxecwapt() and Lxecwapt().  Develop the array of requests
2526  * that will do the job, then write them to the specified control file
2527  * descriptor.  Return the non-zero errno if the write fails.
2528  */
2529 static int
2530 execute_wapt(
2531     int ctldfd,          /* process or LWP control file descriptor */
2532     const fltset_t *faultset, /* current set of traced faults */
2533     const sigset_t *sigmask, /* current signal mask */
2534     const prwatch_t *wp) /* watchpoint descriptor */
2535 {
2536     long ctl[
2537         1 + sizeof (sigset_t) / sizeof (long) + /* PCSHOLD */
2538         1 + sizeof (fltset_t) / sizeof (long) + /* PCSFAULT */
2539         1 + sizeof (prwatch_t) / sizeof (long) + /* PCWATCH */
2540         2 + /* PCRUN */
2541         1 + /* PCWSTOP */
2542         1 + /* PCCFAULT */
2543         1 + sizeof (prwatch_t) / sizeof (long) + /* PCWATCH */
2544         1 + sizeof (fltset_t) / sizeof (long) + /* PCSFAULT */
2545         1 + sizeof (sigset_t) / sizeof (long)];

2547     long *ctlp = ctl;
2548     int error = 0;

2550     sigset_t unblock;
2551     sigset_t *holdp;
2552     fltset_t *faultp;
2553     prwatch_t *prw;
2554     ssize_t ssize;
2555     size_t size;

2557     (void) sigprocmask(SIG_BLOCK, &blockable_sigs, &unblock);

2559     /*
2560      * Hold all posted signals in the victim process prior to stepping.
2561      */
2562     *ctlp++ = PCSHOLD;
2563     holdp = (sigset_t *)ctlp;
2564     prfillset(holdp);
2565     prdelset(holdp, SIGKILL);
2566     prdelset(holdp, SIGSTOP);

```

```

2567     ctlp += sizeof (sigset_t) / sizeof (long);

2569     /*
2570      * Force tracing of FLTRACE since we need to single step.
2571      */
2572     if (!(prismember(faultset, FLTRACE))) {
2573         *ctlp++ = PCSFAULT;
2574         faultp = (fltset_t *)ctlp;
2575         *faultp = *faultset;
2576         praddset(faultp, FLTRACE);
2577         ctlp += sizeof (fltset_t) / sizeof (long);
2578     }

2580     /*
2581      * Clear only the current watchpoint by setting pr_wflags to zero.
2582      */
2583     *ctlp++ = PCWATCH;
2584     prw = (prwatch_t *)ctlp;
2585     prw->pr_vaddr = wp->pr_vaddr;
2586     prw->pr_size = wp->pr_size;
2587     prw->pr_wflags = 0;
2588     ctlp += sizeof (prwatch_t) / sizeof (long);

2590     /*
2591      * Clear the current signal and fault; set running with single-step.
2592      * Then wait for the victim to stop and cancel the FLTRACE.
2593      */
2594     *ctlp++ = PCRUN;
2595     *ctlp++ = PRCSIG | PRCFAULT | PRSTEP;
2596     *ctlp++ = PCWSTOP;
2597     *ctlp++ = PCCFAULT;

2599     /*
2600      * Restore the current watchpoint.
2601      */
2602     *ctlp++ = PCWATCH;
2603     (void) memcpy(ctlp, wp, sizeof (prwatch_t));
2604     ctlp += sizeof (prwatch_t) / sizeof (long);

2606     /*
2607      * Restore fault tracing set if we modified it.
2608      */
2609     if (!(prismember(faultset, FLTRACE))) {
2610         *ctlp++ = PCSFAULT;
2611         *(fltset_t *)ctlp = *faultset;
2612         ctlp += sizeof (fltset_t) / sizeof (long);
2613     }

2615     /*
2616      * Restore the hold mask to the current hold mask (i.e. the one
2617      * before we executed any of the previous operations).
2618      */
2619     *ctlp++ = PCSHOLD;
2620     *(sigset_t *)ctlp = *sigmask;
2621     ctlp += sizeof (sigset_t) / sizeof (long);

2623     size = (char *)ctlp - (char *)ctl;
2624     if ((ssize = write(ctldfd, ctl, size)) != size)
2625         error = (ssize == -1)? errno : EINTR;
2626     (void) sigprocmask(SIG_SETMASK, &unblock, NULL);
2627     return (error);
2628 }

2630 /*
2631  * Step over a watchpoint, i.e., execute the instruction that was stopped by
2632  * the watchpoint, and then leave the LWP stopped at the next instruction.

```

```

2633 */
2634 int
2635 Pxecwapt(struct ps_prochandle *P, const prwatch_t *wp)
2636 {
2637     int ctlfd = (P->agentctlfd >= 0)? P->agentctlfd : P->ctlfd;
2638     int rv, error;

2640     if (P->state != PS_STOP) {
2641         errno = EBUSY;
2642         return (-1);
2643     }

2645     Psync(P);
2646     error = execute_wapt(ctlfd,
2647         &P->status.pr_filttrace, &P->status.pr_lwp.pr_lwphold, wp);
2648     rv = Pstopstatus(P, PCNULL, 0);

2650     if (error != 0) {
2651         if (P->status.pr_lwp.pr_why == PR_JOBCONTROL &&
2652             error == EBUSY) { /* jobcontrol stop -- back off */
2653             P->state = PS_RUN;
2654             return (0);
2655         }
2656         if (error == ENOENT)
2657             return (0);
2658         errno = error;
2659         return (-1);
2660     }

2662     return (rv);
2663 }

2665 int
2666 Psetflags(struct ps_prochandle *P, long flags)
2667 {
2668     int rc;
2669     long ctl[2];

2671     ctl[0] = PCSET;
2672     ctl[1] = flags;

2674     if (write(P->ctlfd, ctl, 2*sizeof (long)) != 2*sizeof (long)) {
2675         rc = -1;
2676     } else {
2677         P->status.pr_flags |= flags;
2678         P->status.pr_lwp.pr_flags |= flags;
2679         rc = 0;
2680     }

2682     return (rc);
2683 }

2685 int
2686 Punsetflags(struct ps_prochandle *P, long flags)
2687 {
2688     int rc;
2689     long ctl[2];

2691     ctl[0] = PCUNSET;
2692     ctl[1] = flags;

2694     if (write(P->ctlfd, ctl, 2*sizeof (long)) != 2*sizeof (long)) {
2695         rc = -1;
2696     } else {
2697         P->status.pr_flags &= ~flags;
2698         P->status.pr_lwp.pr_flags &= ~flags;

```

```

2699         rc = 0;
2700     }

2702     return (rc);
2703 }

2705 /*
2706  * Common function to allow clients to manipulate the action to be taken
2707  * on receipt of a signal, receipt of machine fault, entry to a system call,
2708  * or exit from a system call. We make use of our private prset_* functions
2709  * in order to make this code be common. The 'which' parameter identifies
2710  * the code for the event of interest (0 means change the entire set), and
2711  * the 'stop' parameter is a boolean indicating whether the process should
2712  * stop when the event of interest occurs. The previous value is returned
2713  * to the caller; -1 is returned if an error occurred.
2714  */
2715 static int
2716 Psetaction(struct ps_prochandle *P, void *sp, size_t size,
2717     uint_t flag, int max, int which, int stop)
2718 {
2719     int oldval;

2721     if (which < 0 || which > max) {
2722         errno = EINVAL;
2723         return (-1);
2724     }

2726     if (P->state == PS_DEAD || P->state == PS_UNDEAD ||
2727         P->state == PS_IDLE) {
2728         errno = ENOENT;
2729         return (-1);
2730     }

2732     oldval = prset_ismember(sp, size, which) ? TRUE : FALSE;

2734     if (stop) {
2735         if (which == 0) {
2736             prset_fill(sp, size);
2737             P->flags |= flag;
2738         } else if (!oldval) {
2739             prset_add(sp, size, which);
2740             P->flags |= flag;
2741         }
2742     } else {
2743         if (which == 0) {
2744             prset_empty(sp, size);
2745             P->flags |= flag;
2746         } else if (oldval) {
2747             prset_del(sp, size, which);
2748             P->flags |= flag;
2749         }
2750     }

2752     if (P->state == PS_RUN)
2753         Psync(P);

2755     return (oldval);
2756 }

2758 /*
2759  * Set action on specified signal.
2760  */
2761 int
2762 Psignal(struct ps_prochandle *P, int which, int stop)
2763 {
2764     int oldval;

```

```

2766     if (which == SIGKILL && stop != 0) {
2767         errno = EINVAL;
2768         return (-1);
2769     }

2771     oldval = Psetaction(P, &P->status.pr_sigtrace, sizeof (sigset_t),
2772         SETSIG, PRMAXSIG, which, stop);

2774     if (oldval != -1 && which == 0 && stop != 0)
2775         prdelset(&P->status.pr_sigtrace, SIGKILL);

2777     return (oldval);
2778 }

2780 /*
2781  * Set all signal tracing flags.
2782  */
2783 void
2784 Psetsignal(struct ps_prochandle *P, const sigset_t *set)
2785 {
2786     if (P->state == PS_DEAD || P->state == PS_UNDEAD ||
2787         P->state == PS_IDLE)
2788         return;

2790     P->status.pr_sigtrace = *set;
2791     P->flags |= SETSIG;

2793     if (P->state == PS_RUN)
2794         Psync(P);
2795 }

2797 /*
2798  * Set action on specified fault.
2799  */
2800 int
2801 Pfault(struct ps_prochandle *P, int which, int stop)
2802 {
2803     return (Psetaction(P, &P->status.pr_fltrace, sizeof (fltset_t),
2804         SETFAULT, PRMAXFAULT, which, stop));
2805 }

2807 /*
2808  * Set all machine fault tracing flags.
2809  */
2810 void
2811 Psetfault(struct ps_prochandle *P, const fltset_t *set)
2812 {
2813     if (P->state == PS_DEAD || P->state == PS_UNDEAD ||
2814         P->state == PS_IDLE)
2815         return;

2817     P->status.pr_fltrace = *set;
2818     P->flags |= SETFAULT;

2820     if (P->state == PS_RUN)
2821         Psync(P);
2822 }

2824 /*
2825  * Set action on specified system call entry.
2826  */
2827 int
2828 Psysentry(struct ps_prochandle *P, int which, int stop)
2829 {
2830     return (Psetaction(P, &P->status.pr_sysentry, sizeof (sysset_t),

```

```

2831         SETENTRY, PRMAXSYS, which, stop));
2832 }

2834 /*
2835  * Set all system call entry tracing flags.
2836  */
2837 void
2838 Psetsysentry(struct ps_prochandle *P, const sysset_t *set)
2839 {
2840     if (P->state == PS_DEAD || P->state == PS_UNDEAD ||
2841         P->state == PS_IDLE)
2842         return;

2844     P->status.pr_sysentry = *set;
2845     P->flags |= SETENTRY;

2847     if (P->state == PS_RUN)
2848         Psync(P);
2849 }

2851 /*
2852  * Set action on specified system call exit.
2853  */
2854 int
2855 Psysexit(struct ps_prochandle *P, int which, int stop)
2856 {
2857     return (Psetaction(P, &P->status.pr_sysexit, sizeof (sysset_t),
2858         SETEXIT, PRMAXSYS, which, stop));
2859 }

2861 /*
2862  * Set all system call exit tracing flags.
2863  */
2864 void
2865 Psetsysexit(struct ps_prochandle *P, const sysset_t *set)
2866 {
2867     if (P->state == PS_DEAD || P->state == PS_UNDEAD ||
2868         P->state == PS_IDLE)
2869         return;

2871     P->status.pr_sysexit = *set;
2872     P->flags |= SETEXIT;

2874     if (P->state == PS_RUN)
2875         Psync(P);
2876 }

2878 /*
2879  * Utility function to read the contents of a file that contains a
2880  * prheader_t at the start (/proc/pid/lstatus or /proc/pid/lpsinfo).
2881  * Returns a malloc(d) buffer or NULL on failure.
2882  */
2883 static prheader_t *
2884 read_lfile(struct ps_prochandle *P, const char *lname)
2885 {
2886     prheader_t *Lhp;
2887     char lpath[PATH_MAX];
2888     struct stat64 statb;
2889     int fd;
2890     size_t size;
2891     ssize_t rval;

2893     (void) snprintf(lpath, sizeof (lpath), "%s/%d/%s", procfs_path,
2894         (int)P->status.pr_pid, lname);
2895     if ((fd = open(lpath, O_RDONLY)) < 0 || fstat64(fd, &statb) != 0) {
2896         if (fd >= 0)

```

```

2897         (void) close(fd);
2898         return (NULL);
2899     }

2901     /*
2902     * 'size' is just the initial guess at the buffer size.
2903     * It will have to grow if the number of lwps increases
2904     * while we are looking at the process.
2905     * 'size' must be larger than the actual file size.
2906     */
2907     size = statb.st_size + 32;

2909     for (;;) {
2910         if ((Lhp = malloc(size)) == NULL)
2911             break;
2912         if ((rval = pread(fd, Lhp, size, 0)) < 0 ||
2913             rval <= sizeof (prheader_t)) {
2914             free(Lhp);
2915             Lhp = NULL;
2916             break;
2917         }
2918         if (rval < size)
2919             break;
2920         /* need a bigger buffer */
2921         free(Lhp);
2922         size *= 2;
2923     }

2925     (void) close(fd);
2926     return (Lhp);
2927 }

2929 /*
2930 * LWP iteration interface.
2931 */
2932 int
2933 Plwp_iter(struct ps_prochandle *P, proc_lwp_f *func, void *cd)
2934 {
2935     prheader_t *Lhp;
2936     lwpstatus_t *Lsp;
2937     long nlwp;
2938     int rv;

2940     switch (P->state) {
2941     case PS_RUN:
2942         (void) Pstopstatus(P, PCNULL, 0);
2943         break;

2945     case PS_STOP:
2946         Psync(P);
2947         break;

2949     case PS_IDLE:
2950         errno = ENODATA;
2951         return (-1);
2952     }

2954     /*
2955     * For either live processes or cores, the single LWP case is easy:
2956     * the pstatus_t contains the lwpstatus_t for the only LWP.
2957     */
2958     if (P->status.pr_nlwp <= 1)
2959         return (func(cd, &P->status.pr_lwp));

2961     /*
2962     * For the core file multi-LWP case, we just iterate through the

```

```

2963     * list of LWP structs we read in from the core file.
2964     */
2965     if (P->state == PS_DEAD) {
2966         core_info_t *core = P->data;
2967         lwp_info_t *lwp = list_prev(&core->core_lwp_head);
2968         uint_t i;

2970         for (i = 0; i < core->core_nlwp; i++, lwp = list_prev(lwp)) {
2971             if (lwp->lwp_psinfo.pr_sname != 'Z' &&
2972                 (rv = func(cd, &lwp->lwp_status)) != 0)
2973                 break;
2974         }

2976         return (rv);
2977     }

2979     /*
2980     * For the live process multi-LWP case, we have to work a little
2981     * harder: the /proc/pid/lstatus file has the array of LWP structs.
2982     */
2983     if ((Lhp = Plstatus(P)) == NULL)
2984         return (-1);

2986     for (nlwp = Lhp->pr_nent, Lsp = (lwpstatus_t *) (uintptr_t) (Lhp + 1);
2987          nlwp > 0;
2988          nlwp--, Lsp = (lwpstatus_t *) ((uintptr_t) Lsp + Lhp->pr_entsize)) {
2989         if ((rv = func(cd, Lsp)) != 0)
2990             break;
2991     }

2993     free(Lhp);
2994     return (rv);
2995 }

2997 /*
2998 * Extended LWP iteration interface.
2999 * Iterate over all LWPs, active and zombie.
3000 */
3001 int
3002 Plwp_iter_all(struct ps_prochandle *P, proc_lwp_all_f *func, void *cd)
3003 {
3004     prheader_t *Lhp = NULL;
3005     lwpstatus_t *Lsp;
3006     lwpstatus_t *lsp;
3007     prheader_t *Lphp = NULL;
3008     lwpstatus_t *Llsp;
3009     long nstat;
3010     long ninfo;
3011     int rv;

3013     retry:
3014     if (Lhp != NULL)
3015         free(Lhp);
3016     if (Lphp != NULL)
3017         free(Lphp);
3018     if (P->state == PS_RUN)
3019         (void) Pstopstatus(P, PCNULL, 0);
3020     (void) Ppsinfo(P);

3022     if (P->state == PS_STOP)
3023         Psync(P);

3025     /*
3026     * For either live processes or cores, the single LWP case is easy:
3027     * the pstatus_t contains the lwpstatus_t for the only LWP and
3028     * the psinfo_t contains the lwpstatus_t for the only LWP.

```

```

3029  */
3030  if (P->status.pr_nlwp + P->status.pr_nzomb <= 1)
3031      return (func(cd, &P->status.pr_lwp, &P->psinfo.pr_lwp));

3033  /*
3034  * For the core file multi-LWP case, we just iterate through the
3035  * list of LWP structs we read in from the core file.
3036  */
3037  if (P->state == PS_DEAD) {
3038      core_info_t *core = P->data;
3039      lwp_info_t *lwp = list_prev(&core->core_lwp_head);
3040      uint_t i;

3042      for (i = 0; i < core->core_nlwp; i++, lwp = list_prev(lwp)) {
3043          sp = (lwp->lwp_psinfo.pr_sname == 'Z')? NULL :
3044              &lwp->lwp_status;
3045          if ((rv = func(cd, sp, &lwp->lwp_psinfo)) != 0)
3046              break;
3047      }

3049      return (rv);
3050  }

3052  /*
3053  * For all other cases retrieve the array of lwpstatus_t's and
3054  * lwpsinfo_t's.
3055  */
3056  if ((Lhp = Plstatus(P)) == NULL)
3057      return (-1);
3058  if ((Lphp = Plpsinfo(P)) == NULL) {
3059      free(Lhp);
3060      return (-1);
3061  }

3063  /*
3064  * If we are looking at a running process, or one we do not control,
3065  * the active and zombie lwps in the process may have changed since
3066  * we read the process status structure.  If so, just start over.
3067  */
3068  if (Lhp->pr_nent != P->status.pr_nlwp ||
3069      Lphp->pr_nent != P->status.pr_nlwp + P->status.pr_nzomb)
3070      goto retry;

3072  /*
3073  * To be perfectly safe, prescan the two arrays, checking consistency.
3074  * We rely on /proc giving us lwpstatus_t's and lwpsinfo_t's in the
3075  * same order (the lwp directory order) in their respective files.
3076  * We also rely on there being (possibly) more lwpsinfo_t's than
3077  * lwpstatus_t's (the extra lwpsinfo_t's are for zombie lwps).
3078  */
3079  Lsp = (lwpstatus_t *) (uintptr_t) (Lhp + 1);
3080  Lpsp = (lwpsinfo_t *) (uintptr_t) (Lphp + 1);
3081  nstat = Lhp->pr_nent;
3082  for (ninfo = Lphp->pr_nent; ninfo != 0; ninfo--) {
3083      if (Lpsp->pr_sname != 'Z') {
3084          /*
3085           * Not a zombie lwp; check for matching lwpids.
3086           */
3087          if (nstat == 0 || Lsp->pr_lwpid != Lpsp->pr_lwpid)
3088              goto retry;
3089          Lsp = (lwpstatus_t *) ((uintptr_t) Lsp + Lhp->pr_entsize);
3090          nstat--;
3091      }
3092      Lpsp = (lwpsinfo_t *) ((uintptr_t) Lpsp + Lphp->pr_entsize);
3093  }
3094  if (nstat != 0)

```

```

3095      goto retry;

3097  /*
3098  * Rescan, this time for real.
3099  */
3100  Lsp = (lwpstatus_t *) (uintptr_t) (Lhp + 1);
3101  Lpsp = (lwpsinfo_t *) (uintptr_t) (Lphp + 1);
3102  for (ninfo = Lphp->pr_nent; ninfo != 0; ninfo--) {
3103      if (Lpsp->pr_sname != 'Z') {
3104          sp = Lsp;
3105          Lsp = (lwpstatus_t *) ((uintptr_t) Lsp + Lhp->pr_entsize);
3106      } else {
3107          sp = NULL;
3108      }
3109      if ((rv = func(cd, sp, Lpsp)) != 0)
3110          break;
3111      Lpsp = (lwpsinfo_t *) ((uintptr_t) Lpsp + Lphp->pr_entsize);
3112  }

3114      free(Lhp);
3115      free(Lphp);
3116      return (rv);
3117  }

3119  core_content_t
3120  Pcontent(struct ps_prochandle *P)
3121  {
3122      core_info_t *core = P->data;

3124      if (P->state == PS_DEAD)
3125          return (core->core_content);
3126      if (P->state == PS_IDLE)
3127          return (CC_CONTENT_TEXT | CC_CONTENT_DATA | CC_CONTENT_CTF);

3129      return (CC_CONTENT_ALL);
3130  }

3132  /*
3133  * =====
3134  * The remainder of the functions in this file are for the
3135  * control of individual LWPs in the controlled process.
3136  * =====
3137  */

3139  /*
3140  * Find an entry in the process hash table for the specified lwpid.
3141  * The entry will either point to an existing struct ps_lwphandle
3142  * or it will point to an empty slot for a new struct ps_lwphandle.
3143  */
3144  static struct ps_lwphandle **
3145  Lfind(struct ps_prochandle *P, lwpid_t lwpid)
3146  {
3147      struct ps_lwphandle **Lp;
3148      struct ps_lwphandle *L;

3150      for (Lp = &P->hashtab[lwpid % (HASHSIZE - 1)];
3151           (L = *Lp) != NULL; Lp = &L->lwp_hash)
3152          if (L->lwp_id == lwpid)
3153              break;
3154      return (Lp);
3155  }

3157  /*
3158  * Grab an LWP contained within the controlled process.
3159  * Return an opaque pointer to its LWP control structure.
3160  * perr: pointer to error return code.

```

```

3161 */
3162 struct ps_lwphandle *
3163 Lgrab(struct ps_prochandle *P, lwpid_t lwpid, int *perr)
3164 {
3165     struct ps_lwphandle **Lp;
3166     struct ps_lwphandle *L;
3167     int fd;
3168     char procname[PATH_MAX];
3169     char *fname;
3170     int rc = 0;
3171
3172     (void) mutex_lock(&P->proc_lock);
3173
3174     if (P->state == PS_UNDEAD || P->state == PS_IDLE)
3175         rc = G_NOPROC;
3176     else if (P->hashtab == NULL &&
3177             (P->hashtab = calloc(HASHSIZE, sizeof (struct ps_lwphandle *)))
3178              == NULL)
3179         rc = G_STRANGE;
3180     else if (*(Lp = Lfind(P, lwpid)) != NULL)
3181         rc = G_BUSY;
3182     else if ((L = malloc(sizeof (struct ps_lwphandle))) == NULL)
3183         rc = G_STRANGE;
3184     if (rc) {
3185         *perr = rc;
3186         (void) mutex_unlock(&P->proc_lock);
3187         return (NULL);
3188     }
3189
3190     (void) memset(L, 0, sizeof (*L));
3191     L->lwp_ctlfd = -1;
3192     L->lwp_statfd = -1;
3193     L->lwp_proc = P;
3194     L->lwp_id = lwpid;
3195     *Lp = L; /* insert into the hash table */
3196
3197     if (P->state == PS_DEAD) { /* core file */
3198         if (getlwpstatus(P, lwpid, &L->lwp_status) == -1) {
3199             rc = G_NOPROC;
3200             goto err;
3201         }
3202         L->lwp_state = PS_DEAD;
3203         *perr = 0;
3204         (void) mutex_unlock(&P->proc_lock);
3205         return (L);
3206     }
3207
3208     /*
3209     * Open the /proc/<pid>/lwp/<lwpid> files
3210     */
3211     (void) snprintf(procname, sizeof (procname), "%s/%d/lwp/%d/",
3212                   procfs_path, (int)P->pid, (int)lwpid);
3213     fname = procname + strlen(procname);
3214     (void) set_minfd();
3215
3216     (void) strcpy(fname, "lwpstatus");
3217     if ((fd = open(procname, O_RDONLY)) < 0 ||
3218         (fd = dupfd(fd, 0)) < 0) {
3219         switch (errno) {
3220             case ENOENT:
3221                 rc = G_NOPROC;
3222                 break;
3223             default:
3224                 dprintf("Lgrab: failed to open %s: %s\n",
3225                       procname, strerror(errno));
3226                 rc = G_STRANGE;

```

```

3227         break;
3228     }
3229     goto err;
3230 }
3231 L->lwp_statfd = fd;
3232
3233 if (pread(fd, &L->lwp_status, sizeof (L->lwp_status), (off_t)0) < 0) {
3234     switch (errno) {
3235         case ENOENT:
3236             rc = G_NOPROC;
3237             break;
3238         default:
3239             dprintf("Lgrab: failed to read %s: %s\n",
3240                   procname, strerror(errno));
3241             rc = G_STRANGE;
3242             break;
3243     }
3244     goto err;
3245 }
3246
3247 (void) strcpy(fname, "lwpctl");
3248 if ((fd = open(procname, O_WRONLY)) < 0 ||
3249     (fd = dupfd(fd, 0)) < 0) {
3250     switch (errno) {
3251         case ENOENT:
3252             rc = G_NOPROC;
3253             break;
3254         default:
3255             dprintf("Lgrab: failed to open %s: %s\n",
3256                   procname, strerror(errno));
3257             rc = G_STRANGE;
3258             break;
3259     }
3260     goto err;
3261 }
3262 L->lwp_ctlfd = fd;
3263
3264 L->lwp_state =
3265     ((L->lwp_status.pr_flags & (PR_STOPPED|PR_ISTOP))
3266      == (PR_STOPPED|PR_ISTOP)) ?
3267     PS_STOP : PS_RUN;
3268
3269 *perr = 0;
3270 (void) mutex_unlock(&P->proc_lock);
3271 return (L);
3272
3273 err:
3274     Lfree_internal(P, L);
3275     *perr = rc;
3276     (void) mutex_unlock(&P->proc_lock);
3277     return (NULL);
3278 }
3279
3280 /*
3281  * Return a printable string corresponding to an Lgrab() error return.
3282  */
3283 const char *
3284 Lgrab_error(int error)
3285 {
3286     const char *str;
3287
3288     switch (error) {
3289     case G_NOPROC:
3290         str = "no such LWP";
3291         break;
3292     case G_BUSY:

```

```

3293         str = "LWP already grabbed";
3294         break;
3295     case G_STRANGE:
3296         str = "unanticipated system error";
3297         break;
3298     default:
3299         str = "unknown error";
3300         break;
3301     }
3302
3303     return (str);
3304 }
3305
3306 /*
3307  * Free an LWP control structure.
3308  */
3309 void
3310 Lfree(struct ps_lwphandle *L)
3311 {
3312     struct ps_prochandle *P = L->lwp_proc;
3313
3314     (void) mutex_lock(&P->proc_lock);
3315     Lfree_internal(P, L);
3316     (void) mutex_unlock(&P->proc_lock);
3317 }
3318
3319 static void
3320 Lfree_internal(struct ps_prochandle *P, struct ps_lwphandle *L)
3321 {
3322     *Lfind(P, L->lwp_id) = L->lwp_hash;    /* delete from hash table */
3323     if (L->lwp_ctldfd >= 0)
3324         (void) close(L->lwp_ctldfd);
3325     if (L->lwp_statfd >= 0)
3326         (void) close(L->lwp_statfd);
3327
3328     /* clear out the structure as a precaution against reuse */
3329     (void) memset(L, 0, sizeof (*L));
3330     L->lwp_ctldfd = -1;
3331     L->lwp_statfd = -1;
3332
3333     free(L);
3334 }
3335
3336 /*
3337  * Return the state of the process, one of the PS_* values.
3338  */
3339 int
3340 Lstate(struct ps_lwphandle *L)
3341 {
3342     return (L->lwp_state);
3343 }
3344
3345 /*
3346  * Return the open control file descriptor for the LWP.
3347  * Clients must not close this file descriptor, nor use it
3348  * after the LWP is freed.
3349  */
3350 int
3351 Lctldfd(struct ps_lwphandle *L)
3352 {
3353     return (L->lwp_ctldfd);
3354 }
3355
3356 /*
3357  * Return a pointer to the LWP lwpsinfo structure.
3358  * Clients should not hold on to this pointer indefinitely.

```

```

3359  * It will become invalid on Lfree().
3360  */
3361 const lwpsinfo_t *
3362 Lpsinfo(struct ps_lwphandle *L)
3363 {
3364     if (Plwp_getpsinfo(L->lwp_proc, L->lwp_id, &L->lwp_psinfo) == -1)
3365         return (NULL);
3366
3367     return (&L->lwp_psinfo);
3368 }
3369
3370 /*
3371  * Return a pointer to the LWP status structure.
3372  * Clients should not hold on to this pointer indefinitely.
3373  * It will become invalid on Lfree().
3374  */
3375 const lwpstatus_t *
3376 Lstatus(struct ps_lwphandle *L)
3377 {
3378     return (&L->lwp_status);
3379 }
3380
3381 /*
3382  * Given an LWP handle, return the process handle.
3383  */
3384 struct ps_prochandle *
3385 Lprochandle(struct ps_lwphandle *L)
3386 {
3387     return (L->lwp_proc);
3388 }
3389
3390 /*
3391  * Ensure that all cached state is written to the LWP.
3392  * The cached state is the LWP's signal mask and registers.
3393  */
3394 void
3395 Lsync(struct ps_lwphandle *L)
3396 {
3397     int ctldfd = L->lwp_ctldfd;
3398     long cmd[2];
3399     iovec_t iov[4];
3400     int n = 0;
3401
3402     if (L->lwp_flags & SETHOLD) {
3403         cmd[0] = PCSHOLD;
3404         iov[n].iov_base = (caddr_t)&cmd[0];
3405         iov[n+1].iov_len = sizeof (long);
3406         iov[n].iov_base = (caddr_t)&L->lwp_status.pr_lwphold;
3407         iov[n+1].iov_len = sizeof (L->lwp_status.pr_lwphold);
3408     }
3409     if (L->lwp_flags & SETREGS) {
3410         cmd[1] = PCSREG;
3411         iov[n].iov_base = (caddr_t)&cmd[1];
3412         iov[n+1].iov_len = sizeof (long);
3413         iov[n].iov_base = (caddr_t)&L->lwp_status.pr_reg[0];
3414         iov[n+1].iov_len = sizeof (L->lwp_status.pr_reg);
3415     }
3416
3417     if (n == 0 || writev(ctldfd, iov, n) < 0)
3418         return;    /* nothing to do or write failed */
3419
3420     L->lwp_flags &= ~(SETHOLD|SETREGS);
3421 }
3422
3423 /*
3424  * Wait for the specified LWP to stop or terminate.

```

```

3425 * Or, just get the current status (PCNULL).
3426 * Or, direct it to stop and get the current status (PCDSTOP).
3427 */
3428 static int
3429 Lstopstatus(struct ps_lwphandle *L,
3430             long request, /* PCNULL, PCDSTOP, PCSTOP, PCWSTOP */
3431             uint_t msec) /* if non-zero, timeout in milliseconds */
3432 {
3433     int ctlfd = L->lwp_ctlfd;
3434     long ctl[3];
3435     ssize_t rc;
3436     int err;

3438     switch (L->lwp_state) {
3439     case PS_RUN:
3440         break;
3441     case PS_STOP:
3442         if (request != PCNULL && request != PCDSTOP)
3443             return (0);
3444         break;
3445     case PS_LOST:
3446         if (request != PCNULL) {
3447             errno = EAGAIN;
3448             return (-1);
3449         }
3450         break;
3451     case PS_UNDEAD:
3452     case PS_DEAD:
3453         if (request != PCNULL) {
3454             errno = ENOENT;
3455             return (-1);
3456         }
3457         break;
3458     default: /* corrupted state */
3459         dprintf("Lstopstatus: corrupted state: %d\n", L->lwp_state);
3460         errno = EINVAL;
3461         return (-1);
3462     }

3464     ctl[0] = PCDSTOP;
3465     ctl[1] = PCTWSTOP;
3466     ctl[2] = (long)msec;
3467     rc = 0;
3468     switch (request) {
3469     case PCSTOP:
3470         rc = write(ctlfd, &ctl[0], 3*sizeof (long));
3471         break;
3472     case PCWSTOP:
3473         rc = write(ctlfd, &ctl[1], 2*sizeof (long));
3474         break;
3475     case PCDSTOP:
3476         rc = write(ctlfd, &ctl[0], 1*sizeof (long));
3477         break;
3478     case PCNULL:
3479         if (L->lwp_state == PS_DEAD)
3480             return (0); /* Nothing else to do for cores */
3481         break;
3482     default: /* programming error */
3483         errno = EINVAL;
3484         return (-1);
3485     }
3486     err = (rc < 0)? errno : 0;
3487     Lsync(L);

3489     if (pread(L->lwp_statfd, &L->lwp_status,
3490             sizeof (L->lwp_status), (off_t)0) < 0)

```

```

3491         err = errno;

3493     if (err) {
3494         switch (err) {
3495         case EINTR: /* user typed ctl-C */
3496         case ERESTART:
3497             dprintf("Lstopstatus: EINTR\n");
3498             break;
3499         case EAGAIN: /* we lost control of the the process */
3500             dprintf("Lstopstatus: EAGAIN\n");
3501             L->lwp_state = PS_LOST;
3502             errno = err;
3503             return (-1);
3504         default:
3505             if (_libproc_debug) {
3506                 const char *errstr;

3508                 switch (request) {
3509                 case PCNULL:
3510                     errstr = "Lstopstatus PCNULL"; break;
3511                 case PCSTOP:
3512                     errstr = "Lstopstatus PCSTOP"; break;
3513                 case PCDSTOP:
3514                     errstr = "Lstopstatus PCDSTOP"; break;
3515                 case PCWSTOP:
3516                     errstr = "Lstopstatus PCWSTOP"; break;
3517                 default:
3518                     errstr = "Lstopstatus PC???"; break;
3519             }
3520             dprintf("%s: %s\n", errstr, strerror(err));
3521         }
3522         L->lwp_state = PS_UNDEAD;
3523         errno = err;
3524         return (-1);
3525     }
3526 }

3528     if ((L->lwp_status.pr_flags & (PR_STOPPED|PR_ISTOP))
3529         != (PR_STOPPED|PR_ISTOP)) {
3530         L->lwp_state = PS_RUN;
3531         if (request == PCNULL || request == PCDSTOP || msec != 0)
3532             return (0);
3533         dprintf("Lstopstatus: LWP is not stopped\n");
3534         errno = EPROTO;
3535         return (-1);
3536     }

3538     L->lwp_state = PS_STOP;

3540     if (_libproc_debug) /* debugging */
3541         prldump("Lstopstatus", &L->lwp_status);

3543     switch (L->lwp_status.pr_why) {
3544     case PR_SYSENTRY:
3545     case PR_SYSEXIT:
3546     case PR_REQUESTED:
3547     case PR_SIGNALED:
3548     case PR_FAULTED:
3549     case PR_JOBCONTROL:
3550     case PR_SUSPENDED:
3551         break;
3552     default:
3553         errno = EPROTO;
3554         return (-1);
3555     }

```



```

3557     return (0);
3558 }

3560 /*
3561  * Wait for the LWP to stop for any reason.
3562  */
3563 int
3564 Lwait(struct ps_lwphandle *L, uint_t msec)
3565 {
3566     return (Lstopstatus(L, PCWSTOP, msec));
3567 }

3569 /*
3570  * Direct the LWP to stop; wait for it to stop.
3571  */
3572 int
3573 Lstop(struct ps_lwphandle *L, uint_t msec)
3574 {
3575     return (Lstopstatus(L, PCSTOP, msec));
3576 }

3578 /*
3579  * Direct the LWP to stop; don't wait.
3580  */
3581 int
3582 Ldstop(struct ps_lwphandle *L)
3583 {
3584     return (Lstopstatus(L, PCDSTOP, 0));
3585 }

3587 /*
3588  * Get the value of one register from stopped LWP.
3589  */
3590 int
3591 Lgetareg(struct ps_lwphandle *L, int regno, prgreg_t *preg)
3592 {
3593     if (regno < 0 || regno >= NPRGREG) {
3594         errno = EINVAL;
3595         return (-1);
3596     }

3598     if (L->lwp_state != PS_STOP) {
3599         errno = EBUSY;
3600         return (-1);
3601     }

3603     *preg = L->lwp_status.pr_reg[regno];
3604     return (0);
3605 }

3607 /*
3608  * Put value of one register into stopped LWP.
3609  */
3610 int
3611 Lputareg(struct ps_lwphandle *L, int regno, prgreg_t reg)
3612 {
3613     if (regno < 0 || regno >= NPRGREG) {
3614         errno = EINVAL;
3615         return (-1);
3616     }

3618     if (L->lwp_state != PS_STOP) {
3619         errno = EBUSY;
3620         return (-1);
3621     }

```

```

3623     L->lwp_status.pr_reg[regno] = reg;
3624     L->lwp_flags |= SETREGS;      /* set registers before continuing */
3625     return (0);
3626 }

3628 int
3629 Lsetrun(struct ps_lwphandle *L,
3630         int sig,      /* signal to pass to LWP */
3631         int flags) /* PRSTEP|PRABORT|PRSTOP|PRCSIG|PRCFAULT */
3632 {
3633     int ctldfd = L->lwp_ctldfd;
3634     int sbits = (PR_DSTOP | PR_ISTOP | PR_ASLEEP);

3636     long ctl[1 +          /* PCCFAULT */
3637           1 + sizeof (siginfo_t)/sizeof (long) + /* PCSSIG/PCCSIG */
3638           2 ];          /* PCRUN */

3640     long *ctlp = ctl;
3641     size_t size;

3643     if (L->lwp_state != PS_STOP &&
3644         (L->lwp_status.pr_flags & sbits) == 0) {
3645         errno = EBUSY;
3646         return (-1);
3647     }

3649     Lsync(L);      /* flush registers */

3651     if (flags & PRCFAULT) {      /* clear current fault */
3652         *ctlp++ = PCCFAULT;
3653         flags &= ~PRCFAULT;
3654     }

3656     if (flags & PRCSIG) {      /* clear current signal */
3657         *ctlp++ = PCCSIG;
3658         flags &= ~PRCSIG;
3659     } else if (sig && sig != L->lwp_status.pr_cursig) {
3660         /* make current signal */
3661         siginfo_t *infop;

3663         *ctlp++ = PCSSIG;
3664         infop = (siginfo_t *)ctlp;
3665         (void) memset(infop, 0, sizeof (*infop));
3666         infop->si_signo = sig;
3667         ctlp += sizeof (siginfo_t) / sizeof (long);
3668     }

3670     *ctlp++ = PCRUN;
3671     *ctlp++ = flags;
3672     size = (char *)ctlp - (char *)ctl;

3674     L->lwp_proc->info_valid = 0; /* will need to update map and file info */
3675     L->lwp_proc->state = PS_RUN;
3676     L->lwp_state = PS_RUN;

3678     if (write(ctldfd, ctl, size) != size) {
3679         /* Pretend that a job-stopped LWP is running */
3680         if (errno != EBUSY || L->lwp_status.pr_why != PR_JOBCONTROL)
3681             return (Lstopstatus(L, PCNULL, 0));
3682     }

3684     return (0);
3685 }

3687 int
3688 Lclearsig(struct ps_lwphandle *L)

```

```

3689 {
3690     int ctlfd = L->lwp_ctlfd;
3691     long ctl = PCCSIG;

3693     if (write(ctlfd, &ctl, sizeof (ctl)) != sizeof (ctl))
3694         return (-1);
3695     L->lwp_status.pr_cursig = 0;
3696     return (0);
3697 }

3699 int
3700 Lclearfault(struct ps_lwphandle *L)
3701 {
3702     int ctlfd = L->lwp_ctlfd;
3703     long ctl = PCCFAULT;

3705     if (write(ctlfd, &ctl, sizeof (ctl)) != sizeof (ctl))
3706         return (-1);
3707     return (0);
3708 }

3710 /*
3711  * Step over a breakpoint, i.e., execute the instruction that
3712  * really belongs at the breakpoint location (the current %pc)
3713  * and leave the LWP stopped at the next instruction.
3714  */
3715 int
3716 Lxecbkpt(struct ps_lwphandle *L, ulong_t saved)
3717 {
3718     struct ps_prochandle *P = L->lwp_proc;
3719     int rv, error;

3721     if (L->lwp_state != PS_STOP) {
3722         errno = EBUSY;
3723         return (-1);
3724     }

3726     Lsync(L);
3727     error = execute_bkpt(L->lwp_ctlfd,
3728         &P->status.pr_filttrace, &L->lwp_status.pr_lwphold,
3729         L->lwp_status.pr_reg[R_PC], saved);
3730     rv = Lstopstatus(L, PCNULL, 0);

3732     if (error != 0) {
3733         if (L->lwp_status.pr_why == PR_JOBCONTROL &&
3734             error == EBUSY) { /* jobcontrol stop -- back off */
3735             L->lwp_state = PS_RUN;
3736             return (0);
3737         }
3738         if (error == ENOENT)
3739             return (0);
3740         errno = error;
3741         return (-1);
3742     }

3744     return (rv);
3745 }

3747 /*
3748  * Step over a watchpoint, i.e., execute the instruction that was stopped by
3749  * the watchpoint, and then leave the LWP stopped at the next instruction.
3750  */
3751 int
3752 Lxecwapt(struct ps_lwphandle *L, const prwatch_t *wp)
3753 {
3754     struct ps_prochandle *P = L->lwp_proc;

```

```

3755     int rv, error;

3757     if (L->lwp_state != PS_STOP) {
3758         errno = EBUSY;
3759         return (-1);
3760     }

3762     Lsync(L);
3763     error = execute_wapt(L->lwp_ctlfd,
3764         &P->status.pr_filttrace, &L->lwp_status.pr_lwphold, wp);
3765     rv = Lstopstatus(L, PCNULL, 0);

3767     if (error != 0) {
3768         if (L->lwp_status.pr_why == PR_JOBCONTROL &&
3769             error == EBUSY) { /* jobcontrol stop -- back off */
3770             L->lwp_state = PS_RUN;
3771             return (0);
3772         }
3773         if (error == ENOENT)
3774             return (0);
3775         errno = error;
3776         return (-1);
3777     }

3779     return (rv);
3780 }

3782 int
3783 Lstack(struct ps_lwphandle *L, stack_t *stkp)
3784 {
3785     struct ps_prochandle *P = L->lwp_proc;
3786     uintptr_t addr = L->lwp_status.pr_ustack;

3788     if (P->status.pr_dmodel == PR_MODEL_NATIVE) {
3789         if (Pread(P, stkp, sizeof (*stkp), addr) != sizeof (*stkp))
3790             return (-1);
3791 #ifdef LP64
3792     } else {
3793         stack32_t stk32;

3795         if (Pread(P, &stk32, sizeof (stk32), addr) != sizeof (stk32))
3796             return (-1);

3798         stack_32_to_n(&stk32, stkp);
3799 #endif
3800     }

3802     return (0);
3803 }

3805 int
3806 Lmain_stack(struct ps_lwphandle *L, stack_t *stkp)
3807 {
3808     struct ps_prochandle *P = L->lwp_proc;

3810     if (Lstack(L, stkp) != 0)
3811         return (-1);

3813     /*
3814     * If the SS_ONSTACK flag is set then this LWP is operating on the
3815     * alternate signal stack. We can recover the original stack from
3816     * pr_oldcontext.
3817     */
3818     if (!(stkp->ss_flags & SS_ONSTACK))
3819         return (0);

```

```

3821     if (P->status.pr_dmodel == PR_MODEL_NATIVE) {
3822         ucontext_t *ctxp = (void *)L->lwp_status.pr_oldcontext;

3824         if (Pread(P, stkp, sizeof (*stkp),
3825             (uintptr_t)&ctxp->uc_stack) != sizeof (*stkp))
3826             return (-1);
3827 #ifdef _LP64
3828     } else {
3829         ucontext32_t *ctxp = (void *)L->lwp_status.pr_oldcontext;
3830         stack32_t stk32;

3832         if (Pread(P, &stk32, sizeof (stk32),
3833             (uintptr_t)&ctxp->uc_stack) != sizeof (stk32))
3834             return (-1);

3836         stack_32_to_n(&stk32, stkp);
3837 #endif
3838     }

3840     return (0);
3841 }

3843 int
3844 lalt_stack(struct ps_lwphandle *L, stack_t *stkp)
3845 {
3846     if (L->lwp_status.pr_altstack.ss_flags & SS_DISABLE) {
3847         errno = ENODATA;
3848         return (-1);
3849     }

3851     *stkp = L->lwp_status.pr_altstack;

3853     return (0);
3854 }

3856 /*
3857  * Add a mapping to the given proc handle.  Resizes the array as appropriate and
3858  * manages reference counts on the given file_info_t.
3859  *
3860  * The 'map_relocate' member is used to tell Psort_mappings() that the
3861  * associated file_map pointer needs to be relocated after the mappings have
3862  * been sorted.  It is only set for the first mapping, and has no meaning
3863  * outside these two functions.
3864  */
3865 int
3866 Padd_mapping(struct ps_prochandle *P, off64_t off, file_info_t *fp,
3867     prmap_t *pmap)
3868 {
3869     map_info_t *mp;

3871     if (P->map_count == P->map_alloc) {
3872         size_t next = P->map_alloc ? P->map_alloc * 2 : 16;

3874         if ((P->mappings = realloc(P->mappings,
3875             next * sizeof (map_info_t))) == NULL)
3876             return (-1);

3878         P->map_alloc = next;
3879     }

3881     mp = &P->mappings[P->map_count++];

3883     mp->map_offset = off;
3884     mp->map_pmap = *pmap;
3885     mp->map_relocate = 0;
3886     if ((mp->map_file = fp) != NULL) {

```

```

3887         if (fp->file_map == NULL) {
3888             fp->file_map = mp;
3889             mp->map_relocate = 1;
3890         }
3891         fp->file_ref++;
3892     }

3894     return (0);
3895 }

3897 static int
3898 map_sort(const void *a, const void *b)
3899 {
3900     const map_info_t *ap = a, *bp = b;

3902     if (ap->map_pmap.pr_vaddr < bp->map_pmap.pr_vaddr)
3903         return (-1);
3904     else if (ap->map_pmap.pr_vaddr > bp->map_pmap.pr_vaddr)
3905         return (1);
3906     else
3907         return (0);
3908 }

3910 /*
3911  * Sort the current set of mappings.  Should be called during target
3912  * initialization after all calls to Padd_mapping() have been made.
3913  */
3914 void
3915 Psort_mappings(struct ps_prochandle *P)
3916 {
3917     int i;
3918     map_info_t *mp;

3920     qsort(P->mappings, P->map_count, sizeof (map_info_t), map_sort);

3922     /*
3923      * Update all the file_map pointers to refer to the new locations.
3924      */
3925     for (i = 0; i < P->map_count; i++) {
3926         mp = &P->mappings[i];
3927         if (mp->map_relocate)
3928             mp->map_file->file_map = mp;
3929         mp->map_relocate = 0;
3930     }
3931 }

3933 struct ps_prochandle *
3934 Pgrab_ops(pid_t pid, void *data, const ps_ops_t *ops, int flags)
3935 {
3936     struct ps_prochandle *P;

3938     if ((P = calloc(1, sizeof (*P))) == NULL) {
3939         return (NULL);
3940     }

3942     Pinit_ops(&P->ops, ops);
3943     (void) mutex_init(&P->proc_lock, USYNC_THREAD, NULL);
3944     P->pid = pid;
3945     P->state = PS_STOP;
3946     P->asfd = -1;
3947     P->ctlfd = -1;
3948     P->statfd = -1;
3949     P->agentctlfd = -1;
3950     P->agentstatfd = -1;
3951     Pinit_sym(P);
3952     P->data = data;

```

```
3953     Pread_status(P);
3955     if (flags & PGRAB_INCORE) {
3956         P->flags |= INCORE;
3957     }
3959     return (P);
3960 }
```

```

*****
12883 Thu Jun 30 22:02:18 2016
new/usr/src/lib/libproc/common/Pcontrol.h
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
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 /*
26 * Copyright 2012 DEY Storage Systems, Inc. All rights reserved.
27 * Copyright (c) 2014, Joyent, Inc. All rights reserved.
28 * Copyright (c) 2013 by Delphix. All rights reserved.
29 */

31 #ifndef _PCONTROL_H
32 #define _PCONTROL_H

34 /*
35  * Implementation-specific include file for libproc process management.
36  * This is not to be seen by the clients of libproc.
37  */

39 #include <stdio.h>
40 #include <gelf.h>
41 #include <synch.h>
42 #include <procfs.h>
43 #include <rtld_db.h>
44 #include <libproc.h>
45 #include <libctf.h>
46 #include <limits.h>
47 #include <libproc.h>
48 #include <sys/secflags.h>
49 #endif /* ! codereview */

51 #ifdef __cplusplus
52 extern "C" {
53 #endif

55 #include "Putil.h"

57 /*
58  * Definitions of the process control structures, internal to libproc.

```

```

59  * These may change without affecting clients of libproc.
60  */

62 /*
63  * sym_tbl_t contains a primary and an (optional) auxiliary symbol table, which
64  * we wish to treat as a single logical symbol table. In this logical table,
65  * the data from the auxiliary table precedes that from the primary. Symbol
66  * indices start at [0], which is the first item in the auxiliary table
67  * if there is one. The sole purpose for this is so that we can treat the
68  * combination of .SUNW_ldynsym and .dynsym sections as a logically single
69  * entity without having to violate the public interface to libelf.
70  *
71  * Both tables must share the same string table section.
72  *
73  * The symtab_getsym() function serves as a gelf_getsym() replacement
74  * that is aware of the two tables and makes them look like a single table
75  * to the caller.
76  *
77  */
78 typedef struct sym_tbl {
79     Elf_Data *sym_data_pri; /* primary table */
80     Elf_Data *sym_data_aux; /* auxiliary table */
81     size_t sym_symn_aux; /* number of entries in auxiliary table */
82     size_t sym_symn; /* total number of entries in both tables */
83     char *sym_strs; /* ptr to strings */
84     size_t sym_strsz; /* size of string table */
85     GElf_Shdr sym_hdr_pri; /* primary symbol table section header */
86     GElf_Shdr sym_hdr_aux; /* auxiliary symbol table section header */
87     GElf_Shdr sym_strhdr; /* string table section header */
88     Elf *sym_elf; /* faked-up ELF handle from core file */
89     void *sym_elfmem; /* data for faked-up ELF handle */
90     uint_t *sym_byname; /* symbols sorted by name */
91     uint_t *sym_byaddr; /* symbols sorted by addr */
92     size_t sym_count; /* number of symbols in each sorted list */
93 } sym_tbl_t;

95 typedef struct file_info {
96     plist_t file_list; /* symbol information for a mapped file */
97     char file_pname[PRMAPSZ]; /* linked list */
98     struct map_info *file_map; /* name from prmap_t */
99     int file_ref; /* primary (text) mapping */
100    int file_fd; /* references from map_info_t structures */
101    int file_init; /* file descriptor for the mapped file */
102    GElf_Half file_etype; /* 0: initialization yet to be performed */
103    GElf_Half file_class; /* ELF e_type from ehdr */
104    rd_loadobj_t *file_lo; /* ELF e_ident[EI_CLASS] from ehdr */
105    char *file_lname; /* load object structure from rtld_db */
106    char *file_lbase; /* load object name from rtld_db */
107    char *file_rname; /* pointer to basename of file_lname */
108    char *file_rbase; /* resolved on-disk object pathname */
109    Elf *file_elf; /* pointer to basename of file_rname */
110    void *file_elfmem; /* ELF handle so we can close */
111    sym_tbl_t file_symtab; /* data for faked-up ELF handle */
112    sym_tbl_t file_dynsym; /* symbol table */
113    uintptr_t file_dyn_base; /* dynamic symbol table */
114    uintptr_t file_plt_base; /* load address for ET_DYN files */
115    size_t file_plt_size; /* base address for PLT */
116    uintptr_t file_jmp_rel; /* size of PLT region */
117    uintptr_t file_ctf_off; /* base address of PLT relocations */
118    size_t file_ctf_size; /* offset of CTF data in object file */
119    int file_ctf_dyn; /* size of CTF data in object file */
120    void *file_ctf_buf; /* does the CTF data reference the dynsym */
121    ctf_file_t *file_ctf; /* CTF data for this file */
122    char *file_shstrs; /* CTF container for this file */
123    size_t file_shstrsz; /* section header string table */
124    uintptr_t *file_saddr; /* section header string table */

```

```

125     uint_t file_nsaddrs; /* number of section header addresses */
126 } file_info_t;

128 typedef struct map_info { /* description of an address space mapping */
129     pmap_t map_pmap; /* /proc description of this mapping */
130     file_info_t *map_file; /* pointer into list of mapped files */
131     off64_t map_offset; /* offset into core file (if core) */
132     int map_relocate; /* associated file_map needs to be relocated */
133 } map_info_t;

135 typedef struct lwp_info { /* per-lwp information from core file */
136     plist_t lwp_list; /* linked list */
137     lwpid_t lwp_id; /* lwp identifier */
138     lwpsinfo_t lwp_psinfo; /* /proc/<pid>/lwp/<lwpid>/lwpsinfo data */
139     lwpstatus_t lwp_status; /* /proc/<pid>/lwp/<lwpid>/lwpstatus data */
140 #if defined(sparc) || defined(__sparc)
141     gwindows_t *lwp_gwins; /* /proc/<pid>/lwp/<lwpid>/gwindows data */
142     prxregset_t *lwp_xregs; /* /proc/<pid>/lwp/<lwpid>/xregs data */
143     int64_t *lwp_asrs; /* /proc/<pid>/lwp/<lwpid>/asrs data */
144 #endif
145 } lwp_info_t;

147 typedef struct fd_info { /* linked list */
148     plist_t fd_list; /* linked list */
149     prfdinfo_t fd_info; /* fd info */
150 } fd_info_t;

152 typedef struct core_info { /* information specific to core files */
153     char core_dmodel; /* data model for core file */
154     char core_osabi; /* ELF OS ABI */
155     int core_errno; /* error during initialization if != 0 */
156     plist_t core_lwp_head; /* head of list of lwp info */
157     lwp_info_t *core_lwp; /* current lwp information */
158     uint_t core_nlwp; /* number of lwp's in list */
159     off64_t core_size; /* size of core file in bytes */
160     char *core_platform; /* platform string from core file */
161     struct utsname *core_uts; /* uname(2) data from core file */
162     prcred_t *core_cred; /* process credential from core file */
163     core_content_t core_content; /* content dumped to core file */
164     ppriv_t *core_priv; /* process privileges from core file */
165     size_t core_priv_size; /* size of the privileges */
166     void *core_privinfo; /* system privileges info from core file */
167     priv_impl_info_t *core_ppii; /* NOTE entry for core_privinfo */
168     char *core_zonename; /* zone name from core file */
169     prsecflags_t *core_secflags; /* secflags from core file */
170 #endif /* ! codereview */
171 #if defined(__i386) || defined(__amd64)
172     struct ssd *core_ldt; /* LDT entries from core file */
173     uint_t core_nldt; /* number of LDT entries in core file */
174 #endif
175 } core_info_t;

177 typedef struct elf_file_header { /* extended ELF header */
178     unsigned char e_ident[EI_NIDENT];
179     Elf64_Half e_type;
180     Elf64_Half e_machine;
181     Elf64_Word e_version;
182     Elf64_Addr e_entry;
183     Elf64_Off e_phoff;
184     Elf64_Off e_shoff;
185     Elf64_Word e_flags;
186     Elf64_Half e_ehsize;
187     Elf64_Half e_phentsize;
188     Elf64_Half e_shentsize;
189     Elf64_Word e_phnum; /* phdr count extended to 32 bits */
190     Elf64_Word e_shnum; /* shdr count extended to 32 bits */

```

```

191     Elf64_Word e_shstrndx; /* shdr string index extended to 32 bits */
192 } elf_file_header_t;

194 typedef struct elf_file { /* convenience for managing ELF files */
195     elf_file_header_t e_hdr; /* Extended ELF header */
196     Elf *e_elf; /* ELF library handle */
197     int e_fd; /* file descriptor */
198 } elf_file_t;

200 #define HASHSIZE 1024 /* hash table size, power of 2 */

202 struct ps_prochandle {
203     struct ps_lwphandle **hashtab; /* hash table for LWPs (Lgrab()) */
204     mutex_t proc_lock; /* protects hash table; serializes Lgrab() */
205     pstatus_t orig_status; /* remembered status on Pgrab() */
206     pstatus_t status; /* status when stopped */
207     psinfo_t psinfo; /* psinfo_t from last Ppsinfo() request */
208     uintptr_t sysaddr; /* address of most recent syscall instruction */
209     pid_t pid; /* process-ID */
210     int state; /* state of the process, see "libproc.h" */
211     uint_t flags; /* see defines below */
212     uint_t agentcnt; /* Pcreate_agent()/Pdestroy_agent() ref count */
213     int asfd; /* /proc/<pid>/as filedescriptor */
214     int ctldfd; /* /proc/<pid>/ctl filedescriptor */
215     int statfd; /* /proc/<pid>/status filedescriptor */
216     int agentctldfd; /* /proc/<pid>/lwp/agent/ctl */
217     int agentstatfd; /* /proc/<pid>/lwp/agent/status */
218     int info_valid; /* if zero, map and file info need updating */
219     map_info_t *mappings; /* cached process mappings */
220     size_t map_count; /* number of mappings */
221     size_t map_alloc; /* number of mappings allocated */
222     uint_t num_files; /* number of file elements in file_info */
223     plist_t file_head; /* head of mapped files w/ symbol table info */
224     char *execname; /* name of the executable file */
225     auxv_t *auxv; /* the process's aux vector */
226     int nauxv; /* number of aux vector entries */
227     rd_agent_t *rap; /* cookie for rtld_db */
228     map_info_t *map_exec; /* the mapping for the executable file */
229     map_info_t *map_ldso; /* the mapping for ld.so.1 */
230     ps_ops_t ops; /* ops-vector */
231     uintptr_t *ucaddrs; /* ucontext-list addresses */
232     uint_t ucnelems; /* number of elements in the ucaddrs list */
233     char *zoneroot; /* cached path to zone root */
234     plist_t fd_head; /* head of file desc info list */
235     int num_fd; /* number of file descs in list */
236     uintptr_t map_missing; /* first missing mapping in core due to sig */
237     signfo_t killinfo; /* signal that interrupted core dump */
238     psinfo_t spymaster; /* agent LWP's spymaster, if any */
239     void *data; /* private data */
240 };

242 /* flags */
243 #define CREATED 0x01 /* process was created by Pcreate() */
244 #define SETSIG 0x02 /* set signal trace mask before continuing */
245 #define SETFAULT 0x04 /* set fault trace mask before continuing */
246 #define SETENTRY 0x08 /* set sysentry trace mask before continuing */
247 #define SETEXIT 0x10 /* set sysexit trace mask before continuing */
248 #define SETHOLD 0x20 /* set signal hold mask before continuing */
249 #define SETREGS 0x40 /* set registers before continuing */
250 #define INCORE 0x80 /* use in-core data to build symbol tables */

252 struct ps_lwphandle {
253     struct ps_prochandle *lwp_proc; /* process to which this lwp belongs */
254     struct ps_lwphandle *lwp_hash; /* hash table linked list */
255     lwpstatus_t lwp_status; /* status when stopped */
256     lwpsinfo_t lwp_psinfo; /* lwpsinfo_t from last Lpsinfo() */

```

```

257     lwpid_t      lwp_id;      /* lwp identifier */
258     int          lwp_state;    /* state of the lwp, see "libproc.h" */
259     uint_t       lwp_flags;    /* SETHOLD and/or SETREGS */
260     int          lwp_ctlfd;    /* /proc/<pid>/lwp/<lwpid>/lwpctl */
261     int          lwp_statfd;   /* /proc/<pid>/lwp/<lwpid>/lwpstatus */
262 };

264 /*
265  * Implementation functions in the process control library.
266  * These are not exported to clients of the library.
267  */
268 extern void prldump(const char *, lwpstatus_t *);
269 extern int dupfd(int, int);
270 extern int set_minfd(void);
271 extern int Pscantext(struct ps_prochandle *);
272 extern void Pinitsym(struct ps_prochandle *);
273 extern void Preadauxvec(struct ps_prochandle *);
274 extern void optimize_syntab(sym_tbl_t *);
275 extern void Pbuild_file_syntab(struct ps_prochandle *, file_info_t *);
276 extern ctf_file_t *Pbuild_file_ctf(struct ps_prochandle *, file_info_t *);
277 extern map_info_t *Paddr2mptr(struct ps_prochandle *, uintptr_t);
278 extern char *Pfindxec(struct ps_prochandle *, const char *,
279 int (*)(const char *, void *), void *);
280 extern int getlwpstatus(struct ps_prochandle *, lwpid_t, lwpstatus_t *);
281 int Pstopstatus(struct ps_prochandle *, long, uint32_t);
282 extern file_info_t *file_info_new(struct ps_prochandle *, map_info_t *);
283 extern char *Plofspath(const char *, char *, size_t);
284 extern char *Pzoneroot(struct ps_prochandle *, char *, size_t);
285 extern char *Pzonepath(struct ps_prochandle *, const char *, char *,
286 size_t);
287 extern fd_info_t *Pfd2info(struct ps_prochandle *, int);

289 extern char *Pfindmap(struct ps_prochandle *, map_info_t *, char *,
290 size_t);

292 extern int Padd_mapping(struct ps_prochandle *, off64_t, file_info_t *,
293 prmap_t *);
294 extern void Psort_mappings(struct ps_prochandle *);

296 extern char procfs_path[PATH_MAX];

298 /*
299  * Architecture-dependent definition of the breakpoint instruction.
300  */
301 #if defined(sparc) || defined(__sparc)
302 #define BPT ((instr_t)0x91d02001)
303 #elif defined(__i386) || defined(__amd64)
304 #define BPT ((instr_t)0xcc)
305 #endif

307 /*
308  * Simple convenience.
309  */
310 #define TRUE 1
311 #define FALSE 0

313 #ifdef __cplusplus
314 }
315 #endif
317 #endif /* _PCONTROL_H */

```

```

*****
75155 Thu Jun 30 22:02:19 2016
new/usr/src/lib/libproc/common/Pcore.c
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
_____unchanged_portion_omitted_____

160 /*ARGSUSED*/
161 static int
162 Psecflags_core(struct ps_prochandle *P, prsecflags_t **psf, void *data)
163 {
164     core_info_t *core = data;

166     if (core->core_secflags == NULL) {
167         errno = ENODATA;
168         return (-1);
169     }

171     if ((*psf = calloc(1, sizeof (prsecflags_t))) == NULL)
172         return (-1);

174     (void) memcpy(*psf, core->core_secflags, sizeof (prsecflags_t));

176     return (0);
177 }

179 /*ARGSUSED*/
180 static int
181 #endif /* ! codereview */
182 Ppriv_core(struct ps_prochandle *P, prpriv_t **pprv, void *data)
183 {
184     core_info_t *core = data;

186     if (core->core_priv == NULL) {
187         errno = ENODATA;
188         return (-1);
189     }

191     *pprv = malloc(core->core_priv_size);
192     if (*pprv == NULL) {
193         return (-1);
194     }

196     (void) memcpy(*pprv, core->core_priv, core->core_priv_size);
197     return (0);
198 }

200 /*ARGSUSED*/
201 static const psinfo_t *
202 Ppsinfo_core(struct ps_prochandle *P, psinfo_t *psinfo, void *data)
203 {
204     return (&P->psinfo);
205 }

207 /*ARGSUSED*/
208 static void
209 Pfini_core(struct ps_prochandle *P, void *data)
210 {
211     core_info_t *core = data;

213     if (core != NULL) {
214         extern void __priv_free_info(void *);
215         lwp_info_t *nlwp, *lwp = list_next(&core->core_lwp_head);

```

```

216         int i;

218         for (i = 0; i < core->core_nlwp; i++, lwp = nlwp) {
219             nlwp = list_next(lwp);
220             #ifdef __sparc
221                 if (lwp->lwp_gwins != NULL)
222                     free(lwp->lwp_gwins);
223                 if (lwp->lwp_xregs != NULL)
224                     free(lwp->lwp_xregs);
225                 if (lwp->lwp_asrs != NULL)
226                     free(lwp->lwp_asrs);
227             #endif
228                 free(lwp);
229         }

231         if (core->core_platform != NULL)
232             free(core->core_platform);
233         if (core->core_uts != NULL)
234             free(core->core_uts);
235         if (core->core_cred != NULL)
236             free(core->core_cred);
237         if (core->core_priv != NULL)
238             free(core->core_priv);
239         if (core->core_privinfo != NULL)
240             __priv_free_info(core->core_privinfo);
241         if (core->core_ppii != NULL)
242             free(core->core_ppii);
243         if (core->core_zonename != NULL)
244             free(core->core_zonename);
245         if (core->core_secflags != NULL)
246             free(core->core_secflags);
247     #endif /* ! codereview */
248     #ifdef __x86
249         if (core->core_ldt != NULL)
250             free(core->core_ldt);
251     #endif

253     free(core);
254 }
255 }

257 /*ARGSUSED*/
258 static char *
259 Pplatform_core(struct ps_prochandle *P, char *s, size_t n, void *data)
260 {
261     core_info_t *core = data;

263     if (core->core_platform == NULL) {
264         errno = ENODATA;
265         return (NULL);
266     }
267     (void) strncpy(s, core->core_platform, n - 1);
268     s[n - 1] = '\0';
269     return (s);
270 }

272 /*ARGSUSED*/
273 static int
274 Puname_core(struct ps_prochandle *P, struct utsname *u, void *data)
275 {
276     core_info_t *core = data;

278     if (core->core_uts == NULL) {
279         errno = ENODATA;
280         return (-1);
281     }

```



```

282     (void) memcpy(u, core->core_uts, sizeof (struct utsname));
283     return (0);
284 }

286 /*ARGSUSED*/
287 static char *
288 Pzonename_core(struct ps_prochandle *P, char *s, size_t n, void *data)
289 {
290     core_info_t *core = data;

292     if (core->core_zonename == NULL) {
293         errno = ENODATA;
294         return (NULL);
295     }
296     (void) strncpy(s, core->core_zonename, n);
297     return (s);
298 }

300 #ifdef __x86
301 /*ARGSUSED*/
302 static int
303 Pldt_core(struct ps_prochandle *P, struct ssd *pldt, int nldt, void *data)
304 {
305     core_info_t *core = data;

307     if (pldt == NULL || nldt == 0)
308         return (core->core_nldt);

310     if (core->core_ldt != NULL) {
311         nldt = MIN(nldt, core->core_nldt);

313         (void) memcpy(pldt, core->core_ldt,
314                       nldt * sizeof (struct ssd));

316         return (nldt);
317     }

319     errno = ENODATA;
320     return (-1);
321 }
322 #endif

324 static const ps_ops_t P_core_ops = {
325     .pop_pread    = Pread_core,
326     .pop_pwrite   = Pwrite_core,
327     .pop_cred     = Pcred_core,
328     .pop_priv     = Ppriv_core,
329     .pop_psinfo   = Ppsinfo_core,
330     .pop_fini     = Pfini_core,
331     .pop_platform = Pplatform_core,
332     .pop_uname    = Puname_core,
333     .pop_zonename = Pzonename_core,
334     .pop_secflags = Psecflags_core,
335 #endif /* ! codereview */
336 #ifdef __x86
337     .pop_ldt      = Pldt_core
338 #endif
339 };

341 /*
342  * Return the lwp_info_t for the given lwpid.  If no such lwpid has been
343  * encountered yet, allocate a new structure and return a pointer to it.
344  * Create a list of lwp_info_t structures sorted in decreasing lwp_id order.
345  */
346 static lwp_info_t *
347 lwpid2info(struct ps_prochandle *P, lwpid_t id)

```

```

348 {
349     core_info_t *core = P->data;
350     lwp_info_t *lwp = list_next(&core->core_lwp_head);
351     lwp_info_t *next;
352     uint_t i;

354     for (i = 0; i < core->core_nlwp; i++, lwp = list_next(lwp)) {
355         if (lwp->lwp_id == id) {
356             core->core_lwp = lwp;
357             return (lwp);
358         }
359         if (lwp->lwp_id < id) {
360             break;
361         }
362     }

364     next = lwp;
365     if ((lwp = calloc(1, sizeof (lwp_info_t))) == NULL)
366         return (NULL);

368     list_link(lwp, next);
369     lwp->lwp_id = id;

371     core->core_lwp = lwp;
372     core->core_nlwp++;

374     return (lwp);
375 }

377 /*
378  * The core file itself contains a series of NOTE segments containing saved
379  * structures from /proc at the time the process died.  For each note we
380  * comprehend, we define a function to read it in from the core file,
381  * convert it to our native data model if necessary, and store it inside
382  * the ps_prochandle.  Each function is invoked by Pgrab_core() with the
383  * seek pointer on P->asfd positioned appropriately.  We populate a table
384  * of pointers to these note functions below.
385  */

387 static int
388 note_pstatus(struct ps_prochandle *P, size_t nbytes)
389 {
390     #ifdef _LP64
391         core_info_t *core = P->data;

393         if (core->core_dmodel == PR_MODEL_ILP32) {
394             pstatus32_t ps32;

396             if (nbytes < sizeof (pstatus32_t) ||
397                 read(P->asfd, &ps32, sizeof (ps32)) != sizeof (ps32))
398                 goto err;

400             pstatus_32_to_n(&ps32, &P->status);

402         } else
403         #endif
404         if (nbytes < sizeof (pstatus_t) ||
405             read(P->asfd, &P->status, sizeof (pstatus_t)) != sizeof (pstatus_t))
406             goto err;

408     P->orig_status = P->status;
409     P->pid = P->status.pr_pid;

411     return (0);

413 err:

```

```

414     dprintf("Pgrab_core: failed to read NT_PSTATUS\n");
415     return (-1);
416 }

418 static int
419 note_lwpstatus(struct ps_prochandle *P, size_t nbytes)
420 {
421     lwp_info_t *lwp;
422     lwpstatus_t lps;

424 #ifdef _LP64
425     core_info_t *core = P->data;

427     if (core->core_dmodel == PR_MODEL_ILP32) {
428         lwpstatus32_t l32;

430         if (nbytes < sizeof (lwpstatus32_t) ||
431             read(P->asfd, &l32, sizeof (l32)) != sizeof (l32))
432             goto err;

434         lwpstatus_32_to_n(&l32, &lps);
435     } else
436 #endif
437     if (nbytes < sizeof (lwpstatus_t) ||
438         read(P->asfd, &lps, sizeof (lps)) != sizeof (lps))
439         goto err;

441     if ((lwp = lwpid2info(P, lps.pr_lwpid)) == NULL) {
442         dprintf("Pgrab_core: failed to add NT_LWPSTATUS\n");
443         return (-1);
444     }

446     /*
447     * Erase a useless and confusing artifact of the kernel implementation:
448     * the lwps which did *not* create the core will show SIGKILL. We can
449     * be assured this is bogus because SIGKILL can't produce core files.
450     */
451     if (lps.pr_cursig == SIGKILL)
452         lps.pr_cursig = 0;

454     (void) memcpy(&lwp->lwp_status, &lps, sizeof (lps));
455     return (0);

457 err:
458     dprintf("Pgrab_core: failed to read NT_LWPSTATUS\n");
459     return (-1);
460 }

462 #ifdef __x86
464 static void
465 lx_prpsinfo32_to_psinfo(lx_prpsinfo32_t *p32, psinfo_t *psinfo)
466 {
467     psinfo->pr_flag = p32->pr_flag;
468     psinfo->pr_pid = p32->pr_pid;
469     psinfo->pr_ppid = p32->pr_ppid;
470     psinfo->pr_uid = p32->pr_uid;
471     psinfo->pr_gid = p32->pr_gid;
472     psinfo->pr_sid = p32->pr_sid;
473     psinfo->pr_pgid = p32->pr_pgrp;

475     (void) memcpy(psinfo->pr_fname, p32->pr_fname,
476                 sizeof (psinfo->pr_fname));
477     (void) memcpy(psinfo->pr_psargs, p32->pr_psargs,
478                 sizeof (psinfo->pr_psargs));
479 }

```

```

481 static void
482 lx_prpsinfo64_to_psinfo(lx_prpsinfo64_t *p64, psinfo_t *psinfo)
483 {
484     psinfo->pr_flag = p64->pr_flag;
485     psinfo->pr_pid = p64->pr_pid;
486     psinfo->pr_ppid = p64->pr_ppid;
487     psinfo->pr_uid = p64->pr_uid;
488     psinfo->pr_gid = p64->pr_gid;
489     psinfo->pr_sid = p64->pr_sid;
490     psinfo->pr_pgid = p64->pr_pgrp;
491     psinfo->pr_pgid = p64->pr_pgrp;

493     (void) memcpy(psinfo->pr_fname, p64->pr_fname,
494                 sizeof (psinfo->pr_fname));
495     (void) memcpy(psinfo->pr_psargs, p64->pr_psargs,
496                 sizeof (psinfo->pr_psargs));
497 }

499 static int
500 note_linux_psinfo(struct ps_prochandle *P, size_t nbytes)
501 {
502     core_info_t *core = P->data;
503     lx_prpsinfo32_t p32;
504     lx_prpsinfo64_t p64;

506     if (core->core_dmodel == PR_MODEL_ILP32) {
507         if (nbytes < sizeof (p32) ||
508             read(P->asfd, &p32, sizeof (p32)) != sizeof (p32))
509             goto err;

511         lx_prpsinfo32_to_psinfo(&p32, &P->psinfo);
512     } else {
513         if (nbytes < sizeof (p64) ||
514             read(P->asfd, &p64, sizeof (p64)) != sizeof (p64))
515             goto err;

517         lx_prpsinfo64_to_psinfo(&p64, &P->psinfo);
518     }

521     P->status.pr_pid = P->psinfo.pr_pid;
522     P->status.pr_ppid = P->psinfo.pr_ppid;
523     P->status.pr_pgid = P->psinfo.pr_pgid;
524     P->status.pr_sid = P->psinfo.pr_sid;

526     P->psinfo.pr_nlwp = 0;
527     P->status.pr_nlwp = 0;

529     return (0);
530 err:
531     dprintf("Pgrab_core: failed to read NT_PSINFO\n");
532     return (-1);
533 }

535 static void
536 lx_prstatus64_to_lwp(lx_prstatus64_t *prs64, lwp_info_t *lwp)
537 {
538     LTIME_TO_TIMESPEC(lwp->lwp_status.pr_utime, prs64->pr_utime);
539     LTIME_TO_TIMESPEC(lwp->lwp_status.pr_stime, prs64->pr_stime);

541     lwp->lwp_status.pr_reg[REG_R15] = prs64->pr_reg.lxr_r15;
542     lwp->lwp_status.pr_reg[REG_R14] = prs64->pr_reg.lxr_r14;
543     lwp->lwp_status.pr_reg[REG_R13] = prs64->pr_reg.lxr_r13;
544     lwp->lwp_status.pr_reg[REG_R12] = prs64->pr_reg.lxr_r12;
545     lwp->lwp_status.pr_reg[REG_R11] = prs64->pr_reg.lxr_r11;

```

```

546 lwp->lwp_status.pr_reg[REG_R10] = prs64->pr_reg.lxr_r10;
547 lwp->lwp_status.pr_reg[REG_R9] = prs64->pr_reg.lxr_r9;
548 lwp->lwp_status.pr_reg[REG_R8] = prs64->pr_reg.lxr_r8;

550 lwp->lwp_status.pr_reg[REG_RDI] = prs64->pr_reg.lxr_rdi;
551 lwp->lwp_status.pr_reg[REG_RSI] = prs64->pr_reg.lxr_rsi;
552 lwp->lwp_status.pr_reg[REG_RBP] = prs64->pr_reg.lxr_rbp;
553 lwp->lwp_status.pr_reg[REG_RBX] = prs64->pr_reg.lxr_rbx;
554 lwp->lwp_status.pr_reg[REG_RDX] = prs64->pr_reg.lxr_rdx;
555 lwp->lwp_status.pr_reg[REG_RCX] = prs64->pr_reg.lxr_rcx;
556 lwp->lwp_status.pr_reg[REG_RAX] = prs64->pr_reg.lxr_rax;

558 lwp->lwp_status.pr_reg[REG_RIP] = prs64->pr_reg.lxr_rip;
559 lwp->lwp_status.pr_reg[REG_CS] = prs64->pr_reg.lxr_cs;
560 lwp->lwp_status.pr_reg[REG_RSP] = prs64->pr_reg.lxr_rsp;
561 lwp->lwp_status.pr_reg[REG_FS] = prs64->pr_reg.lxr_fs;
562 lwp->lwp_status.pr_reg[REG_SS] = prs64->pr_reg.lxr_ss;
563 lwp->lwp_status.pr_reg[REG_GS] = prs64->pr_reg.lxr_gs;
564 lwp->lwp_status.pr_reg[REG_ES] = prs64->pr_reg.lxr_es;
565 lwp->lwp_status.pr_reg[REG_DS] = prs64->pr_reg.lxr_ds;

567 lwp->lwp_status.pr_reg[REG_GSBASE] = prs64->pr_reg.lxr_gs_base;
568 lwp->lwp_status.pr_reg[REG_FSBASE] = prs64->pr_reg.lxr_fs_base;
569 }

571 static void
572 lx_prstatus32_to_lwp(lx_prstatus32_t *prs32, lwp_info_t *lwp)
573 {
574     LTIME_TO_TIMESPEC(lwp->lwp_status.pr_utime, prs32->pr_utime);
575     LTIME_TO_TIMESPEC(lwp->lwp_status.pr_stime, prs32->pr_stime);

577 #ifdef __amd64
578 lwp->lwp_status.pr_reg[REG_GS] = prs32->pr_reg.lxr_gs;
579 lwp->lwp_status.pr_reg[REG_FS] = prs32->pr_reg.lxr_fs;
580 lwp->lwp_status.pr_reg[REG_DS] = prs32->pr_reg.lxr_ds;
581 lwp->lwp_status.pr_reg[REG_ES] = prs32->pr_reg.lxr_es;
582 lwp->lwp_status.pr_reg[REG_RDI] = prs32->pr_reg.lxr_di;
583 lwp->lwp_status.pr_reg[REG_RSI] = prs32->pr_reg.lxr_si;
584 lwp->lwp_status.pr_reg[REG_RBP] = prs32->pr_reg.lxr_bp;
585 lwp->lwp_status.pr_reg[REG_RBX] = prs32->pr_reg.lxr_bx;
586 lwp->lwp_status.pr_reg[REG_RDX] = prs32->pr_reg.lxr_dx;
587 lwp->lwp_status.pr_reg[REG_RCX] = prs32->pr_reg.lxr_cx;
588 lwp->lwp_status.pr_reg[REG_RAX] = prs32->pr_reg.lxr_ax;
589 lwp->lwp_status.pr_reg[REG_RIP] = prs32->pr_reg.lxr_ip;
590 lwp->lwp_status.pr_reg[REG_CS] = prs32->pr_reg.lxr_cs;
591 lwp->lwp_status.pr_reg[REG_RFL] = prs32->pr_reg.lxr_flags;
592 lwp->lwp_status.pr_reg[REG_RSP] = prs32->pr_reg.lxr_sp;
593 lwp->lwp_status.pr_reg[REG_SS] = prs32->pr_reg.lxr_ss;
594 #else /* __amd64 */
595 lwp->lwp_status.pr_reg[EBX] = prs32->pr_reg.lxr_bx;
596 lwp->lwp_status.pr_reg[ECX] = prs32->pr_reg.lxr_cx;
597 lwp->lwp_status.pr_reg[EDX] = prs32->pr_reg.lxr_dx;
598 lwp->lwp_status.pr_reg[ESI] = prs32->pr_reg.lxr_si;
599 lwp->lwp_status.pr_reg[EDI] = prs32->pr_reg.lxr_di;
600 lwp->lwp_status.pr_reg[EBP] = prs32->pr_reg.lxr_bp;
601 lwp->lwp_status.pr_reg[EAX] = prs32->pr_reg.lxr_ax;
602 lwp->lwp_status.pr_reg[EIP] = prs32->pr_reg.lxr_ip;
603 lwp->lwp_status.pr_reg[UESP] = prs32->pr_reg.lxr_sp;

605 lwp->lwp_status.pr_reg[DS] = prs32->pr_reg.lxr_ds;
606 lwp->lwp_status.pr_reg[ES] = prs32->pr_reg.lxr_es;
607 lwp->lwp_status.pr_reg[FS] = prs32->pr_reg.lxr_fs;
608 lwp->lwp_status.pr_reg[GS] = prs32->pr_reg.lxr_gs;
609 lwp->lwp_status.pr_reg[CS] = prs32->pr_reg.lxr_cs;
610 lwp->lwp_status.pr_reg[SS] = prs32->pr_reg.lxr_ss;

```

```

612 lwp->lwp_status.pr_reg[EFL] = prs32->pr_reg.lxr_flags;
613 #endif /* !__amd64 */
614 }

616 static int
617 note_linux_prstatus(struct ps_prochandle *P, size_t nbytes)
618 {
619     core_info_t *core = P->data;

621     lx_prstatus64_t prs64;
622     lx_prstatus32_t prs32;
623     lwp_info_t *lwp;
624     lwpid_t tid;

626     dprintf("looking for model %d, %ld/%ld\n", core->core_dmodel,
627           (ulong_t)nbytes, (ulong_t)sizeof(prs32));
628     if (core->core_dmodel == PR_MODEL_ILP32) {
629         if (nbytes < sizeof(prs32) ||
630             read(P->asfd, &prs32, sizeof(prs32)) != nbytes)
631             goto err;
632         tid = prs32.pr_pid;
633     } else {
634         if (nbytes < sizeof(prs64) ||
635             read(P->asfd, &prs64, sizeof(prs64)) != nbytes)
636             goto err;
637         tid = prs64.pr_pid;
638     }

640     if ((lwp = lwpid2info(P, tid)) == NULL) {
641         dprintf("Pgrab_core: failed to add lwpid2info "
642               "linux_prstatus\n");
643         return (-1);
644     }

646     P->psinfo.pr_nlwp++;
647     P->status.pr_nlwp++;

649     lwp->lwp_status.pr_lwpid = tid;

651     if (core->core_dmodel == PR_MODEL_ILP32)
652         lx_prstatus32_to_lwp(&prs32, lwp);
653     else
654         lx_prstatus64_to_lwp(&prs64, lwp);

656     return (0);
657 err:
658     dprintf("Pgrab_core: failed to read NT_PRSTATUS\n");
659     return (-1);
660 }

662 #endif /* __x86 */

664 static int
665 note_psinfo(struct ps_prochandle *P, size_t nbytes)
666 {
667     #ifdef LP64
668     core_info_t *core = P->data;

670     if (core->core_dmodel == PR_MODEL_ILP32) {
671         psinfo32_t ps32;

673         if (nbytes < sizeof(psinfo32_t) ||
674             read(P->asfd, &ps32, sizeof(ps32)) != sizeof(ps32))
675             goto err;

677         psinfo_32_to_n(&ps32, &P->psinfo);

```

```

678     } else
679 #endif
680     if (nbytes < sizeof (psinfo_t) ||
681         read(P->asfd, &P->psinfo, sizeof (psinfo_t)) != sizeof (psinfo_t))
682         goto err;

684     dprintf("pr_fname = <%s>\n", P->psinfo.pr_fname);
685     dprintf("pr_psargs = <%s>\n", P->psinfo.pr_psargs);
686     dprintf("pr_wstat = 0x%x\n", P->psinfo.pr_wstat);

688     return (0);

690 err:
691     dprintf("Pgrab_core: failed to read NT_PSINFO\n");
692     return (-1);
693 }

695 static int
696 note_lwpsinfo(struct ps_prochandle *P, size_t nbytes)
697 {
698     lwp_info_t *lwp;
699     lwpsinfo_t lps;

701 #ifdef LP64
702     core_info_t *core = P->data;

704     if (core->core_dmodel == PR_MODEL_ILP32) {
705         lwpsinfo32_t l32;

707         if (nbytes < sizeof (lwpsinfo32_t) ||
708             read(P->asfd, &l32, sizeof (l32)) != sizeof (l32))
709             goto err;

711         lwpsinfo_32_to_n(&l32, &lps);
712     } else
713 #endif
714     if (nbytes < sizeof (lwpsinfo_t) ||
715         read(P->asfd, &lps, sizeof (lps)) != sizeof (lps))
716         goto err;

718     if ((lwp = lwpid2info(P, lps.pr_lwpid)) == NULL) {
719         dprintf("Pgrab_core: failed to add NT_LWPSINFO\n");
720         return (-1);
721     }

723     (void) memcpy(&lwp->lwp_psinfo, &lps, sizeof (lps));
724     return (0);

726 err:
727     dprintf("Pgrab_core: failed to read NT_LWPSINFO\n");
728     return (-1);
729 }

731 static int
732 note_fdinfo(struct ps_prochandle *P, size_t nbytes)
733 {
734     prfdinfo_t prfd;
735     fd_info_t *fip;

737     if ((nbytes < sizeof (prfd)) ||
738         (read(P->asfd, &prfd, sizeof (prfd)) != sizeof (prfd))) {
739         dprintf("Pgrab_core: failed to read NT_FDINFO\n");
740         return (-1);
741     }

743     if ((fip = Pfd2info(P, prfd.pr_fd)) == NULL) {

```

```

744         dprintf("Pgrab_core: failed to add NT_FDINFO\n");
745         return (-1);
746     }
747     (void) memcpy(&fip->fd_info, &prfd, sizeof (prfd));
748     return (0);
749 }

751 static int
752 note_platform(struct ps_prochandle *P, size_t nbytes)
753 {
754     core_info_t *core = P->data;
755     char *plat;

757     if (core->core_platform != NULL)
758         return (0); /* Already seen */

760     if (nbytes != 0 && ((plat = malloc(nbytes + 1)) != NULL)) {
761         if (read(P->asfd, plat, nbytes) != nbytes) {
762             dprintf("Pgrab_core: failed to read NT_PLATFORM\n");
763             free(plat);
764             return (-1);
765         }
766         plat[nbytes - 1] = '\0';
767         core->core_platform = plat;
768     }

770     return (0);
771 }

773 static int
774 note_secflags(struct ps_prochandle *P, size_t nbytes)
775 {
776     core_info_t *core = P->data;
777     prsecflags_t *psf;

779     if (core->core_secflags != NULL)
780         return (0); /* Already seen */

782     if (sizeof (*psf) != nbytes) {
783         dprintf("Pgrab_core: NT_SECFLAGS changed size."
784             " Need to handle a version change?\n");
785         return (-1);
786     }

788     if (nbytes != 0 && ((psf = malloc(nbytes)) != NULL)) {
789         if (read(P->asfd, psf, nbytes) != nbytes) {
790             dprintf("Pgrab_core: failed to read NT_SECFLAGS\n");
791             free(psf);
792             return (-1);
793         }

795         core->core_secflags = psf;
796     }

798     return (0);
799 }

801 static int
802 #endif /* ! codereview */
803 note_utsname(struct ps_prochandle *P, size_t nbytes)
804 {
805     core_info_t *core = P->data;
806     size_t ubytes = sizeof (struct utsname);
807     struct utsname *utsp;

809     if (core->core_uts != NULL || nbytes < ubytes)

```

```

810         return (0);      /* Already seen or bad size */
811
812     if ((utsp = malloc(ubytes)) == NULL)
813         return (-1);
814
815     if (read(P->asfd, utsp, ubytes) != ubytes) {
816         dprintf("Pgrab_core: failed to read NT_UTSNAME\n");
817         free(utsp);
818         return (-1);
819     }
820
821     if (_libproc_debug) {
822         dprintf("uts.sysname = \"%s\"\n", utsp->sysname);
823         dprintf("uts.nodename = \"%s\"\n", utsp->nodename);
824         dprintf("uts.release = \"%s\"\n", utsp->release);
825         dprintf("uts.version = \"%s\"\n", utsp->version);
826         dprintf("uts.machine = \"%s\"\n", utsp->machine);
827     }
828
829     core->core_uts = utsp;
830     return (0);
831 }
832
833 static int
834 note_content(struct ps_prochandle *P, size_t nbytes)
835 {
836     core_info_t *core = P->data;
837     core_content_t content;
838
839     if (sizeof (core->core_content) != nbytes)
840         return (-1);
841
842     if (read(P->asfd, &content, sizeof (content)) != sizeof (content))
843         return (-1);
844
845     core->core_content = content;
846
847     dprintf("core content = %llx\n", content);
848
849     return (0);
850 }
851
852 static int
853 note_cred(struct ps_prochandle *P, size_t nbytes)
854 {
855     core_info_t *core = P->data;
856     prcred_t *pcrp;
857     int ngroups;
858     const size_t min_size = sizeof (prcred_t) - sizeof (gid_t);
859
860     /*
861      * We allow for prcred_t notes that are actually smaller than a
862      * prcred_t since the last member isn't essential if there are
863      * no group memberships. This allows for more flexibility when it
864      * comes to slightly malformed -- but still valid -- notes.
865      */
866     if (core->core_cred != NULL || nbytes < min_size)
867         return (0);      /* Already seen or bad size */
868
869     ngroups = (nbytes - min_size) / sizeof (gid_t);
870     nbytes = sizeof (prcred_t) + (ngroups - 1) * sizeof (gid_t);
871
872     if ((pcrp = malloc(nbytes)) == NULL)
873         return (-1);
874
875     if (read(P->asfd, pcrp, nbytes) != nbytes) {

```

```

876         dprintf("Pgrab_core: failed to read NT_PRCRED\n");
877         free(pcrp);
878         return (-1);
879     }
880
881     if (pcrp->pr_ngroups > ngroups) {
882         dprintf("pr_ngroups = %d; resetting to %d based on note size\n",
883             pcrp->pr_ngroups, ngroups);
884         pcrp->pr_ngroups = ngroups;
885     }
886
887     core->core_cred = pcrp;
888     return (0);
889 }
890
891 #ifdef __x86
892 static int
893 note_ldt(struct ps_prochandle *P, size_t nbytes)
894 {
895     core_info_t *core = P->data;
896     struct ssd *pldt;
897     uint_t nldt;
898
899     if (core->core_ldt != NULL || nbytes < sizeof (struct ssd))
900         return (0);      /* Already seen or bad size */
901
902     nldt = nbytes / sizeof (struct ssd);
903     nbytes = nldt * sizeof (struct ssd);
904
905     if ((pldt = malloc(nbytes)) == NULL)
906         return (-1);
907
908     if (read(P->asfd, pldt, nbytes) != nbytes) {
909         dprintf("Pgrab_core: failed to read NT_LDT\n");
910         free(pldt);
911         return (-1);
912     }
913
914     core->core_ldt = pldt;
915     core->core_nldt = nldt;
916     return (0);
917 }
918 #endif /* __i386 */
919
920 static int
921 note_priv(struct ps_prochandle *P, size_t nbytes)
922 {
923     core_info_t *core = P->data;
924     prpriv_t *pprvp;
925
926     if (core->core_priv != NULL || nbytes < sizeof (prpriv_t))
927         return (0);      /* Already seen or bad size */
928
929     if ((pprvp = malloc(nbytes)) == NULL)
930         return (-1);
931
932     if (read(P->asfd, pprvp, nbytes) != nbytes) {
933         dprintf("Pgrab_core: failed to read NT_PRPRIV\n");
934         free(pprvp);
935         return (-1);
936     }
937
938     core->core_priv = pprvp;
939     core->core_priv_size = nbytes;
940     return (0);
941 }

```

```

943 static int
944 note_priv_info(struct ps_prochandle *P, size_t nbytes)
945 {
946     core_info_t *core = P->data;
947     extern void *__priv_parse_info();
948     priv_impl_info_t *ppii;
949
950     if (core->core_privinfo != NULL ||
951         nbytes < sizeof (priv_impl_info_t))
952         return (0); /* Already seen or bad size */
953
954     if ((ppii = malloc(nbytes)) == NULL)
955         return (-1);
956
957     if (read(P->asfd, ppii, nbytes) != nbytes ||
958         PRIV_IMPL_INFO_SIZE(ppii) != nbytes) {
959         dprintf("Pgrab_core: failed to read NT_PRPRIVINFO\n");
960         free(ppii);
961         return (-1);
962     }
963
964     core->core_privinfo = __priv_parse_info(ppii);
965     core->core_ppii = ppii;
966     return (0);
967 }
968
969 static int
970 note_zonename(struct ps_prochandle *P, size_t nbytes)
971 {
972     core_info_t *core = P->data;
973     char *zonename;
974
975     if (core->core_zonename != NULL)
976         return (0); /* Already seen */
977
978     if (nbytes != 0) {
979         if ((zonename = malloc(nbytes)) == NULL)
980             return (-1);
981         if (read(P->asfd, zonename, nbytes) != nbytes) {
982             dprintf("Pgrab_core: failed to read NT_ZONENAME\n");
983             free(zonename);
984             return (-1);
985         }
986         zonename[nbytes - 1] = '\0';
987         core->core_zonename = zonename;
988     }
989
990     return (0);
991 }
992
993 static int
994 note_auxv(struct ps_prochandle *P, size_t nbytes)
995 {
996     size_t n, i;
997
998     #ifdef _LP64
999     core_info_t *core = P->data;
1000
1001     if (core->core_dmodel == PR_MODEL_ILP32) {
1002         auxv32_t *a32;
1003
1004         n = nbytes / sizeof (auxv32_t);
1005         nbytes = n * sizeof (auxv32_t);
1006         a32 = alloca(nbytes);

```

```

1008         if (read(P->asfd, a32, nbytes) != nbytes) {
1009             dprintf("Pgrab_core: failed to read NT_AUXV\n");
1010             return (-1);
1011         }
1012
1013         if ((P->auxv = malloc(sizeof (auxv_t) * (n + 1))) == NULL)
1014             return (-1);
1015
1016         for (i = 0; i < n; i++)
1017             auxv_32_to_n(&a32[i], &P->auxv[i]);
1018     } else {
1019     #endif
1020         n = nbytes / sizeof (auxv_t);
1021         nbytes = n * sizeof (auxv_t);
1022
1023         if ((P->auxv = malloc(nbytes + sizeof (auxv_t))) == NULL)
1024             return (-1);
1025
1026         if (read(P->asfd, P->auxv, nbytes) != nbytes) {
1027             free(P->auxv);
1028             P->auxv = NULL;
1029             return (-1);
1030         }
1031     #ifdef _LP64
1032     }
1033     #endif
1034 }
1035
1036     if (_libproc_debug) {
1037         for (i = 0; i < n; i++) {
1038             dprintf("P->auxv[%lu] = ( %d, 0x%lx )\n", (ulong_t)i,
1039                 P->auxv[i].a_type, P->auxv[i].a_un.a_val);
1040         }
1041     }
1042
1043     /*
1044     * Defensive coding for loops which depend upon the auxv array being
1045     * terminated by an AT_NULL element; in each case, we've allocated
1046     * P->auxv to have an additional element which we force to be AT_NULL.
1047     */
1048     P->auxv[n].a_type = AT_NULL;
1049     P->auxv[n].a_un.a_val = 0L;
1050     P->nauxv = (int)n;
1051
1052     return (0);
1053 }
1054
1055 #ifdef __sparc
1056 static int
1057 note_xreg(struct ps_prochandle *P, size_t nbytes)
1058 {
1059     core_info_t *core = P->data;
1060     lwp_info_t *lwp = core->core_lwp;
1061     size_t xbytes = sizeof (prxregset_t);
1062     prxregset_t *xregs;
1063
1064     if (lwp == NULL || lwp->lwp_xregs != NULL || nbytes < xbytes)
1065         return (0); /* No lwp yet, already seen, or bad size */
1066
1067     if ((xregs = malloc(xbytes)) == NULL)
1068         return (-1);
1069
1070     if (read(P->asfd, xregs, xbytes) != xbytes) {
1071         dprintf("Pgrab_core: failed to read NT_PRXREG\n");
1072         free(xregs);
1073         return (-1);

```

```

1074     }
1076     lwp->lwp_xregs = xregs;
1077     return (0);
1078 }

1080 static int
1081 note_gwindows(struct ps_prochandle *P, size_t nbytes)
1082 {
1083     core_info_t *core = P->data;
1084     lwp_info_t *lwp = core->core_lwp;

1086     if (lwp == NULL || lwp->lwp_gwins != NULL || nbytes == 0)
1087         return (0); /* No lwp yet or already seen or no data */

1089     if ((lwp->lwp_gwins = malloc(sizeof(gwindows_t))) == NULL)
1090         return (-1);

1092     /*
1093      * Since the amount of gwindows data varies with how many windows were
1094      * actually saved, we just read up to the minimum of the note size
1095      * and the size of the gwindows_t type. It doesn't matter if the read
1096      * fails since we have to zero out gwindows first anyway.
1097      */
1098     #ifdef LP64
1099     if (core->core_dmodel == PR_MODEL_ILP32) {
1100         gwindows32_t g32;

1102         (void) memset(&g32, 0, sizeof(g32));
1103         (void) read(P->asfd, &g32, MIN(nbytes, sizeof(g32)));
1104         gwindows_32_to_n(&g32, lwp->lwp_gwins);

1106     } else {
1107     #endif
1108         (void) memset(lwp->lwp_gwins, 0, sizeof(gwindows_t));
1109         (void) read(P->asfd, lwp->lwp_gwins,
1110             MIN(nbytes, sizeof(gwindows_t)));
1111     #ifdef LP64
1112     }
1113     #endif
1114     return (0);
1115 }

1117 #ifdef __sparcv9
1118 static int
1119 note_asrs(struct ps_prochandle *P, size_t nbytes)
1120 {
1121     core_info_t *core = P->data;
1122     lwp_info_t *lwp = core->core_lwp;
1123     int64_t *asrs;

1125     if (lwp == NULL || lwp->lwp_asrs != NULL || nbytes < sizeof(asrset_t))
1126         return (0); /* No lwp yet, already seen, or bad size */

1128     if ((asrs = malloc(sizeof(asrset_t))) == NULL)
1129         return (-1);

1131     if (read(P->asfd, asrs, sizeof(asrset_t)) != sizeof(asrset_t)) {
1132         dprintf("Pgrab_core: failed to read NT_ASRS\n");
1133         free(asrs);
1134         return (-1);
1135     }

1137     lwp->lwp_asrs = asrs;
1138     return (0);
1139 }

```

```

1140 #endif /* __sparcv9 */
1141 #endif /* __sparc */

1143 static int
1144 note_spymaster(struct ps_prochandle *P, size_t nbytes)
1145 {
1146     #ifdef LP64
1147         core_info_t *core = P->data;

1149         if (core->core_dmodel == PR_MODEL_ILP32) {
1150             psinfo32_t ps32;

1152             if (nbytes < sizeof(psinfo32_t) ||
1153                 read(P->asfd, &ps32, sizeof(ps32)) != sizeof(ps32))
1154                 goto err;

1156             psinfo_32_to_n(&ps32, &P->spymaster);
1157         } else
1158     #endif
1159     if (nbytes < sizeof(psinfo_t) || read(P->asfd,
1160         &P->spymaster, sizeof(psinfo_t)) != sizeof(psinfo_t))
1161         goto err;

1163     dprintf("spymaster pr_fname = <%s>\n", P->psinfo.pr_fname);
1164     dprintf("spymaster pr_psargs = <%s>\n", P->psinfo.pr_psargs);
1165     dprintf("spymaster pr_wstat = 0x%x\n", P->psinfo.pr_wstat);

1167     return (0);

1169 err:
1170     dprintf("Pgrab_core: failed to read NT_SPYMASTER\n");
1171     return (-1);
1172 }

1174 /*ARGSUSED*/
1175 static int
1176 note_notsup(struct ps_prochandle *P, size_t nbytes)
1177 {
1178     dprintf("skipping unsupported note type of size %ld bytes\n",
1179         (ulong_t)nbytes);
1180     return (0);
1181 }

1183 /*
1184  * Populate a table of function pointers indexed by Note type with our
1185  * functions to process each type of core file note:
1186  */
1187 static int (*nhldrs[])(struct ps_prochandle *, size_t) = {
1188     note_notsup, /* 0 unassigned */
1189     #ifdef __x86
1190     note_linux_prstatus, /* 1 NT_PRSTATUS (old) */
1191     #else
1192     note_notsup, /* 1 NT_PRSTATUS (old) */
1193     #endif
1194     note_notsup, /* 2 NT_PRFPREG (old) */
1195     #ifdef __x86
1196     note_linux_psinfo, /* 3 NT_PRPSINFO (old) */
1197     #else
1198     note_notsup, /* 3 NT_PRPSINFO (old) */
1199     #endif
1200     #ifdef __sparc
1201     note_xreg, /* 4 NT_PRXREG */
1202     #else
1203     note_notsup, /* 4 NT_PRXREG */
1204     #endif
1205     note_platform, /* 5 NT_PLATFORM */

```

```

1206     note_auxv,          /* 6  NT_AUXV          */
1207 #ifdef __sparc
1208     note_gwindows,      /* 7  NT_GWINDOWS     */
1209 #ifdef __sparcv9
1210     note_asrs,          /* 8  NT_ASRS         */
1211 #else
1212     note_notsup,        /* 8  NT_ASRS         */
1213 #endif
1214 #else
1215     note_notsup,        /* 7  NT_GWINDOWS     */
1216     note_notsup,        /* 8  NT_ASRS         */
1217 #endif
1218 #ifdef __x86
1219     note_ldt,           /* 9  NT_LDT          */
1220 #else
1221     note_notsup,        /* 9  NT_LDT          */
1222 #endif
1223     note_pstatus,      /* 10 NT_PSTATUS      */
1224     note_notsup,        /* 11 unassigned      */
1225     note_notsup,        /* 12 unassigned      */
1226     note_psinfo,        /* 13 NT_PSINFO       */
1227     note_cred,          /* 14 NT_PRCRED       */
1228     note_utsname,       /* 15 NT_UTSNAME      */
1229     note_lwpstatus,     /* 16 NT_LWPSTATUS    */
1230     note_lwpsinfo,      /* 17 NT_LWPSINFO     */
1231     note_priv,          /* 18 NT_PRPRIV       */
1232     note_priv_info,     /* 19 NT_PRPRIVINFO   */
1233     note_content,       /* 20 NT_CONTENT       */
1234     note_zonename,      /* 21 NT_ZONENAME     */
1235     note_fdinfo,        /* 22 NT_FDINFO       */
1236     note_spymaster,     /* 23 NT_SPYMASTER    */
1237     note_secflags,      /* 24 NT_SECFLAGS     */
1238 #endif /* ! codereview */
1239 };

1241 static void
1242 core_report_mapping(struct ps_prochandle *P, GElf_Phdr *php)
1243 {
1244     prkillinfo_t killinfo;
1245     siginfo_t *si = &killinfo.prk_info;
1246     char signame[SIG2STR_MAX], sig[64], info[64];
1247     void *addr = (void *) (uintptr_t) php->p_vaddr;

1249     const char *errfmt = "core file data for mapping at %p not saved: %s\n";
1250     const char *incfmt = "core file incomplete due to %s%s\n";
1251     const char *msgfmt = "mappings at and above %p are missing\n";

1253     if (!(php->p_flags & PF_SUNW_KILLED)) {
1254         int err = 0;

1256         (void) pread64(P->asfd, &err,
1257             sizeof (err), (off64_t) php->p_offset);

1259         Perror_printf(P, errfmt, addr, strerror(err));
1260         dprintf(errfmt, addr, strerror(err));
1261         return;
1262     }

1264     if (!(php->p_flags & PF_SUNW_SIGINFO))
1265         return;

1267     (void) memset(&killinfo, 0, sizeof (killinfo));

1269     (void) pread64(P->asfd, &killinfo,
1270         sizeof (killinfo), (off64_t) php->p_offset);

```

```

1272     /*
1273     * While there is (or at least should be) only one segment that has
1274     * PF_SUNW_SIGINFO set, the signal information there is globally
1275     * useful (even if only to those debugging libproc consumers); we hang
1276     * the signal information gleaned here off of the ps_prochandle.
1277     */
1278     P->map_missing = php->p_vaddr;
1279     P->killinfo = killinfo.prk_info;

1281     if (sig2str(si->si_signo, signame) == -1) {
1282         (void) snprintf(sig, sizeof (sig),
1283             "<Unknown signal: 0x%x>", si->si_signo);
1284     } else {
1285         (void) snprintf(sig, sizeof (sig), "SIG%s", signame);
1286     }

1288     if (si->si_code == SI_USER || si->si_code == SI_QUEUE) {
1289         (void) snprintf(info, sizeof (info),
1290             "pid=%d uid=%d zone=%d ctid=%d",
1291             si->si_pid, si->si_uid, si->si_zoneid, si->si_ctid);
1292     } else {
1293         (void) snprintf(info, sizeof (info),
1294             "code=%d", si->si_code);
1295     }

1297     Perror_printf(P, incfmt, sig, info);
1298     Perror_printf(P, msgfmt, addr);

1300     dprintf(incfmt, sig, info);
1301     dprintf(msgfmt, addr);
1302 }

1304 /*
1305 * Add information on the address space mapping described by the given
1306 * PT_LOAD program header. We fill in more information on the mapping later.
1307 */
1308 static int
1309 core_add_mapping(struct ps_prochandle *P, GElf_Phdr *php)
1310 {
1311     core_info_t *core = P->data;
1312     pmap_t pmap;

1314     dprintf("mapping base %llx filesz %llx memsz %llx offset %llx\n",
1315         (u_longlong_t) php->p_vaddr, (u_longlong_t) php->p_filesz,
1316         (u_longlong_t) php->p_memsz, (u_longlong_t) php->p_offset);

1318     pmap.pr_vaddr = (uintptr_t) php->p_vaddr;
1319     pmap.pr_size = php->p_memsz;

1321     /*
1322     * If Pgcure() or elfcore() fail to write a mapping, they will set
1323     * PF_SUNW_FAILURE in the Phdr and try to stash away the errno for us.
1324     */
1325     if (php->p_flags & PF_SUNW_FAILURE) {
1326         core_report_mapping(P, php);
1327     } else if (php->p_filesz != 0 && php->p_offset >= core->core_size) {
1328         Perror_printf(P, "core file may be corrupt -- data for mapping "
1329             "at %p is missing\n", (void *) (uintptr_t) php->p_vaddr);
1330         dprintf("core file may be corrupt -- data for mapping "
1331             "at %p is missing\n", (void *) (uintptr_t) php->p_vaddr);
1332     }

1334     /*
1335     * The mapping name and offset will hopefully be filled in
1336     * by the librtld_db agent. Unfortunately, if it isn't a
1337     * shared library mapping, this information is gone forever.

```



```

1338     */
1339     pmap.pr_mapname[0] = '\0';
1340     pmap.pr_offset = 0;

1342     pmap.pr_mflags = 0;
1343     if (php->p_flags & PF_R)
1344         pmap.pr_mflags |= MA_READ;
1345     if (php->p_flags & PF_W)
1346         pmap.pr_mflags |= MA_WRITE;
1347     if (php->p_flags & PF_X)
1348         pmap.pr_mflags |= MA_EXEC;

1350     if (php->p_filesz == 0)
1351         pmap.pr_mflags |= MA_RESERVED1;

1353     /*
1354     * At the time of adding this mapping, we just zero the pagesize.
1355     * Once we've processed more of the core file, we'll have the
1356     * pagesize from the auxv's AT_PAGESZ element and we can fill this in.
1357     */
1358     pmap.pr_pagesize = 0;

1360     /*
1361     * Unfortunately whether or not the mapping was a System V
1362     * shared memory segment is lost. We use -1 to mark it as not shm.
1363     */
1364     pmap.pr_shmid = -1;

1366     return (Padd_mapping(P, php->p_offset, NULL, &pmap));
1367 }

1369 /*
1370 * Given a virtual address, name the mapping at that address using the
1371 * specified name, and return the map_info_t pointer.
1372 */
1373 static map_info_t *
1374 core_name_mapping(struct ps_prochandle *P, uintptr_t addr, const char *name)
1375 {
1376     map_info_t *mp = Paddr2mptr(P, addr);

1378     if (mp != NULL) {
1379         (void) strncpy(mp->map_pmap.pr_mapname, name, PRMAPSZ);
1380         mp->map_pmap.pr_mapname[PRMAPSZ - 1] = '\0';
1381     }

1383     return (mp);
1384 }

1386 /*
1387 * libproc uses libelf for all of its symbol table manipulation. This function
1388 * takes a symbol table and string table from a core file and places them
1389 * in a memory backed elf file.
1390 */
1391 static void
1392 fake_up_syntab(struct ps_prochandle *P, const elf_file_header_t *ehdr,
1393               GElf_Shdr *syntab, GElf_Shdr *strtab)
1394 {
1395     size_t size;
1396     off64_t off, base;
1397     map_info_t *mp;
1398     file_info_t *fip;
1399     Elf_Scn *scn;
1400     Elf_Data *data;

1402     if (syntab->sh_addr == 0 ||
1403         (mp = Paddr2mptr(P, syntab->sh_addr)) == NULL ||

```

```

1404         (fp = mp->map_file) == NULL) {
1405             dprintf("fake_up_syntab: invalid section\n");
1406             return;
1407         }

1409     if (fp->file_syntab.sym_data_pri != NULL) {
1410         dprintf("Symbol table already loaded (sh_addr 0x%lx)\n",
1411             (long)syntab->sh_addr);
1412         return;
1413     }

1415     if (P->status.pr_dmodel == PR_MODEL_ILP32) {
1416         struct
1417         {
1418             Elf32_Ehdr ehdr;
1419             Elf32_Shdr shdr[3];
1420             char data[1];
1421         } *b;

1422         base = sizeof (b->ehdr) + sizeof (b->shdr);
1423         size = base + syntab->sh_size + strtab->sh_size;

1425         if ((b = calloc(1, size)) == NULL)
1426             return;

1428         (void) memcpy(b->ehdr.e_ident, ehdr->e_ident,
1429             sizeof (ehdr->e_ident));
1430         b->ehdr.e_type = ehdr->e_type;
1431         b->ehdr.e_machine = ehdr->e_machine;
1432         b->ehdr.e_version = ehdr->e_version;
1433         b->ehdr.e_flags = ehdr->e_flags;
1434         b->ehdr.e_ehsize = sizeof (b->ehdr);
1435         b->ehdr.e_shoff = sizeof (b->ehdr);
1436         b->ehdr.e_shentsize = sizeof (b->shdr[0]);
1437         b->ehdr.e_shnum = 3;
1438         off = 0;

1440         b->shdr[1].sh_size = syntab->sh_size;
1441         b->shdr[1].sh_type = SHT_SYMTAB;
1442         b->shdr[1].sh_offset = off + base;
1443         b->shdr[1].sh_entsize = sizeof (Elf32_Sym);
1444         b->shdr[1].sh_link = 2;
1445         b->shdr[1].sh_info = syntab->sh_info;
1446         b->shdr[1].sh_addralign = syntab->sh_addralign;

1448         if (pread64(P->asfd, &b->data[off], b->shdr[1].sh_size,
1449             syntab->sh_offset) != b->shdr[1].sh_size) {
1450             dprintf("fake_up_syntab: pread of syntab[1] failed\n");
1451             free(b);
1452             return;
1453         }

1455         off += b->shdr[1].sh_size;

1457         b->shdr[2].sh_flags = SHF_STRINGS;
1458         b->shdr[2].sh_size = strtab->sh_size;
1459         b->shdr[2].sh_type = SHT_STRTAB;
1460         b->shdr[2].sh_offset = off + base;
1461         b->shdr[2].sh_info = strtab->sh_info;
1462         b->shdr[2].sh_addralign = 1;

1464         if (pread64(P->asfd, &b->data[off], b->shdr[2].sh_size,
1465             strtab->sh_offset) != b->shdr[2].sh_size) {
1466             dprintf("fake_up_syntab: pread of syntab[2] failed\n");
1467             free(b);
1468             return;
1469         }

```

```

1471         off += b->shdr[2].sh_size;
1473         fp->file_syntab.sym_elf = elf_memory((char *)b, size);
1474         if (fp->file_syntab.sym_elf == NULL) {
1475             free(b);
1476             return;
1477         }
1479         fp->file_syntab.sym_elfmem = b;
1480 #ifdef LP64
1481     } else {
1482         struct {
1483             Elf64_Ehdr ehdr;
1484             Elf64_Shdr shdr[3];
1485             char data[1];
1486         } *b;
1488         base = sizeof(b->ehdr) + sizeof(b->shdr);
1489         size = base + syntab->sh_size + strtb->sh_size;
1491         if ((b = calloc(1, size)) == NULL)
1492             return;
1494         (void) memcpy(b->ehdr.e_ident, ehdr->e_ident,
1495                     sizeof(ehdr->e_ident));
1496         b->ehdr.e_type = ehdr->e_type;
1497         b->ehdr.e_machine = ehdr->e_machine;
1498         b->ehdr.e_version = ehdr->e_version;
1499         b->ehdr.e_flags = ehdr->e_flags;
1500         b->ehdr.e_ehsize = sizeof(b->ehdr);
1501         b->ehdr.e_shoff = sizeof(b->ehdr);
1502         b->ehdr.e_shentsize = sizeof(b->shdr[0]);
1503         b->ehdr.e_shnum = 3;
1504         off = 0;
1506         b->shdr[1].sh_size = syntab->sh_size;
1507         b->shdr[1].sh_type = SHT_SYMTAB;
1508         b->shdr[1].sh_offset = off + base;
1509         b->shdr[1].sh_entsize = sizeof(Elf64_Sym);
1510         b->shdr[1].sh_link = 2;
1511         b->shdr[1].sh_info = syntab->sh_info;
1512         b->shdr[1].sh_addralign = syntab->sh_addralign;
1514         if (pread64(P->asfd, &b->data[off], b->shdr[1].sh_size,
1515                 syntab->sh_offset) != b->shdr[1].sh_size) {
1516             free(b);
1517             return;
1518         }
1520         off += b->shdr[1].sh_size;
1522         b->shdr[2].sh_flags = SHF_STRINGS;
1523         b->shdr[2].sh_size = strtb->sh_size;
1524         b->shdr[2].sh_type = SHT_STRTAB;
1525         b->shdr[2].sh_offset = off + base;
1526         b->shdr[2].sh_info = strtb->sh_info;
1527         b->shdr[2].sh_addralign = 1;
1529         if (pread64(P->asfd, &b->data[off], b->shdr[2].sh_size,
1530                 strtb->sh_offset) != b->shdr[2].sh_size) {
1531             free(b);
1532             return;
1533         }
1535         off += b->shdr[2].sh_size;

```

```

1537         fp->file_syntab.sym_elf = elf_memory((char *)b, size);
1538         if (fp->file_syntab.sym_elf == NULL) {
1539             free(b);
1540             return;
1541         }
1543         fp->file_syntab.sym_elfmem = b;
1544 #endif
1545     }
1547     if ((scn = elf_getscn(fp->file_syntab.sym_elf, 1)) == NULL ||
1548         (fp->file_syntab.sym_data_pri = elf_getdata(scn, NULL)) == NULL ||
1549         (scn = elf_getscn(fp->file_syntab.sym_elf, 2)) == NULL ||
1550         (data = elf_getdata(scn, NULL)) == NULL) {
1551         dprintf("fake_up_syntab: failed to get section data at %p\n",
1552             (void *)scn);
1553         goto err;
1554     }
1556     fp->file_syntab.sym_strs = data->d_buf;
1557     fp->file_syntab.sym_strsz = data->d_size;
1558     fp->file_syntab.sym_symn = syntab->sh_size / syntab->sh_entsize;
1559     fp->file_syntab.sym_hdr_pri = *syntab;
1560     fp->file_syntab.sym_strhdr = *strtab;
1562     optimize_syntab(&fp->file_syntab);
1564     return;
1565 err:
1566     (void) elf_end(fp->file_syntab.sym_elf);
1567     free(fp->file_syntab.sym_elfmem);
1568     fp->file_syntab.sym_elf = NULL;
1569     fp->file_syntab.sym_elfmem = NULL;
1570 }
1572 static void
1573 core_phdr_to_gelf(const Elf32_Phdr *src, GElf_Phdr *dst)
1574 {
1575     dst->p_type = src->p_type;
1576     dst->p_flags = src->p_flags;
1577     dst->p_offset = (Elf64_Off)src->p_offset;
1578     dst->p_vaddr = (Elf64_Addr)src->p_vaddr;
1579     dst->p_paddr = (Elf64_Addr)src->p_paddr;
1580     dst->p_filesz = (Elf64_Xword)src->p_filesz;
1581     dst->p_memsz = (Elf64_Xword)src->p_memsz;
1582     dst->p_align = (Elf64_Xword)src->p_align;
1583 }
1585 static void
1586 core_shdr_to_gelf(const Elf32_Shdr *src, GElf_Shdr *dst)
1587 {
1588     dst->sh_name = src->sh_name;
1589     dst->sh_type = src->sh_type;
1590     dst->sh_flags = (Elf64_Xword)src->sh_flags;
1591     dst->sh_addr = (Elf64_Addr)src->sh_addr;
1592     dst->sh_offset = (Elf64_Off)src->sh_offset;
1593     dst->sh_size = (Elf64_Xword)src->sh_size;
1594     dst->sh_link = src->sh_link;
1595     dst->sh_info = src->sh_info;
1596     dst->sh_addralign = (Elf64_Xword)src->sh_addralign;
1597     dst->sh_entsize = (Elf64_Xword)src->sh_entsize;
1598 }
1600 /*
1601  * Perform elf_begin on efp->e_fd and verify the ELF file's type and class.

```

```

1602 */
1603 static int
1604 core_elf_fdopen(elf_file_t *efp, GElf_Half type, int *perr)
1605 {
1606 #ifdef _BIG_ENDIAN
1607     uchar_t order = ELFDATA2MSB;
1608 #else
1609     uchar_t order = ELFDATA2LSB;
1610 #endif
1611     Elf32_Ehdr e32;
1612     int is_noelf = -1;
1613     int isa_err = 0;
1614
1615     /*
1616      * Because 32-bit libelf cannot deal with large files, we need to read,
1617      * check, and convert the file header manually in case type == ET_CORE.
1618      */
1619     if (pread64(efp->e_fd, &e32, sizeof (e32), 0) != sizeof (e32)) {
1620         if (perr != NULL)
1621             *perr = G_FORMAT;
1622         goto err;
1623     }
1624     if ((is_noelf == memcmp(&e32.e_ident[EI_MAG0], ELFMAG, SELFMAG)) != 0 ||
1625         e32.e_type != type || (isa_err = (e32.e_ident[EI_DATA] != order)) ||
1626         e32.e_version != EV_CURRENT) {
1627         if (perr != NULL) {
1628             if (is_noelf == 0 && isa_err) {
1629                 *perr = G_ISAINVAL;
1630             } else {
1631                 *perr = G_FORMAT;
1632             }
1633         }
1634         goto err;
1635     }
1636
1637     /*
1638      * If the file is 64-bit and we are 32-bit, fail with G_LP64. If the
1639      * file is 64-bit and we are 64-bit, re-read the header as a Elf64_Ehdr,
1640      * and convert it to a elf_file_header_t. Otherwise, the file is
1641      * 32-bit, so convert e32 to a elf_file_header_t.
1642      */
1643     if (e32.e_ident[EI_CLASS] == ELFCLASS64) {
1644 #ifdef _LP64
1645         Elf64_Ehdr e64;
1646
1647         if (pread64(efp->e_fd, &e64, sizeof (e64), 0) != sizeof (e64)) {
1648             if (perr != NULL)
1649                 *perr = G_FORMAT;
1650             goto err;
1651         }
1652
1653         (void) memcpy(efp->e_hdr.e_ident, e64.e_ident, EI_NIDENT);
1654         efp->e_hdr.e_type = e64.e_type;
1655         efp->e_hdr.e_machine = e64.e_machine;
1656         efp->e_hdr.e_version = e64.e_version;
1657         efp->e_hdr.e_entry = e64.e_entry;
1658         efp->e_hdr.e_phoff = e64.e_phoff;
1659         efp->e_hdr.e_shoff = e64.e_shoff;
1660         efp->e_hdr.e_flags = e64.e_flags;
1661         efp->e_hdr.e_ehsize = e64.e_ehsize;
1662         efp->e_hdr.e_phentsize = e64.e_phentsize;
1663         efp->e_hdr.e_phnum = (Elf64_Word)e64.e_phnum;
1664         efp->e_hdr.e_shentsize = e64.e_shentsize;
1665         efp->e_hdr.e_shnum = (Elf64_Word)e64.e_shnum;
1666         efp->e_hdr.e_shstrndx = (Elf64_Word)e64.e_shstrndx;
1667 #else /* _LP64 */

```

```

1668         if (perr != NULL)
1669             *perr = G_LP64;
1670         goto err;
1671 #endif /* _LP64 */
1672     } else {
1673         (void) memcpy(efp->e_hdr.e_ident, e32.e_ident, EI_NIDENT);
1674         efp->e_hdr.e_type = e32.e_type;
1675         efp->e_hdr.e_machine = e32.e_machine;
1676         efp->e_hdr.e_version = e32.e_version;
1677         efp->e_hdr.e_entry = (Elf64_Addr)e32.e_entry;
1678         efp->e_hdr.e_phoff = (Elf64_Off)e32.e_phoff;
1679         efp->e_hdr.e_shoff = (Elf64_Off)e32.e_shoff;
1680         efp->e_hdr.e_flags = e32.e_flags;
1681         efp->e_hdr.e_ehsize = e32.e_ehsize;
1682         efp->e_hdr.e_phentsize = e32.e_phentsize;
1683         efp->e_hdr.e_phnum = (Elf64_Word)e32.e_phnum;
1684         efp->e_hdr.e_shentsize = e32.e_shentsize;
1685         efp->e_hdr.e_shnum = (Elf64_Word)e32.e_shnum;
1686         efp->e_hdr.e_shstrndx = (Elf64_Word)e32.e_shstrndx;
1687     }
1688
1689     /*
1690      * If the number of section headers or program headers or the section
1691      * header string table index would overflow their respective fields
1692      * in the ELF header, they're stored in the section header at index
1693      * zero. To simplify use elsewhere, we look for those sentinel values
1694      * here.
1695      */
1696     if ((efp->e_hdr.e_shnum == 0 && efp->e_hdr.e_shoff != 0) ||
1697         efp->e_hdr.e_shstrndx == SHN_XINDEX ||
1698         efp->e_hdr.e_phnum == PN_XNUM) {
1699         GElf_Shdr shdr;
1700
1701         dprintf("extended ELF header\n");
1702
1703         if (efp->e_hdr.e_shoff == 0) {
1704             if (perr != NULL)
1705                 *perr = G_FORMAT;
1706             goto err;
1707         }
1708
1709         if (efp->e_hdr.e_ident[EI_CLASS] == ELFCLASS32) {
1710             Elf32_Shdr shdr32;
1711
1712             if (pread64(efp->e_fd, &shdr32, sizeof (shdr32),
1713                 efp->e_hdr.e_shoff) != sizeof (shdr32)) {
1714                 if (perr != NULL)
1715                     *perr = G_FORMAT;
1716                 goto err;
1717             }
1718
1719             core_shdr_to_gelf(&shdr32, &shdr);
1720         } else {
1721             if (pread64(efp->e_fd, &shdr, sizeof (shdr),
1722                 efp->e_hdr.e_shoff) != sizeof (shdr)) {
1723                 if (perr != NULL)
1724                     *perr = G_FORMAT;
1725                 goto err;
1726             }
1727         }
1728
1729         if (efp->e_hdr.e_shnum == 0) {
1730             efp->e_hdr.e_shnum = shdr.sh_size;
1731             dprintf("section header count %lu\n",
1732                 (ulong_t)shdr.sh_size);
1733         }

```

```

1735     if (efp->e_hdr.e_shstrndx == SHN_XINDEX) {
1736         efp->e_hdr.e_shstrndx = shdr.sh_link;
1737         dprintf("section string index %u\n", shdr.sh_link);
1738     }

1740     if (efp->e_hdr.e_phnum == PN_XNUM && shdr.sh_info != 0) {
1741         efp->e_hdr.e_phnum = shdr.sh_info;
1742         dprintf("program header count %u\n", shdr.sh_info);
1743     }

1745 } else if (efp->e_hdr.e_phoff != 0) {
1746     GElf_Phdr phdr;
1747     uint64_t phnum;

1749     /*
1750     * It's possible this core file came from a system that
1751     * accidentally truncated the e_phnum field without correctly
1752     * using the extended format in the section header at index
1753     * zero. We try to detect and correct that specific type of
1754     * corruption by using the knowledge that the core dump
1755     * routines usually place the data referenced by the first
1756     * program header immediately after the last header element.
1757     */
1758     if (efp->e_hdr.e_ident[EI_CLASS] == ELFCLASS32) {
1759         Elf32_Phdr phdr32;

1761         if (pread64(efp->e_fd, &phdr32, sizeof(phdr32),
1762             efp->e_hdr.e_phoff) != sizeof(phdr32)) {
1763             if (perr != NULL)
1764                 *perr = G_FORMAT;
1765             goto err;
1766         }

1768         core_phdr_to_gelf(&phdr32, &phdr);
1769     } else {
1770         if (pread64(efp->e_fd, &phdr, sizeof(phdr),
1771             efp->e_hdr.e_phoff) != sizeof(phdr)) {
1772             if (perr != NULL)
1773                 *perr = G_FORMAT;
1774             goto err;
1775         }
1776     }

1778     phnum = phdr.p_offset - efp->e_hdr.e_ehsize -
1779         (uint64_t)efp->e_hdr.e_shnum * efp->e_hdr.e_shentsize;
1780     phnum /= efp->e_hdr.e_phentsize;

1782     if (phdr.p_offset != 0 && phnum != efp->e_hdr.e_phnum) {
1783         dprintf("suspicious program header count %u %u\n",
1784             (uint_t)phnum, efp->e_hdr.e_phnum);

1786         /*
1787         * If the new program header count we computed doesn't
1788         * jive with count in the ELF header, we'll use the
1789         * data that's there and hope for the best.
1790         *
1791         * If it does, it's also possible that the section
1792         * header offset is incorrect; we'll check that and
1793         * possibly try to fix it.
1794         */
1795         if (phnum <= INT_MAX &&
1796             (uint16_t)phnum == efp->e_hdr.e_phnum) {

1798             if (efp->e_hdr.e_shoff == efp->e_hdr.e_phoff +
1799                 efp->e_hdr.e_phentsize *

```

```

1800         (uint_t)efp->e_hdr.e_phnum) {
1801             efp->e_hdr.e_shoff =
1802                 efp->e_hdr.e_phoff +
1803                 efp->e_hdr.e_phentsize * phnum;
1804         }

1806         efp->e_hdr.e_phnum = (Elf64_Word)phnum;
1807         dprintf("using new program header count\n");
1808     } else {
1809         dprintf("inconsistent program header count\n");
1810     }
1811 }
1812 }

1814 /*
1815 * The libelf implementation was never ported to be large-file aware.
1816 * This is typically not a problem for your average executable or
1817 * shared library, but a large 32-bit core file can exceed 2GB in size.
1818 * So if type is ET_CORE, we don't bother doing elf_begin; the code
1819 * in Pfgrab_core() below will do its own i/o and struct conversion.
1820 */

1822 if (type == ET_CORE) {
1823     efp->e_elf = NULL;
1824     return (0);
1825 }

1827 if ((efp->e_elf = elf_begin(efp->e_fd, ELF_C_READ, NULL)) == NULL) {
1828     if (perr != NULL)
1829         *perr = G_ELF;
1830     goto err;
1831 }

1833 return (0);

1835 err:
1836     efp->e_elf = NULL;
1837     return (-1);
1838 }

1840 /*
1841 * Open the specified file and then do a core_elf_fdopen on it.
1842 */
1843 static int
1844 core_elf_open(elf_file_t *efp, const char *path, GElf_Half type, int *perr)
1845 {
1846     (void) memset(efp, 0, sizeof(elf_file_t));

1848     if ((efp->e_fd = open64(path, O_RDONLY)) >= 0) {
1849         if (core_elf_fdopen(efp, type, perr) == 0)
1850             return (0);

1852         (void) close(efp->e_fd);
1853         efp->e_fd = -1;
1854     }

1856     return (-1);
1857 }

1859 /*
1860 * Close the ELF handle and file descriptor.
1861 */
1862 static void
1863 core_elf_close(elf_file_t *efp)
1864 {
1865     if (efp->e_elf != NULL) {

```

```

1866         (void) elf_end(efp->e_elf);
1867         efp->e_elf = NULL;
1868     }
1870     if (efp->e_fd != -1) {
1871         (void) close(efp->e_fd);
1872         efp->e_fd = -1;
1873     }
1874 }
1876 /*
1877  * Given an ELF file for a statically linked executable, locate the likely
1878  * primary text section and fill in rl_base with its virtual address.
1879  */
1880 static map_info_t *
1881 core_find_text(struct ps_prochandle *P, Elf *elf, rd_loadobj_t *rlp)
1882 {
1883     GElf_Phdr phdr;
1884     uint_t i;
1885     size_t nphdrs;
1887     if (elf_getphdrnum(elf, &nphdrs) == -1)
1888         return (NULL);
1890     for (i = 0; i < nphdrs; i++) {
1891         if (gelf_getphdr(elf, i, &phdr) != NULL &&
1892             phdr.p_type == PT_LOAD && (phdr.p_flags & PF_X)) {
1893             rlp->rl_base = phdr.p_vaddr;
1894             return (Paddr2mptr(P, rlp->rl_base));
1895         }
1896     }
1898     return (NULL);
1899 }
1901 /*
1902  * Given an ELF file and the librtld_db structure corresponding to its primary
1903  * text mapping, deduce where its data segment was loaded and fill in
1904  * rl_data_base and prmap_t.pr_offset accordingly.
1905  */
1906 static map_info_t *
1907 core_find_data(struct ps_prochandle *P, Elf *elf, rd_loadobj_t *rlp)
1908 {
1909     GElf_Ehdr ehdr;
1910     GElf_Phdr phdr;
1911     map_info_t *mp;
1912     uint_t i, pagemask;
1913     size_t nphdrs;
1915     rlp->rl_data_base = NULL;
1917     /*
1918     * Find the first loadable, writeable Phdr and compute rl_data_base
1919     * as the virtual address at which it was loaded.
1920     */
1921     if (gelf_getehdr(elf, &ehdr) == NULL ||
1922         elf_getphdrnum(elf, &nphdrs) == -1)
1923         return (NULL);
1925     for (i = 0; i < nphdrs; i++) {
1926         if (gelf_getphdr(elf, i, &phdr) != NULL &&
1927             phdr.p_type == PT_LOAD && (phdr.p_flags & PF_W)) {
1928             rlp->rl_data_base = phdr.p_vaddr;
1929             if (ehdr.e_type == ET_DYN)
1930                 rlp->rl_data_base += rlp->rl_base;
1931             break;

```

```

1932     }
1933 }
1935 /*
1936  * If we didn't find an appropriate phdr or if the address we
1937  * computed has no mapping, return NULL.
1938  */
1939 if (rlp->rl_data_base == NULL ||
1940     (mp = Paddr2mptr(P, rlp->rl_data_base)) == NULL)
1941     return (NULL);
1943 /*
1944  * It wouldn't be procs-related code if we didn't make use of
1945  * unclean knowledge of segvn, even in userland ... the prmap_t's
1946  * pr_offset field will be the segvn offset from mmap(2)ing the
1947  * data section, which will be the file offset & PAGEMASK.
1948  */
1949 pagemask = ~(mp->map_pmap.pr_pagesize - 1);
1950 mp->map_pmap.pr_offset = phdr.p_offset & pagemask;
1952     return (mp);
1953 }
1955 /*
1956  * Librtld_db agent callback for iterating over load object mappings.
1957  * For each load object, we allocate a new file_info_t, perform naming,
1958  * and attempt to construct a symbol table for the load object.
1959  */
1960 static int
1961 core_iter_mapping(const rd_loadobj_t *rlp, struct ps_prochandle *P)
1962 {
1963     core_info_t *core = P->data;
1964     char lname[PATH_MAX], buf[PATH_MAX];
1965     file_info_t *fp;
1966     map_info_t *mp;
1968     if (Pread_string(P, lname, PATH_MAX, (off_t)rlp->rl_nameaddr) <= 0) {
1969         dprintf("failed to read name %p\n", (void *)rlp->rl_nameaddr);
1970         return (1); /* Keep going; forget this if we can't get a name */
1971     }
1973     dprintf("rd_loadobj name = \"%s\" rl_base = %p\n",
1974           lname, (void *)rlp->rl_base);
1976     if ((mp = Paddr2mptr(P, rlp->rl_base)) == NULL) {
1977         dprintf("no mapping for %p\n", (void *)rlp->rl_base);
1978         return (1); /* No mapping; advance to next mapping */
1979     }
1981     /*
1982     * Create a new file_info_t for this mapping, and therefore for
1983     * this load object.
1984     *
1985     * If there's an ELF header at the beginning of this mapping,
1986     * file_info_new() will try to use its section headers to
1987     * identify any other mappings that belong to this load object.
1988     */
1989     if ((fp = mp->map_file) == NULL &&
1990         (fp = file_info_new(P, mp)) == NULL) {
1991         core->core_errno = errno;
1992         dprintf("failed to malloc mapping data\n");
1993         return (0); /* Abort */
1994     }
1995     fp->file_map = mp;
1997     /* Create a local copy of the load object representation */

```

```

1998     if ((fp->file_lo = calloc(1, sizeof (rd_loadobj_t))) == NULL) {
1999         core->core_errno = errno;
2000         dprintf("failed to malloc mapping data\n");
2001         return (0); /* Abort */
2002     }
2003     *fp->file_lo = *rlp;
2004
2005     if (lname[0] != '\0') {
2006         /*
2007          * Naming dance part 1: if we got a name from librtld_db, then
2008          * copy this name to the pmap_t if it is unnamed. If the
2009          * file_info_t is unnamed, name it after the lname.
2010          */
2011         if (mp->map_pmap.pr_mapname[0] == '\0') {
2012             (void) strncpy(mp->map_pmap.pr_mapname, lname, PRMAPSZ);
2013             mp->map_pmap.pr_mapname[PRMAPSZ - 1] = '\0';
2014         }
2015
2016         if (fp->file_lname == NULL)
2017             fp->file_lname = strdup(lname);
2018
2019     } else if (fp->file_lname == NULL &&
2020             mp->map_pmap.pr_mapname[0] != '\0') {
2021         /*
2022          * Naming dance part 2: if the mapping is named and the
2023          * file_info_t is not, name the file after the mapping.
2024          */
2025         fp->file_lname = strdup(mp->map_pmap.pr_mapname);
2026     }
2027
2028     if ((fp->file_rname == NULL) &&
2029         (Pfindmap(P, mp, buf, sizeof (buf)) != NULL))
2030         fp->file_rname = strdup(buf);
2031
2032     if (fp->file_lname != NULL)
2033         fp->file_lbase = basename(fp->file_lname);
2034     if (fp->file_rname != NULL)
2035         fp->file_rbase = basename(fp->file_rname);
2036
2037     /* Associate the file and the mapping. */
2038     (void) strncpy(fp->file_pname, mp->map_pmap.pr_mapname, PRMAPSZ);
2039     fp->file_pname[PRMAPSZ - 1] = '\0';
2040
2041     /*
2042      * If no section headers were available then we'll have to
2043      * identify this load object's other mappings with what we've
2044      * got: the start and end of the object's corresponding
2045      * address space.
2046      */
2047     if (fp->file_saddr == NULL) {
2048         for (mp = fp->file_map + 1; mp < P->mappings + P->map_count &&
2049             mp->map_pmap.pr_vaddr < rlp->rl_bend; mp++) {
2050
2051             if (mp->map_file == NULL) {
2052                 dprintf("core_iter_mapping %s: associating "
2053                     "segment at %p\n",
2054                     fp->file_pname,
2055                     (void *)mp->map_pmap.pr_vaddr);
2056                 mp->map_file = fp;
2057                 fp->file_ref++;
2058             } else {
2059                 dprintf("core_iter_mapping %s: segment at "
2060                     "%p already associated with %s\n",
2061                     fp->file_pname,
2062                     (void *)mp->map_pmap.pr_vaddr,
2063                     (mp == fp->file_map ? "this file" :

```

```

2064             mp->map_file->file_pname));
2065         }
2066     }
2067 }
2068
2069 /* Ensure that all this file's mappings are named. */
2070 for (mp = fp->file_map; mp < P->mappings + P->map_count &&
2071     mp->map_file == fp; mp++) {
2072     if (mp->map_pmap.pr_mapname[0] == '\0' &&
2073         !(mp->map_pmap.pr_mflags & MA_BREAK)) {
2074         (void) strncpy(mp->map_pmap.pr_mapname, fp->file_pname,
2075             PRMAPSZ);
2076         mp->map_pmap.pr_mapname[PRMAPSZ - 1] = '\0';
2077     }
2078 }
2079
2080 /* Attempt to build a symbol table for this file. */
2081 Pbuild_file_syntab(P, fp);
2082 if (fp->file_elf == NULL)
2083     dprintf("core_iter_mapping: no syntab for %s\n",
2084         fp->file_pname);
2085
2086 /* Locate the start of a data segment associated with this file. */
2087 if ((mp = core_find_data(P, fp->file_elf, fp->file_lo)) != NULL) {
2088     dprintf("found data for %s at %p (pr_offset 0x%llx)\n",
2089         fp->file_pname, (void *)fp->file_lo->rl_data_base,
2090         mp->map_pmap.pr_offset);
2091 } else {
2092     dprintf("core_iter_mapping: no data found for %s\n",
2093         fp->file_pname);
2094 }
2095
2096 return (1); /* Advance to next mapping */
2097 }
2098
2099 /*
2100  * Callback function for Pfindexec(). In order to confirm a given pathname,
2101  * we verify that we can open it as an ELF file of type ET_EXEC or ET_DYN.
2102  */
2103 static int
2104 core_exec_open(const char *path, void *efp)
2105 {
2106     if (core_elf_open(efp, path, ET_EXEC, NULL) == 0)
2107         return (1);
2108     if (core_elf_open(efp, path, ET_DYN, NULL) == 0)
2109         return (1);
2110     return (0);
2111 }
2112
2113 /*
2114  * Attempt to load any section headers found in the core file. If present,
2115  * this will refer to non-loadable data added to the core file by the kernel
2116  * based on coreadm(1M) settings, including CTF data and the symbol table.
2117  */
2118 static void
2119 core_load_shdrs(struct ps_prochandle *P, elf_file_t *efp)
2120 {
2121     GElf_Shdr *shp, *shdrs = NULL;
2122     char *shstrtab = NULL;
2123     ulong_t shstrtabsz;
2124     const char *name;
2125     map_info_t *mp;
2126
2127     size_t nbytes;
2128     void *buf;
2129     int i;

```

```

2131     if (efp->e_hdr.e_shstrndx >= efp->e_hdr.e_shnum) {
2132         dprintf("corrupt shstrndx (%u) exceeds shnum (%u)\n",
2133             efp->e_hdr.e_shstrndx, efp->e_hdr.e_shnum);
2134         return;
2135     }
2137     /*
2138     * Read the section header table from the core file and then iterate
2139     * over the section headers, converting each to a GElf_Shdr.
2140     */
2141     if ((shdrs = malloc(efp->e_hdr.e_shnum * sizeof(GElf_Shdr))) == NULL) {
2142         dprintf("failed to malloc %u section headers: %s\n",
2143             (uint_t)efp->e_hdr.e_shnum, strerror(errno));
2144         return;
2145     }
2147     nbytes = efp->e_hdr.e_shnum * efp->e_hdr.e_shentsize;
2148     if ((buf = malloc(nbytes)) == NULL) {
2149         dprintf("failed to malloc %d bytes: %s\n", (int)nbytes,
2150             strerror(errno));
2151         free(shdrs);
2152         goto out;
2153     }
2155     if (pread64(efp->e_fd, buf, nbytes, efp->e_hdr.e_shoff) != nbytes) {
2156         dprintf("failed to read section headers at off %lld: %s\n",
2157             (longlong_t)efp->e_hdr.e_shoff, strerror(errno));
2158         free(buf);
2159         goto out;
2160     }
2162     for (i = 0; i < efp->e_hdr.e_shnum; i++) {
2163         void *p = (uchar_t *)buf + efp->e_hdr.e_shentsize * i;
2165         if (efp->e_hdr.e_ident[EI_CLASS] == ELFCLASS32)
2166             core_shdr_to_gelf(p, &shdrs[i]);
2167         else
2168             (void) memcpy(&shdrs[i], p, sizeof(GElf_Shdr));
2169     }
2171     free(buf);
2172     buf = NULL;
2174     /*
2175     * Read the .shstrtab section from the core file, terminating it with
2176     * an extra \0 so that a corrupt section will not cause us to die.
2177     */
2178     shp = &shdrs[efp->e_hdr.e_shstrndx];
2179     shstrtab = shp->sh_size;
2181     if ((shstrtab = malloc(shstrtab + 1)) == NULL) {
2182         dprintf("failed to allocate %lu bytes for shstrtab\n",
2183             (ulong_t)shstrtab);
2184         goto out;
2185     }
2187     if (pread64(efp->e_fd, shstrtab, shstrtab,
2188         shp->sh_offset) != shstrtab) {
2189         dprintf("failed to read %lu bytes of shstrs at off %lld: %s\n",
2190             shstrtab, (longlong_t)shp->sh_offset, strerror(errno));
2191         goto out;
2192     }
2194     shstrtab[shstrtab] = '\0';

```

```

2196     /*
2197     * Now iterate over each section in the section header table, locating
2198     * sections of interest and initializing more of the ps_prochandle.
2199     */
2200     for (i = 0; i < efp->e_hdr.e_shnum; i++) {
2201         shp = &shdrs[i];
2202         name = shstrtab + shp->sh_name;
2204         if (shp->sh_name >= shstrtab + shstrtab) {
2205             dprintf("skipping section [%d]: corrupt sh_name\n", i);
2206             continue;
2207         }
2209         if (shp->sh_link >= efp->e_hdr.e_shnum) {
2210             dprintf("skipping section [%d]: corrupt sh_link\n", i);
2211             continue;
2212         }
2214         dprintf("found section header %s (sh_addr 0x%llx)\n",
2215             name, (u_longlong_t)shp->sh_addr);
2217         if (strcmp(name, ".SUNW_ctf") == 0) {
2218             if ((mp = Paddr2mptr(P, shp->sh_addr)) == NULL) {
2219                 dprintf("no map at addr 0x%llx for %s [%d]\n",
2220                     (u_longlong_t)shp->sh_addr, name, i);
2221                 continue;
2222             }
2224             if (mp->map_file == NULL ||
2225                 mp->map_file->file_ctf_buf != NULL) {
2226                 dprintf("no mapping file or duplicate buffer "
2227                     "for %s [%d]\n", name, i);
2228                 continue;
2229             }
2231             if ((buf = malloc(shp->sh_size)) == NULL ||
2232                 pread64(efp->e_fd, buf, shp->sh_size,
2233                     shp->sh_offset) != shp->sh_size) {
2234                 dprintf("skipping section %s [%d]: %s\n",
2235                     name, i, strerror(errno));
2236                 free(buf);
2237                 continue;
2238             }
2240             mp->map_file->file_ctf_size = shp->sh_size;
2241             mp->map_file->file_ctf_buf = buf;
2243             if (shdrs[shp->sh_link].sh_type == SHT_DYNSYM)
2244                 mp->map_file->file_ctf_dyn = 1;
2246             } else if (strcmp(name, ".symtab") == 0) {
2247                 fake_up_symtab(P, &efp->e_hdr,
2248                     shp, &shdrs[shp->sh_link]);
2249             }
2250         }
2251     out:
2252         free(shstrtab);
2253         free(shdrs);
2254     }
2256     /*
2257     * Main engine for core file initialization: given an fd for the core file
2258     * and an optional pathname, construct the ps_prochandle. The aout_path can
2259     * either be a suggested executable pathname, or a suggested directory to
2260     * use as a possible current working directory.
2261     */

```

```

2262 struct ps_prochandle *
2263 Pfgrab_core(int core_fd, const char *aout_path, int *perr)
2264 {
2265     struct ps_prochandle *P;
2266     core_info_t *core_info;
2267     map_info_t *stk_mp, *brk_mp;
2268     const char *execname;
2269     char *interp;
2270     int i, notes, pagesize;
2271     uintptr_t addr, base_addr;
2272     struct stat64 stbuf;
2273     void *phbuf, *php;
2274     size_t nbytes;
2275 #ifdef _x86
2276     boolean_t from_linux = B_FALSE;
2277 #endif
2279     elf_file_t aout;
2280     elf_file_t core;
2282     Elf_Scn *scn, *intp_scn = NULL;
2283     Elf_Data *dp;
2285     GElf_Phdr phdr, note_phdr;
2286     GElf_Shdr shdr;
2287     GElf_Xword nleft;
2289     if (elf_version(EV_CURRENT) == EV_NONE) {
2290         dprintf("libproc ELF version is more recent than libelf\n");
2291         *perr = G_ELF;
2292         return (NULL);
2293     }
2295     aout.e_elf = NULL;
2296     aout.e_fd = -1;
2298     core.e_elf = NULL;
2299     core.e_fd = core_fd;
2301     /*
2302      * Allocate and initialize a ps_prochandle structure for the core.
2303      * There are several key pieces of initialization here:
2304      *
2305      * 1. The PS_DEAD state flag marks this prochandle as a core file.
2306      *    PS_DEAD also thus prevents all operations which require state
2307      *    to be PS_STOP from operating on this handle.
2308      *
2309      * 2. We keep the core file fd in P->asfd since the core file contains
2310      *    the remnants of the process address space.
2311      *
2312      * 3. We set the P->info_valid bit because all information about the
2313      *    core is determined by the end of this function; there is no need
2314      *    for proc_update_maps() to reload mappings at any later point.
2315      *
2316      * 4. The read/write ops vector uses our core_rw() function defined
2317      *    above to handle i/o requests.
2318      */
2319     if ((P = malloc(sizeof (struct ps_prochandle))) == NULL) {
2320         *perr = G_STRANGE;
2321         return (NULL);
2322     }
2324     (void) memset(P, 0, sizeof (struct ps_prochandle));
2325     (void) mutex_init(&P->proc_lock, USYNC_THREAD, NULL);
2326     P->state = PS_DEAD;
2327     P->pid = (pid_t)-1;

```

```

2328     P->asfd = core.e_fd;
2329     P->ctlfd = -1;
2330     P->statfd = -1;
2331     P->agentctlfd = -1;
2332     P->agentstatfd = -1;
2333     P->zoneroot = NULL;
2334     P->info_valid = 1;
2335     Pinit_ops(&P->ops, &P_core_ops);
2337     Pinit_sym(P);
2339     /*
2340      * Fstat and open the core file and make sure it is a valid ELF core.
2341      */
2342     if (fstat64(P->asfd, &stbuf) == -1) {
2343         *perr = G_STRANGE;
2344         goto err;
2345     }
2347     if (core_elf_fdopen(&core, ET_CORE, perr) == -1)
2348         goto err;
2350     /*
2351      * Allocate and initialize a core_info_t to hang off the ps_prochandle
2352      * structure. We keep all core-specific information in this structure.
2353      */
2354     if ((core_info = calloc(1, sizeof (core_info_t))) == NULL) {
2355         *perr = G_STRANGE;
2356         goto err;
2357     }
2359     P->data = core_info;
2360     list_link(&core_info->core_lwp_head, NULL);
2361     core_info->core_size = stbuf.st_size;
2362     /*
2363      * In the days before adjustable core file content, this was the
2364      * default core file content. For new core files, this value will
2365      * be overwritten by the NT_CONTENT note section.
2366      */
2367     core_info->core_content = CC_CONTENT_STACK | CC_CONTENT_HEAP |
2368         CC_CONTENT_DATA | CC_CONTENT_RODATA | CC_CONTENT_ANON |
2369         CC_CONTENT_SHANON;
2371     switch (core.e_hdr.e_ident[EI_CLASS]) {
2372     case ELFCLASS32:
2373         core_info->core_dmodel = PR_MODEL_ILP32;
2374         break;
2375     case ELFCLASS64:
2376         core_info->core_dmodel = PR_MODEL_LP64;
2377         break;
2378     default:
2379         *perr = G_FORMAT;
2380         goto err;
2381     }
2382     core_info->core_osabi = core.e_hdr.e_ident[EI_OSABI];
2384     /*
2385      * Because the core file may be a large file, we can't use libelf to
2386      * read the Phdrs. We use e_phnum and e_phentsize to simplify things.
2387      */
2388     nbytes = core.e_hdr.e_phnum * core.e_hdr.e_phentsize;
2390     if ((phbuf = malloc(nbytes)) == NULL) {
2391         *perr = G_STRANGE;
2392         goto err;
2393     }

```



```

2395     if (pread64(core_fd, phbuf, nbytes, core.e_hdr.e_phoff) != nbytes) {
2396         *perr = G_STRANGE;
2397         free(phbuf);
2398         goto err;
2399     }

2401     /*
2402     * Iterate through the program headers in the core file.
2403     * We're interested in two types of Phdrs: PT_NOTE (which
2404     * contains a set of saved /proc structures), and PT_LOAD (which
2405     * represents a memory mapping from the process's address space).
2406     * In the case of PT_NOTE, we're interested in the last PT_NOTE
2407     * in the core file; currently the first PT_NOTE (if present)
2408     * contains /proc structs in the pre-2.6 unstructured /proc format.
2409     */
2410     for (php = phbuf, notes = 0, i = 0; i < core.e_hdr.e_phnum; i++) {
2411         if (core.e_hdr.e_ident[EI_CLASS] == ELFCLASS64)
2412             (void) memcpy(&phdr, php, sizeof(GElf_Phdr));
2413         else
2414             core_phdr_to_gelf(php, &phdr);

2416         switch (phdr.p_type) {
2417         case PT_NOTE:
2418             note_phdr = phdr;
2419             notes++;
2420             break;

2422         case PT_LOAD:
2423             if (core_add_mapping(P, &phdr) == -1) {
2424                 *perr = G_STRANGE;
2425                 free(phbuf);
2426                 goto err;
2427             }
2428             break;
2429         default:
2430             dprintf("Pgrab_core: unknown phdr %d\n", phdr.p_type);
2431             break;
2432         }

2434         php = (char *)php + core.e_hdr.e_phentsize;
2435     }

2437     free(phbuf);

2439     Psort_mappings(P);

2441     /*
2442     * If we couldn't find anything of type PT_NOTE, or only one PT_NOTE
2443     * was present, abort. The core file is either corrupt or too old.
2444     */
2445     if (notes == 0 || (notes == 1 && core_info->core_osabi ==
2446         ELFOSABI_SOLARIS)) {
2447         *perr = G_NOTE;
2448         goto err;
2449     }

2451     /*
2452     * Advance the seek pointer to the start of the PT_NOTE data
2453     */
2454     if (lseek64(P->asfd, note_phdr.p_offset, SEEK_SET) == (off64_t)-1) {
2455         dprintf("Pgrab_core: failed to lseek to PT_NOTE data\n");
2456         *perr = G_STRANGE;
2457         goto err;
2458     }

```

```

2460     /*
2461     * Now process the PT_NOTE structures. Each one is preceded by
2462     * an Elf{32/64}_Nhdr structure describing its type and size.
2463     *
2464     * +-----+
2465     * | header |
2466     * +-----+
2467     * | name   |
2468     * | ...    |
2469     * +-----+
2470     * | desc   |
2471     * | ...    |
2472     * +-----+
2473     */
2474     for (nleft = note_phdr.p_filesz; nleft > 0; ) {
2475         Elf64_Nhdr nhdr;
2476         off64_t off, namesz, descsz;

2478         /*
2479         * Although <sys/elf.h> defines both Elf32_Nhdr and Elf64_Nhdr
2480         * as different types, they are both of the same content and
2481         * size, so we don't need to worry about 32/64 conversion here.
2482         */
2483         if (read(P->asfd, &nhdr, sizeof(nhdr)) != sizeof(nhdr)) {
2484             dprintf("Pgrab_core: failed to read ELF note header\n");
2485             *perr = G_NOTE;
2486             goto err;
2487         }

2489         /*
2490         * According to the System V ABI, the amount of padding
2491         * following the name field should align the description
2492         * field on a 4 byte boundary for 32-bit binaries or on an 8
2493         * byte boundary for 64-bit binaries. However, this change
2494         * was not made correctly during the 64-bit port so all
2495         * descriptions can assume only 4-byte alignment. We ignore
2496         * the name field and the padding to 4-byte alignment.
2497         */
2498         namesz = P2ROUNDUP((off64_t)nhdr.n_namesz, (off64_t)4);

2500         if (lseek64(P->asfd, namesz, SEEK_CUR) == (off64_t)-1) {
2501             dprintf("failed to seek past name and padding\n");
2502             *perr = G_STRANGE;
2503             goto err;
2504         }

2506         dprintf("Note hdr n_type=%u n_namesz=%u n_descsz=%u\n",
2507             nhdr.n_type, nhdr.n_namesz, nhdr.n_descsz);

2509         off = lseek64(P->asfd, (off64_t)0L, SEEK_CUR);

2511         /*
2512         * Invoke the note handler function from our table
2513         */
2514         if (nhdr.n_type < sizeof(nhdlrs) / sizeof(nhdlrs[0])) {
2515             if (nhdlrs[nhdr.n_type](P, nhdr.n_descsz) < 0) {
2516                 dprintf("handler for type %d returned < 0",
2517                     nhdr.n_type);
2518                 *perr = G_NOTE;
2519                 goto err;
2520             }
2521         }
2522         /*
2523         * The presence of either of these notes indicates that
2524         * the dump was generated on Linux.
2525         */
2525     #ifdef __x86

```

```

2526         if (nhdr.n_type == NT_PRSTATUS ||
2527             nhdr.n_type == NT_PRPSINFO)
2528             from_linux = B_TRUE;
2529 #endif
2530     } else {
2531         (void) note_notsup(P, nhdr.n_descsz);
2532     }
2533
2534     /*
2535      * Seek past the current note data to the next Elf_Nhdr
2536      */
2537     descsize = P2ROUNDUP((off64_t)nhdr.n_descsz, (off64_t)4);
2538     if (lseek64(P->asfd, off + descsize, SEEK_SET) == (off64_t)-1) {
2539         dprintf("Pgrab_core: failed to seek to next nhdr\n");
2540         *perr = G_STRANGE;
2541         goto err;
2542     }
2543
2544     /*
2545      * Subtract the size of the header and its data from what
2546      * we have left to process.
2547      */
2548     nleft -= sizeof (nhdr) + namesz + descsize;
2549 }
2550
2551 #ifdef __x86
2552     if (from_linux) {
2553         size_t tcount, pid;
2554         lwp_info_t *lwp;
2555
2556         P->status.pr_dmodel = core_info->core_dmodel;
2557
2558         lwp = list_next(&core_info->core_lwp_head);
2559
2560         pid = P->status.pr_pid;
2561
2562         for (tcount = 0; tcount < core_info->core_nlwp;
2563              tcount++, lwp = list_next(lwp)) {
2564             dprintf("Linux thread with id %d\n", lwp->lwp_id);
2565
2566             /*
2567              * In the case we don't have a valid psinfo (i.e. pid is
2568              * 0, probably because of gdb creating the core) assume
2569              * lowest pid count is the first thread (what if the
2570              * next thread wraps the pid around?)
2571              */
2572             if (P->status.pr_pid == 0 &&
2573                 ((pid == 0 && lwp->lwp_id > 0) ||
2574                  (lwp->lwp_id < pid))) {
2575                 pid = lwp->lwp_id;
2576             }
2577
2578             if (P->status.pr_pid != pid) {
2579                 dprintf("No valid pid, setting to %ld\n", (ulong_t)pid);
2580                 P->status.pr_pid = pid;
2581                 P->psinfo.pr_pid = pid;
2582             }
2583
2584             /*
2585              * Consumers like mdb expect the first thread to actually have
2586              * an id of 1, on linux that is actually the pid. Find the the
2587              * thread with our process id, and set the id to 1
2588              */
2589             if ((lwp = lwpid2info(P, pid)) == NULL) {
2590                 dprintf("Couldn't find first thread\n");

```

```

2592         *perr = G_STRANGE;
2593         goto err;
2594     }
2595
2596     dprintf("setting representative thread: %d\n", lwp->lwp_id);
2597
2598     lwp->lwp_id = 1;
2599     lwp->lwp_status.pr_lwpid = 1;
2600
2601     /* set representative thread */
2602     (void) memcpy(&P->status.pr_lwp, &lwp->lwp_status,
2603                 sizeof (P->status.pr_lwp));
2604 #endif /* __x86 */
2605
2606     if (nleft != 0) {
2607         dprintf("Pgrab_core: note section malformed\n");
2608         *perr = G_STRANGE;
2609         goto err;
2610     }
2611
2612     if ((pagesize = Pgetauxval(P, AT_PAGESZ)) == -1) {
2613         pagesize = getpagesize();
2614         dprintf("AT_PAGESZ missing; defaulting to %d\n", pagesize);
2615     }
2616
2617     /*
2618      * Locate and label the mappings corresponding to the end of the
2619      * heap (MA_BREAK) and the base of the stack (MA_STACK).
2620      */
2621     if ((P->status.pr_brkbase != 0 || P->status.pr_brksize != 0) &&
2622         (brk_mp = Paddr2mptr(P, P->status.pr_brkbase +
2623                             P->status.pr_brksize - 1)) != NULL)
2624         brk_mp->map_pmap.pr_mflags |= MA_BREAK;
2625     else
2626         brk_mp = NULL;
2627
2628     if ((stk_mp = Paddr2mptr(P, P->status.pr_stkbase)) != NULL)
2629         stk_mp->map_pmap.pr_mflags |= MA_STACK;
2630
2631     /*
2632      * At this point, we have enough information to look for the
2633      * executable and open it: we have access to the auxv, a psinfo_t,
2634      * and the ability to read from mappings provided by the core file.
2635      */
2636     (void) Pfindexec(P, aout_path, core_exec_open, &aout);
2637     dprintf("P->execname = \"%s\"\n", P->execname ? P->execname : "NULL");
2638     execname = P->execname ? P->execname : "a.out";
2639
2640     /*
2641      * Iterate through the sections, looking for the .dynamic and .interp
2642      * sections. If we encounter them, remember their section pointers.
2643      */
2644     for (scn = NULL; (scn = elf_nextscn(aout.e_elf, scn)) != NULL; ) {
2645         char *sname;
2646
2647         if ((gelf_getshdr(scn, &shdr) == NULL) ||
2648             (sname = elf_strptr(aout.e_elf, aout.e_hdr.e_shstrndx,
2649                                (size_t)shdr.sh_name)) == NULL)
2650             continue;
2651
2652         if (strcmp(sname, ".interp") == 0)
2653             intp_scn = scn;
2654     }
2655
2656     /*

```

```

2658 * Get the AT_BASE auxv element. If this is missing (-1), then
2659 * we assume this is a statically-linked executable.
2660 */
2661 base_addr = Pgetauxval(P, AT_BASE);

2663 /*
2664 * In order to get librtld_db initialized, we'll need to identify
2665 * and name the mapping corresponding to the run-time linker. The
2666 * AT_BASE auxv element tells us the address where it was mapped,
2667 * and the .interp section of the executable tells us its path.
2668 * If for some reason that doesn't pan out, just use ld.so.1.
2669 */
2670 if (intp_scn != NULL && (dp = elf_getdata(intp_scn, NULL)) != NULL &&
2671     dp->d_size != 0) {
2672     dprintf(".interp = <%s>\n", (char *)dp->d_buf);
2673     interp = dp->d_buf;
2674 } else if (base_addr != (uintptr_t)-1L) {
2675     if (core_info->core_dmodel == PR_MODEL_LP64)
2676         interp = "/usr/lib/64/ld.so.1";
2677     else
2678         interp = "/usr/lib/ld.so.1";
2679 }

2681 dprintf(".interp section is missing or could not be read; "
2682         "defaulting to %s\n", interp);
2683 } else
2684     dprintf("detected statically linked executable\n");

2686 /*
2687 * If we have an AT_BASE element, name the mapping at that address
2688 * using the interpreter pathname. Name the corresponding data
2689 * mapping after the interpreter as well.
2690 */
2691 if (base_addr != (uintptr_t)-1L) {
2692     elf_file_t intf;

2694     P->map_ldso = core_name_mapping(P, base_addr, interp);

2696     if (core_elf_open(&intf, interp, ET_DYN, NULL) == 0) {
2697         rd_loadobj_t r1;
2698         map_info_t *dmp;

2700         r1.r1_base = base_addr;
2701         dmp = core_find_data(P, intf.e_elf, &r1);

2703         if (dmp != NULL) {
2704             dprintf("renamed data at %p to %s\n",
2705                 (void *)r1.r1_data_base, interp);
2706             (void) strncpy(dmp->map_pmap.pr_mapname,
2707                 interp, PRMAPSZ);
2708             dmp->map_pmap.pr_mapname[PRMAPSZ - 1] = '\0';
2709         }
2710     }

2712     core_elf_close(&intf);
2713 }

2715 /*
2716 * If we have an AT_ENTRY element, name the mapping at that address
2717 * using the special name "a.out" just like /proc does.
2718 */
2719 if ((addr = Pgetauxval(P, AT_ENTRY)) != (uintptr_t)-1L)
2720     P->map_exec = core_name_mapping(P, addr, "a.out");

2722 /*
2723 * If we're a statically linked executable (or we're on x86 and looking

```

```

2724 * at a Linux core dump), then just locate the executable's text and
2725 * data and name them after the executable.
2726 */
2727 #ifndef __x86
2728 if (base_addr == (uintptr_t)-1L) {
2729 #else
2730 if (base_addr == (uintptr_t)-1L || from_linux) {
2731 #endif
2732     dprintf("looking for text and data: %s\n", execname);
2733     map_info_t *tmp, *dmp;
2734     file_info_t *fp;
2735     rd_loadobj_t r1;

2737     if ((tmp = core_find_text(P, aout.e_elf, &r1)) != NULL &&
2738         (dmp = core_find_data(P, aout.e_elf, &r1)) != NULL) {
2739         (void) strncpy(tmp->map_pmap.pr_mapname,
2740             execname, PRMAPSZ);
2741         tmp->map_pmap.pr_mapname[PRMAPSZ - 1] = '\0';
2742         (void) strncpy(dmp->map_pmap.pr_mapname,
2743             execname, PRMAPSZ);
2744         dmp->map_pmap.pr_mapname[PRMAPSZ - 1] = '\0';
2745     }

2747     if ((P->map_exec = tmp) != NULL &&
2748         (fp = malloc(sizeof (file_info_t))) != NULL) {

2750         (void) memset(fp, 0, sizeof (file_info_t));

2752         list_link(fp, &P->file_head);
2753         tmp->map_file = fp;
2754         P->num_files++;

2756         fp->file_ref = 1;
2757         fp->file_fd = -1;

2759         fp->file_lo = malloc(sizeof (rd_loadobj_t));
2760         fp->file_lname = strdup(execname);

2762         if (fp->file_lo)
2763             *fp->file_lo = r1;
2764         if (fp->file_lname)
2765             fp->file_lbase = basename(fp->file_lname);
2766         if (fp->file_rname)
2767             fp->file_rbase = basename(fp->file_rname);

2769         (void) strcpy(fp->file_pname,
2770             P->mappings[0].map_pmap.pr_mapname);
2771         fp->file_map = tmp;

2773         Pbuild_file_syntab(P, fp);

2775         if (dmp != NULL) {
2776             dmp->map_file = fp;
2777             fp->file_ref++;
2778         }
2779     }

2782     core_elf_close(&aout);

2784 /*
2785 * We now have enough information to initialize librtld_db.
2786 * After it warms up, we can iterate through the load object chain
2787 * in the core, which will allow us to construct the file info
2788 * we need to provide symbol information for the other shared
2789 * libraries, and also to fill in the missing mapping names.

```

```
2790     */
2791     rd_log(_libproc_debug);
2793     if ((P->rap = rd_new(P)) != NULL) {
2794         (void) rd_loadobj_iter(P->rap, (rl_iter_f *)
2795             core_iter_mapping, P);
2797         if (core_info->core_errno != 0) {
2798             errno = core_info->core_errno;
2799             *perr = G_STRANGE;
2800             goto err;
2801         }
2802     } else
2803         dprintf("failed to initialize rtdb agent\n");
2805     /*
2806     * If there are sections, load them and process the data from any
2807     * sections that we can use to annotate the file_info_t's.
2808     */
2809     core_load_shdrs(P, &core);
2811     /*
2812     * If we previously located a stack or break mapping, and they are
2813     * still anonymous, we now assume that they were MAP_ANON mappings.
2814     * If brk_mp turns out to now have a name, then the heap is still
2815     * sitting at the end of the executable's data+bss mapping: remove
2816     * the previous MA_BREAK setting to be consistent with /proc.
2817     */
2818     if (stk_mp != NULL && stk_mp->map_pmap.pr_mapname[0] == '\0')
2819         stk_mp->map_pmap.pr_mflags |= MA_ANON;
2820     if (brk_mp != NULL && brk_mp->map_pmap.pr_mapname[0] == '\0')
2821         brk_mp->map_pmap.pr_mflags |= MA_ANON;
2822     else if (brk_mp != NULL)
2823         brk_mp->map_pmap.pr_mflags &= ~MA_BREAK;
2825     *perr = 0;
2826     return (P);
2828 err:
2829     Pfree(P);
2830     core_elf_close(&aout);
2831     return (NULL);
2832 }
2834 /*
2835 * Grab a core file using a pathname. We just open it and call Pgrab_core().
2836 */
2837 struct ps_prochandle *
2838 Pgrab_core(const char *core, const char *aout, int gflag, int *perr)
2839 {
2840     int fd, oflag = (gflag & PGRAB_RDONLY) ? O_RDONLY : O_RDWR;
2842     if ((fd = open64(core, oflag)) >= 0)
2843         return (Pgrab_core(fd, aout, perr));
2845     if (errno != ENOENT)
2846         *perr = G_STRANGE;
2847     else
2848         *perr = G_NOCORE;
2850     return (NULL);
2851 }
```

```

*****
40388 Thu Jun 30 22:02:20 2016
new/usr/src/lib/libproc/common/Pgcore.c
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
_____unchanged_portion_omitted_____

1035 /*
1036  * Don't explicitly stop the process; that's up to the consumer.
1037  */
1038 int
1039 Pfgcore(struct ps_prochandle *P, int fd, core_content_t content)
1040 {
1041     char plat[SYS_NMLN];
1042     char zonename[ZONENAME_MAX];
1043     int platlen = -1;
1044     pgcore_t pgc;
1045     off64_t poff, soff, doff, boff;
1046     struct utsname uts;
1047     uint_t nphdrs, nshdrs;

1049     if (ftruncate64(fd, 0) != 0)
1050         return (-1);

1052     if (content == CC_CONTENT_INVALID) {
1053         errno = EINVAL;
1054         return (-1);
1055     }

1057     /*
1058      * Cache the mappings and other useful data.
1059      */
1060     (void) Prd_agent(P);
1061     (void) Ppsinfo(P);

1063     pgc.P = P;
1064     pgc.pgc_fd = fd;
1065     pgc.pgc_poff = &poff;
1066     pgc.pgc_soff = &soff;
1067     pgc.pgc_doff = &doff;
1068     pgc.pgc_content = content;
1069     pgc.pgc_chunksz = PAGE_SIZE;
1070     if ((pgc.pgc_chunk = malloc(pgc.pgc_chunksz)) == NULL)
1071         return (-1);

1073     shstrtab_init(&pgc.pgc_shstrtab);

1075     /*
1076      * There are two PT_NOTE program headers for ancillary data, and
1077      * one for each mapping.
1078      */
1079     nphdrs = 2 + P->map_count;
1080     nshdrs = count_sections(&pgc);

1082     (void) Pplatform(P, plat, sizeof(plat));
1083     platlen = strlen(plat) + 1;
1084     Preadauxvec(P);
1085     (void) Puname(P, &uts);
1086     if (Pzonename(P, zonename, sizeof(zonename)) == NULL)
1087         zonename[0] = '\0';

1089     /*
1090      * The core file contents may require zero section headers, but if we

```

```

1091     * overflow the 16 bits allotted to the program header count in the ELF
1092     * header, we'll need that program header at index zero.
1093     */
1094     if (nshdrs == 0 && nphdrs >= PN_XNUM)
1095         nshdrs = 1;

1097     /*
1098      * Set up the ELF header.
1099      */
1100     if (P->status.pr_dmodel == PR_MODEL_ILP32) {
1101         Elf32_Ehdr ehdr;

1103         bzero(&ehdr, sizeof(ehdr));
1104         ehdr.e_ident[EI_MAG0] = ELFMAG0;
1105         ehdr.e_ident[EI_MAG1] = ELFMAG1;
1106         ehdr.e_ident[EI_MAG2] = ELFMAG2;
1107         ehdr.e_ident[EI_MAG3] = ELFMAG3;
1108         ehdr.e_type = ET_CORE;

1110         ehdr.e_ident[EI_CLASS] = ELFCLASS32;
1111 #if defined(__sparc)
1112         ehdr.e_machine = EM_SPARC;
1113         ehdr.e_ident[EI_DATA] = ELFDATA2MSB;
1114 #elif defined(__i386) || defined(__amd64)
1115         ehdr.e_machine = EM_386;
1116         ehdr.e_ident[EI_DATA] = ELFDATA2LSB;
1117 #else
1118 #error "unknown machine type"
1119 #endif
1120         ehdr.e_ident[EI_VERSION] = EV_CURRENT;

1122         ehdr.e_version = EV_CURRENT;
1123         ehdr.e_ehsize = sizeof(ehdr);

1125         if (nphdrs >= PN_XNUM)
1126             ehdr.e_phnum = PN_XNUM;
1127         else
1128             ehdr.e_phnum = (unsigned short)nphdrs;

1130         ehdr.e_phentsize = sizeof(Elf32_Phdr);
1131         ehdr.e_phoff = ehdr.e_ehsize;

1133         if (nshdrs > 0) {
1134             if (nshdrs >= SHN_LORESERVE)
1135                 ehdr.e_shnum = 0;
1136             else
1137                 ehdr.e_shnum = (unsigned short)nshdrs;

1139             if (nshdrs - 1 >= SHN_LORESERVE)
1140                 ehdr.e_shstrndx = SHN_XINDEX;
1141             else
1142                 ehdr.e_shstrndx = (unsigned short)(nshdrs - 1);

1144             ehdr.e_shentsize = sizeof(Elf32_Shdr);
1145             ehdr.e_shoff = ehdr.e_phoff + ehdr.e_phentsize * nphdrs;
1146         }

1148         if (gc_pwrite64(fd, &ehdr, sizeof(ehdr), 0) != 0)
1149             goto err;

1151         poff = ehdr.e_phoff;
1152         soff = ehdr.e_shoff;
1153         doff = boff = ehdr.e_ehsize +
1154             ehdr.e_phentsize * nphdrs +
1155             ehdr.e_shentsize * nshdrs;

```

```

1157 #ifdef _LP64
1158     } else {
1159         Elf64_Ehdr ehdr;

1161         bzero(&ehdr, sizeof (ehdr));
1162         ehdr.e_ident[EI_MAG0] = ELFMAG0;
1163         ehdr.e_ident[EI_MAG1] = ELFMAG1;
1164         ehdr.e_ident[EI_MAG2] = ELFMAG2;
1165         ehdr.e_ident[EI_MAG3] = ELFMAG3;
1166         ehdr.e_type = ET_CORE;

1168         ehdr.e_ident[EI_CLASS] = ELFCLASS64;
1169 #if defined(__sparc)
1170         ehdr.e_machine = EM_SPARCV9;
1171         ehdr.e_ident[EI_DATA] = ELFDATA2MSB;
1172 #elif defined(__i386) || defined(__amd64)
1173         ehdr.e_machine = EM_AMD64;
1174         ehdr.e_ident[EI_DATA] = ELFDATA2LSB;
1175 #else
1176 #error "unknown machine type"
1177 #endif
1178         ehdr.e_ident[EI_VERSION] = EV_CURRENT;

1180         ehdr.e_version = EV_CURRENT;
1181         ehdr.e_ehsize = sizeof (ehdr);

1183         if (nphdrs >= PN_XNUM)
1184             ehdr.e_phnum = PN_XNUM;
1185         else
1186             ehdr.e_phnum = (unsigned short)nphdrs;

1188         ehdr.e_phentsize = sizeof (Elf64_Phdr);
1189         ehdr.e_phoff = ehdr.e_ehsize;

1191         if (nshdrs > 0) {
1192             if (nshdrs >= SHN_LORESERVE)
1193                 ehdr.e_shnum = 0;
1194             else
1195                 ehdr.e_shnum = (unsigned short)nshdrs;

1197             if (nshdrs - 1 >= SHN_LORESERVE)
1198                 ehdr.e_shstrndx = SHN_XINDEX;
1199             else
1200                 ehdr.e_shstrndx = (unsigned short)(nshdrs - 1);

1202             ehdr.e_shentsize = sizeof (Elf64_Shdr);
1203             ehdr.e_shoff = ehdr.e_phoff + ehdr.e_phentsize * nphdrs;
1204         }

1206         if (gc_pwrite64(fd, &ehdr, sizeof (ehdr), 0) != 0)
1207             goto err;

1209         poff = ehdr.e_phoff;
1210         soff = ehdr.e_shoff;
1211         doff = boff = ehdr.e_ehsize +
1212             ehdr.e_phentsize * nphdrs +
1213             ehdr.e_shentsize * nshdrs;

1215 #endif /* _LP64 */
1216     }

1218     /*
1219     * Write the zero indexed section if it exists.
1220     */
1221     if (nshdrs > 0 && write_shdr(&pgc, STR_NONE, 0, 0, 0, 0,
1222         nshdrs >= SHN_LORESERVE ? nshdrs : 0,

```

```

1223         nshdrs - 1 >= SHN_LORESERVE ? nshdrs - 1 : 0,
1224         nphdrs >= PN_XNUM ? nphdrs : 0, 0, 0) != 0)
1225             goto err;

1227     /*
1228     * Construct the old-style note header and section.
1229     */

1231     if (P->status.pr_dmodel == PR_MODEL_NATIVE) {
1232         prpsinfo_t prpsinfo;

1234         mkprpsinfo(P, &prpsinfo);
1235         if (write_note(fd, NT_PRPSINFO, &prpsinfo, sizeof (prpsinfo_t),
1236             &doff) != 0) {
1237             goto err;
1238         }
1239         if (write_note(fd, NT_AUXV, P->auxv,
1240             P->nauxv * sizeof (P->auxv[0]), &doff) != 0) {
1241             goto err;
1242         }
1243 #ifdef _LP64
1244     } else {
1245         prpsinfo32_t pi32;
1246         auxv32_t *av32;
1247         size_t size = sizeof (auxv32_t) * P->nauxv;
1248         int i;

1250         mkprpsinfo32(P, &pi32);
1251         if (write_note(fd, NT_PRPSINFO, &pi32, sizeof (prpsinfo32_t),
1252             &doff) != 0) {
1253             goto err;
1254         }

1256         if ((av32 = malloc(size)) == NULL)
1257             goto err;

1259         for (i = 0; i < P->nauxv; i++) {
1260             auxv_n_to_32(&P->auxv[i], &av32[i]);
1261         }

1263         if (write_note(fd, NT_AUXV, av32, size, &doff) != 0) {
1264             free(av32);
1265             goto err;
1266         }

1268         free(av32);
1269 #endif /* _LP64 */
1270     }

1272     if (write_note(fd, NT_PLATFORM, plat, platlen, &doff) != 0)
1273         goto err;

1275     if (Plwp_iter_all(P, old_per_lwp, &pgc) != 0)
1276         goto err;

1278     if (P->status.pr_dmodel == PR_MODEL_ILP32) {
1279         Elf32_Phdr phdr;

1281         bzero(&phdr, sizeof (phdr));
1282         phdr.p_type = PT_NOTE;
1283         phdr.p_flags = PF_R;
1284         phdr.p_offset = (Elf32_Off)boff;
1285         phdr.p_filesz = doff - boff;
1286         boff = doff;

1288         if (gc_pwrite64(fd, &phdr, sizeof (phdr), poff) != 0)

```

```

1289         goto err;
1290     poff += sizeof (phdr);
1291 #ifdef _LP64
1292     } else {
1293         Elf64_Phdr phdr;
1294
1295         bzero(&phdr, sizeof (phdr));
1296         phdr.p_type = PT_NOTE;
1297         phdr.p_flags = PF_R;
1298         phdr.p_offset = boff;
1299         phdr.p_filesz = doff - boff;
1300         boff = doff;
1301
1302         if (gc_pwrite64(fd, &phdr, sizeof (phdr), poff) != 0)
1303             goto err;
1304         poff += sizeof (phdr);
1305 #endif /* _LP64 */
1306     }
1307
1308     /*
1309     * Construct the new-style note header and section.
1310     */
1311
1312     if (P->status.pr_dmodel == PR_MODEL_NATIVE) {
1313         if (write_note(fd, NT_PSINFO, &P->psinfo, sizeof (psinfo_t),
1314             &doff) != 0) {
1315             goto err;
1316         }
1317         if (write_note(fd, NT_PSTATUS, &P->status, sizeof (pstatus_t),
1318             &doff) != 0) {
1319             goto err;
1320         }
1321         if (write_note(fd, NT_AUXV, P->auxv,
1322             P->nauxv * sizeof (P->auxv[0]), &doff) != 0) {
1323             goto err;
1324         }
1325 #ifdef _LP64
1326     } else {
1327         psinfo32_t pi32;
1328         pstatus32_t ps32;
1329         auxv32_t *av32;
1330         size_t size = sizeof (auxv32_t) * P->nauxv;
1331         int i;
1332
1333         psinfo_n_to_32(&P->psinfo, &pi32);
1334         if (write_note(fd, NT_PSINFO, &pi32, sizeof (psinfo32_t),
1335             &doff) != 0) {
1336             goto err;
1337         }
1338         pstatus_n_to_32(&P->status, &ps32);
1339         if (write_note(fd, NT_PSTATUS, &ps32, sizeof (pstatus32_t),
1340             &doff) != 0) {
1341             goto err;
1342         }
1343         if ((av32 = malloc(size)) == NULL)
1344             goto err;
1345
1346         for (i = 0; i < P->nauxv; i++) {
1347             auxv_n_to_32(&P->auxv[i], &av32[i]);
1348         }
1349
1350         if (write_note(fd, NT_AUXV, av32, size, &doff) != 0) {
1351             free(av32);
1352             goto err;
1353         }

```

```

1355         free(av32);
1356 #endif /* _LP64 */
1357     }
1358
1359     if (write_note(fd, NT_PLATFORM, plat, platlen, &doff) != 0 ||
1360         write_note(fd, NT_UTSNAME, &uts, sizeof (uts), &doff) != 0 ||
1361         write_note(fd, NT_CONTENT, &content, sizeof (content), &doff) != 0)
1362         goto err;
1363
1364     {
1365         prcred_t cred, *cp;
1366         size_t size = sizeof (prcred_t);
1367
1368         if (Pcred(P, &cred, 0) != 0)
1369             goto err;
1370
1371         if (cred.pr_ngroups > 0)
1372             size += sizeof (gid_t) * (cred.pr_ngroups - 1);
1373         if ((cp = malloc(size)) == NULL)
1374             goto err;
1375
1376         if (Pcred(P, cp, cred.pr_ngroups) != 0 ||
1377             write_note(fd, NT_PRCRED, cp, size, &doff) != 0) {
1378             free(cp);
1379             goto err;
1380         }
1381
1382         free(cp);
1383     }
1384
1385     {
1386         prpriv_t *ppriv = NULL;
1387         const priv_impl_info_t *pinfo;
1388         size_t pprivsz, pinfosz;
1389
1390         if (Ppriv(P, &ppriv) == -1)
1391             goto err;
1392         pprivsz = PRIV_PPRIV_SIZE(ppriv);
1393
1394         if (write_note(fd, NT_PPRIV, ppriv, pprivsz, &doff) != 0) {
1395             Ppriv_free(P, ppriv);
1396             goto err;
1397         }
1398         Ppriv_free(P, ppriv);
1399
1400         if ((pinfo = getprivimplinfo()) == NULL)
1401             goto err;
1402         pinfosz = PRIV_IMPL_INFO_SIZE(pinfo);
1403
1404         if (write_note(fd, NT_PPRIVINFO, pinfo, pinfosz, &doff) != 0)
1405             goto err;
1406     }
1407
1408     if (write_note(fd, NT_ZONENAME, zonename, strlen(zonename) + 1,
1409         &doff) != 0)
1410         goto err;
1411
1412     {
1413         fditer_t iter;
1414         iter.fd_fd = fd;
1415         iter.fd_doff = &doff;
1416
1417         if (Pfdinfo_iter(P, iter_fd, &iter) != 0)
1418             goto err;
1419     }

```

```

1422     {
1423         prsecflags_t *psf = NULL;

1425         if (Psecflags(P, &psf) != 0)
1426             goto err;

1428         if (write_note(fd, NT_SECFLAGS, psf,
1429             sizeof (prsecflags_t), &doff) != 0) {
1430             Psecflags_free(psf);
1431             goto err;
1432         }

1434         Psecflags_free(psf);
1435     }

1437 #endif /* ! codereview */
1438 #if defined(__i386) || defined(__amd64)
1439     /* CSTYLED */
1440     {
1441         struct ssd *ldtp;
1442         size_t size;
1443         int nldt;

1445         /*
1446          * Only dump out non-zero sized LDT notes.
1447          */
1448         if ((nldt = Pldt(P, NULL, 0)) != 0) {
1449             size = sizeof (struct ssd) * nldt;
1450             if ((ldtp = malloc(size)) == NULL)
1451                 goto err;

1453             if (Pldt(P, ldtp, nldt) == -1 ||
1454                 write_note(fd, NT_LDT, ldtp, size, &doff) != 0) {
1455                 free(ldtp);
1456                 goto err;
1457             }

1459             free(ldtp);
1460         }
1461     }
1462 #endif /* __i386 || __amd64 */

1464     if (Plwp_iter_all(P, new_per_lwp, &pgc) != 0)
1465         goto err;

1467     if (P->status.pr_dmodel == PR_MODEL_ILP32) {
1468         Elf32_Phdr phdr;

1470         bzero(&phdr, sizeof (phdr));
1471         phdr.p_type = PT_NOTE;
1472         phdr.p_flags = PF_R;
1473         phdr.p_offset = (Elf32_Off)boff;
1474         phdr.p_filesz = doff - boff;
1475         boff = doff;

1477         if (gc_pwrite64(fd, &phdr, sizeof (phdr), poff) != 0)
1478             goto err;
1479         poff += sizeof (phdr);

1480 #ifndef _LP64
1481     } else {
1482         Elf64_Phdr phdr;

1484         bzero(&phdr, sizeof (phdr));
1485         phdr.p_type = PT_NOTE;
1486         phdr.p_flags = PF_R;

```

```

1487         phdr.p_offset = boff;
1488         phdr.p_filesz = doff - boff;
1489         boff = doff;

1491         if (gc_pwrite64(fd, &phdr, sizeof (phdr), poff) != 0)
1492             goto err;
1493         poff += sizeof (phdr);
1494 #endif /* _LP64 */
1495     }

1497     /*
1498     * Construct the headers for each mapping and write out its data
1499     * if the content parameter indicates that it should be present
1500     * in the core file.
1501     */
1502     if (Pmapping_iter(P, dump_map, &pgc) != 0)
1503         goto err;

1505     if (dump_sections(&pgc) != 0)
1506         goto err;

1508     if (write_shstrtab(P, &pgc) != 0)
1509         goto err;

1511     free(pgc.pgc_chunk);

1513     return (0);

1515 err:
1516     /*
1517     * Wipe out anything we may have written if there was an error.
1518     */
1519     (void) ftruncate64(fd, 0);
1520     free(pgc.pgc_chunk);

1522 #endif /* ! codereview */
1523     return (-1);
1524 }

1526 static const char *content_str[] = {
1527     "stack",          /* CC_CONTENT_STACK */
1528     "heap",          /* CC_CONTENT_HEAP */
1529     "shfile",        /* CC_CONTENT_SHFILE */
1530     "shanon",        /* CC_CONTENT_SHANON */
1531     "text",          /* CC_CONTENT_TEXT */
1532     "data",          /* CC_CONTENT_DATA */
1533     "rodata",        /* CC_CONTENT_RODATA */
1534     "anon",          /* CC_CONTENT_ANON */
1535     "shm",           /* CC_CONTENT_SHM */
1536     "ism",           /* CC_CONTENT_ISM */
1537     "dism",          /* CC_CONTENT_DISM */
1538     "ctf",           /* CC_CONTENT_CTF */
1539     "symtab",        /* CC_CONTENT_SYMTAB */
1540 };

1542 static uint_t ncontent_str = sizeof (content_str) / sizeof (content_str[0]);

1544 #define STREQ(a, b, n) (strlen(b) == (n) && strncmp(a, b, n) == 0)

1546 int
1547 proc_str2content(const char *str, core_content_t *cp)
1548 {
1549     const char *cur = str;
1550     int add = 1;
1551     core_content_t mask, content = 0;

```



```

1553     for (;;) {
1554         for (cur = str; isalpha(*cur); cur++)
1555             continue;
1557         if (STREQ(str, "default", cur - str)) {
1558             mask = CC_CONTENT_DEFAULT;
1559         } else if (STREQ(str, "all", cur - str)) {
1560             mask = CC_CONTENT_ALL;
1561         } else if (STREQ(str, "none", cur - str)) {
1562             mask = 0;
1563         } else {
1564             int i = 0;
1566             while (!STREQ(str, content_str[i], cur - str)) {
1567                 i++;
1569                 if (i >= ncontent_str)
1570                     return (-1);
1571             }
1573             mask = (core_content_t)1 << i;
1574         }
1576         if (add)
1577             content |= mask;
1578         else
1579             content &= ~mask;
1581         switch (*cur) {
1582             case '\0':
1583                 *cp = content;
1584                 return (0);
1585             case '+':
1586                 add = 1;
1587                 break;
1588             case '-':
1589                 add = 0;
1590                 break;
1591             default:
1592                 return (-1);
1593         }
1595         str = cur + 1;
1596     }
1597 }
1599 static int
1600 popc(core_content_t x)
1601 {
1602     int i;
1604     for (i = 0; x != 0; i++)
1605         x &= x - 1;
1607     return (i);
1608 }
1610 int
1611 proc_content2str(core_content_t content, char *buf, size_t size)
1612 {
1613     int nonecnt, defcnt, allcnt;
1614     core_content_t mask, bit;
1615     int first;
1616     uint_t index;
1617     size_t n, tot = 0;

```

```

1619     if (content == 0)
1620         return ((int)strncpy(buf, "none", size));
1622     if (content & ~CC_CONTENT_ALL)
1623         return ((int)strncpy(buf, "<invalid>", size));
1625     nonecnt = popc(content);
1626     defcnt = 1 + popc(content ^ CC_CONTENT_DEFAULT);
1627     allcnt = 1 + popc(content ^ CC_CONTENT_ALL);
1629     if (defcnt <= nonecnt && defcnt <= allcnt) {
1630         mask = content ^ CC_CONTENT_DEFAULT;
1631         first = 0;
1632         tot += (n = strncpy(buf, "default", size));
1633         if (n > size)
1634             n = size;
1635         buf += n;
1636         size -= n;
1637     } else if (allcnt < nonecnt) {
1638         mask = content ^ CC_CONTENT_ALL;
1639         first = 0;
1640         tot += (n = strncpy(buf, "all", size));
1641         if (n > size)
1642             n = size;
1643         buf += n;
1644         size -= n;
1645     } else {
1646         mask = content;
1647         first = 1;
1648     }
1650     while (mask != 0) {
1651         bit = mask ^ (mask & (mask - 1));
1653         if (!first) {
1654             if (size > 1) {
1655                 *buf = (bit & content) ? '+' : '-';
1656                 buf++;
1657                 size--;
1658             }
1660             tot++;
1661         }
1662         index = popc(bit - 1);
1663         tot += (n = strncpy(buf, content_str[index], size));
1664         if (n > size)
1665             n = size;
1666         buf += n;
1667         size -= n;
1669         mask ^= bit;
1670         first = 0;
1671     }
1673     return ((int)tot);
1674 }

```

```

*****
7204 Thu Jun 30 22:02:21 2016
new/usr/src/lib/libproc/common/Pidle.c
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
_____unchanged_portion_omitted_____

110 static const ps_ops_t P_idle_ops = {
111     .pop_pread      = Pread_idle,
112     .pop_pwrite     = Pwrite_idle,
113     .pop_cred       = (pop_cred_t)Pidle_int,
114     .pop_priv       = Ppriv_idle,
115     .pop_secflags   = (pop_secflags_t)Pidle_int,
116 #endif /* ! codereview */
117     .pop_psinfo     = (pop_psinfo_t)Pidle_voidp,
118     .pop_platform   = (pop_platform_t)Pidle_voidp,
119     .pop_uname      = (pop_uname_t)Pidle_int,
120     .pop_zonename   = (pop_zonename_t)Pidle_voidp,
121 #if defined(__i386) || defined(__amd64)
122     .pop_ldt        = (pop_ldt_t)Pidle_int
123 #endif
124 };

126 static int
127 idle_add_mapping(struct ps_prochandle *P, GElf_Phdr *php, file_info_t *fp)
128 {
129     pmap_t pmap;

131     dprintf("mapping base %llx filesz %llu memsz %llu offset %llu\n",
132         (u_longlong_t)php->p_vaddr, (u_longlong_t)php->p_filesz,
133         (u_longlong_t)php->p_memsz, (u_longlong_t)php->p_offset);

135     pmap.pr_vaddr = (uintptr_t)php->p_vaddr;
136     pmap.pr_size = php->p_filesz;
137     (void) strncpy(pmap.pr_mapname, fp->file_pname,
138         sizeof(pmap.pr_mapname));
139     pmap.pr_offset = php->p_offset;

141     pmap.pr_mflags = 0;
142     if (php->p_flags & PF_R)
143         pmap.pr_mflags |= MA_READ;
144     if (php->p_flags & PF_W)
145         pmap.pr_mflags |= MA_WRITE;
146     if (php->p_flags & PF_X)
147         pmap.pr_mflags |= MA_EXEC;

149     pmap.pr_pagesize = 0;
150     pmap.pr_shmid = -1;

152     return (Padd_mapping(P, php->p_offset, fp, &pmap));
153 }

155 struct ps_prochandle *
156 Pgrab_file(const char *fname, int *perr)
157 {
158     struct ps_prochandle *P = NULL;
159     char buf[PATH_MAX];
160     GElf_Ehdr ehdr;
161     Elf *elf = NULL;
162     size_t phnum;
163     file_info_t *fp = NULL;
164     int fd;
165     int i;

```

```

167     if ((fd = open64(fname, O_RDONLY)) < 0) {
168         dprintf("couldn't open file");
169         *perr = (errno == ENOENT) ? G_NOEXEC : G_STRANGE;
170         return (NULL);
171     }

173     if (elf_version(EV_CURRENT) == EV_NONE) {
174         dprintf("libproc ELF version is more recent than libelf");
175         *perr = G_ELF;
176         goto err;
177     }

179     if ((P = calloc(1, sizeof(struct ps_prochandle))) == NULL) {
180         *perr = G_STRANGE;
181         goto err;
182     }

184     (void) mutex_init(&P->proc_lock, USYNC_THREAD, NULL);
185     P->state = PS_IDLE;
186     P->pid = (pid_t)-1;
187     P->asfd = fd;
188     P->ctlfd = -1;
189     P->statfd = -1;
190     P->agentctlfd = -1;
191     P->agentstatfd = -1;
192     P->info_valid = -1;
193     Pinit_ops(&P->ops, &P_idle_ops);
194     PinitSYM(P);

196     if ((elf = elf_begin(fd, ELF_C_READ, NULL)) == NULL) {
197         *perr = G_ELF;
198         return (NULL);
199     }

201     /*
202      * Construct a file_info_t that corresponds to this file.
203      */
204     if ((fp = calloc(1, sizeof(file_info_t))) == NULL) {
205         *perr = G_STRANGE;
206         goto err;
207     }

209     if ((fp->file_lo = calloc(1, sizeof(rd_loadobj_t))) == NULL) {
210         *perr = G_STRANGE;
211         goto err;
212     }

214     if (*fname == '/') {
215         (void) strncpy(fp->file_pname, fname, sizeof(fp->file_pname));
216     } else {
217         size_t sz;

219         if (getcwd(fp->file_pname, sizeof(fp->file_pname) - 1) ==
220             NULL) {
221             *perr = G_STRANGE;
222             goto err;
223         }

225         sz = strlen(fp->file_pname);
226         (void) snprintf(&fp->file_pname[sz],
227             sizeof(fp->file_pname) - sz, "%s", fname);
228     }

230     fp->file_fd = fd;
231     fp->file_lo->rl_lmident = LM_ID_BASE;

```

```

232     if ((fp->file_lname = strdup(fp->file_pname)) == NULL) {
233         *perr = G_STRANGE;
234         goto err;
235     }
236     fp->file_lbase = basename(fp->file_lname);

238     if ((P->execname = strdup(fp->file_pname)) == NULL) {
239         *perr = G_STRANGE;
240         goto err;
241     }

243     P->num_files++;
244     list_link(fp, &P->file_head);

246     if (gelf_getehdr(elf, &ehdr) == NULL) {
247         *perr = G_STRANGE;
248         goto err;
249     }

251     if (elf_getphdrnum(elf, &phnum) == -1) {
252         *perr = G_STRANGE;
253         goto err;
254     }

256     dprintf("Pgrab_file: program header count = %lu\n", (ulong_t)phnum);

258     /*
259     * Sift through the program headers making the relevant maps.
260     */
261     for (i = 0; i < phnum; i++) {
262         GElf_Phdr phdr, *php;

264         if ((php = gelf_getphdr(elf, i, &phdr)) == NULL) {
265             *perr = G_STRANGE;
266             goto err;
267         }

269         if (php->p_type != PT_LOAD)
270             continue;

272         if (idle_add_mapping(P, php, fp) != 0) {
273             *perr = G_STRANGE;
274             goto err;
275         }
276     }
277     Psort_mappings(P);

279     (void) elf_end(elf);

281     P->map_exec = fp->file_map;

283     P->status.pr_flags = PR_STOPPED;
284     P->status.pr_nlwp = 0;
285     P->status.pr_pid = (pid_t)-1;
286     P->status.pr_ppid = (pid_t)-1;
287     P->status.pr_pgid = (pid_t)-1;
288     P->status.pr_sid = (pid_t)-1;
289     P->status.pr_taskid = (taskid_t)-1;
290     P->status.pr_projid = (projid_t)-1;
291     P->status.pr_zoneid = (zoneid_t)-1;
292     switch (ehdr.e_ident[EI_CLASS]) {
293     case ELFCLASS32:
294         P->status.pr_dmodel = PR_MODEL_ILP32;
295         break;
296     case ELFCLASS64:
297         P->status.pr_dmodel = PR_MODEL_LP64;

```

```

298         break;
299     default:
300         *perr = G_FORMAT;
301         goto err;
302     }

304     /*
305     * Pfindobj() checks what zone a process is associated with, so
306     * we call it after initializing pr_zoneid to -1. This ensures
307     * we don't get associated with any zone on the system.
308     */
309     if (Pfindobj(P, fp->file_lname, buf, sizeof (buf)) != NULL) {
310         free(P->execname);
311         P->execname = strdup(buf);
312         if ((fp->file_rname = strdup(buf)) != NULL)
313             fp->file_rbase = basename(fp->file_rname);
314     }

316     /*
317     * The file and map lists are complete, and will never need to be
318     * adjusted.
319     */
320     P->info_valid = 1;

322     return (P);
323 err:
324     (void) close(fd);
325     if (P != NULL)
326         Pfree(P);
327     if (elf != NULL)
328         (void) elf_end(elf);
329     return (NULL);
330 }

```

```

*****
6520 Thu Jun 30 22:02:22 2016
new/usr/src/lib/libproc/common/Putil.c
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
_____unchanged_portion_omitted_____

```

```

181 static const ps_ops_t P_default_ops = {
182     .pop_pread      = (pop_pread_t)Pdefault_ssize_t,
183     .pop_pwrite     = (pop_pwrite_t)Pdefault_ssize_t,
184     .pop_read_maps  = (pop_read_maps_t)Pdefault_int,
185     .pop_read_aux   = (pop_read_aux_t)Pdefault_void,
186     .pop_cred       = (pop_cred_t)Pdefault_int,
187     .pop_priv       = (pop_priv_t)Pdefault_int,
188     .pop_psinfo     = (pop_psinfo_t)Pdefault_voidp,
189     .pop_status     = (pop_status_t)Pdefault_void,
190     .pop_lstatus    = (pop_lstatus_t)Pdefault_voidp,
191     .pop_lpsinfo    = (pop_lpsinfo_t)Pdefault_voidp,
192     .pop_fini       = (pop_fini_t)Pdefault_void,
193     .pop_platform   = (pop_platform_t)Pdefault_voidp,
194     .pop_username   = (pop_username_t)Pdefault_int,
195     .pop_zonename   = (pop_zonename_t)Pdefault_voidp,
196     .pop_execname   = (pop_execname_t)Pdefault_voidp,
197     .pop_secflags   = (pop_secflags_t)Pdefault_int,
198 #endif /* ! codereview */
199 #if defined(__i386) || defined(__amd64)
200     .pop_ldt        = (pop_ldt_t)Pdefault_int
201 #endif
202 };

204 /*
205  * Initialize the destination ops vector with functions from the source.
206  * Functions which are NULL in the source ops vector are set to corresponding
207  * default function in the destination vector.
208  */
209 void
210 Pinit_ops(ps_ops_t *dst, const ps_ops_t *src)
211 {
212     *dst = P_default_ops;

214     if (src->pop_pread != NULL)
215         dst->pop_pread = src->pop_pread;
216     if (src->pop_pwrite != NULL)
217         dst->pop_pwrite = src->pop_pwrite;
218     if (src->pop_read_maps != NULL)
219         dst->pop_read_maps = src->pop_read_maps;
220     if (src->pop_read_aux != NULL)
221         dst->pop_read_aux = src->pop_read_aux;
222     if (src->pop_cred != NULL)
223         dst->pop_cred = src->pop_cred;
224     if (src->pop_priv != NULL)
225         dst->pop_priv = src->pop_priv;
226     if (src->pop_psinfo != NULL)
227         dst->pop_psinfo = src->pop_psinfo;
228     if (src->pop_status != NULL)
229         dst->pop_status = src->pop_status;
230     if (src->pop_lstatus != NULL)
231         dst->pop_lstatus = src->pop_lstatus;
232     if (src->pop_lpsinfo != NULL)
233         dst->pop_lpsinfo = src->pop_lpsinfo;
234     if (src->pop_fini != NULL)
235         dst->pop_fini = src->pop_fini;
236     if (src->pop_platform != NULL)

```

```

237     dst->pop_platform = src->pop_platform;
238     if (src->pop_username != NULL)
239         dst->pop_username = src->pop_username;
240     if (src->pop_zonename != NULL)
241         dst->pop_zonename = src->pop_zonename;
242     if (src->pop_execname != NULL)
243         dst->pop_execname = src->pop_execname;
244     if (src->pop_secflags != NULL)
245         dst->pop_secflags = src->pop_secflags;
246 #endif /* ! codereview */
247 #if defined(__i386) || defined(__amd64)
248     if (src->pop_ldt != NULL)
249         dst->pop_ldt = src->pop_ldt;
250 #endif
251 }

```

```

*****
31278 Thu Jun 30 22:02:23 2016
new/usr/src/lib/libproc/common/libproc.h
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */

22 /*
23  * Copyright 2009 Sun Microsystems, Inc. All rights reserved.
24  * Use is subject to license terms.
25  *
26  * Portions Copyright 2007 Chad Mynhier
27  * Copyright 2012 DEY Storage Systems, Inc. All rights reserved.
28  * Copyright 2015, Joyent, Inc.
29  * Copyright (c) 2013 by Delphix. All rights reserved.
30  */

32 /*
33  * Interfaces available from the process control library, libproc.
34  */

36 #ifndef _LIBPROC_H
37 #define _LIBPROC_H

39 #include <stdlib.h>
40 #include <unistd.h>
41 #include <fcntl.h>
42 #include <nlist.h>
43 #include <door.h>
44 #include <gelf.h>
45 #include <proc_service.h>
46 #include <rtld_db.h>
47 #include <procfs.h>
48 #include <ucred.h>
49 #include <rctl.h>
50 #include <libctf.h>
51 #include <sys/stat.h>
52 #include <sys/statvfs.h>
53 #include <sys/auxv.h>
54 #include <sys/resource.h>
55 #include <sys/socket.h>
56 #include <sys/utsname.h>
57 #include <sys/corectl.h>
58 #include <sys/secflags.h>

```

```

59 #endif /* ! codereview */
60 #if defined(__i386) || defined(__amd64)
61 #include <sys/sysi86.h>
62 #endif

64 #ifdef __cplusplus
65 extern "C" {
66 #endif

68 /*
69  * Opaque structure tag reference to a process control structure.
70  * Clients of libproc cannot look inside the process control structure.
71  * The implementation of struct ps_prochandle can change w/o affecting clients.
72  */
73 struct ps_prochandle;

75 /*
76  * Opaque structure tag reference to an lwp control structure.
77  */
78 struct ps_lwphandle;

80 extern int    _libproc_debug; /* set non-zero to enable debugging fprintfs */
81 extern int    _libproc_no_qsort; /* set non-zero to inhibit sorting */
82 extern int    _libproc_incore_elf; /* of symbol tables */
83 /* only use in-core elf data */

85 #if defined(__sparc)
86 #define R_RVAL1_R_00 /* register holding a function return value */
87 #define R_RVAL2_R_01 /* 32 more bits for a 64-bit return value */
88 #endif /* __sparc */

90 #if defined(__amd64)
91 #define R_PC      REG_RIP
92 #define R_SP      REG_RSP
93 #define R_RVAL1 REG_RAX /* register holding a function return value */
94 #define R_RVAL2 REG_RDX /* 32 more bits for a 64-bit return value */
95 #elif defined(__i386)
96 #define R_PC      EIP
97 #define R_SP      UESP
98 #define R_RVAL1 EAX /* register holding a function return value */
99 #define R_RVAL2 EDX /* 32 more bits for a 64-bit return value */
100 #endif /* __amd64 || __i386 */

102 #define R_RVAL R_RVAL1 /* simple function return value register */

104 /* maximum sizes of things */
105 #define PRMAXSIG (32 * sizeof (sigset_t) / sizeof (uint32_t))
106 #define PRMAXFAULT (32 * sizeof (fltset_t) / sizeof (uint32_t))
107 #define PRMAXSYS (32 * sizeof (sysset_t) / sizeof (uint32_t))

109 /* State values returned by Pstate() */
110 #define PS_RUN 1 /* process is running */
111 #define PS_STOP 2 /* process is stopped */
112 #define PS_LOST 3 /* process is lost to control (EAGAIN) */
113 #define PS_UNDEAD 4 /* process is terminated (zombie) */
114 #define PS_DEAD 5 /* process is terminated (core file) */
115 #define PS_IDLE 6 /* process has not been run */

117 /* Flags accepted by Pgrab() */
118 #define PGRAB_RETAIN 0x01 /* Retain tracing flags, else clear flags */
119 #define PGRAB_FORCE 0x02 /* Open the process w/o O_EXCL */
120 #define PGRAB_RDONLY 0x04 /* Open the process or core w/ O_RDONLY */
121 #define PGRAB_NOSTOP 0x08 /* Open the process but do not stop it */
122 #define PGRAB_INCORE 0x10 /* Use in-core data to build symbol tables */

124 /* Error codes from Pcreate() */

```

```

125 #define C_STRANGE      -1    /* Unanticipated error, errno is meaningful */
126 #define C_FORK         1     /* Unable to fork */
127 #define C_PERM         2     /* No permission (file set-id or unreadable) */
128 #define C_NOEXEC       3     /* Cannot execute file */
129 #define C_INTR         4     /* Interrupt received while creating */
130 #define C_LP64         5     /* Program is_LP64, self is_ILP32 */
131 #define C_NOENT        6     /* Cannot find executable file */

133 /* Error codes from Pgrab(), Pgrab_core(), and Pgrab_core() */
134 #define G_STRANGE      -1    /* Unanticipated error, errno is meaningful */
135 #define G_NOPROC       1     /* No such process */
136 #define G_NOCORE       2     /* No such core file */
137 #define G_NOPROCCORE   3     /* No such proc or core (for proc_arg_grab) */
138 #define G_NOEXEC       4     /* Cannot locate executable file */
139 #define G_ZOMB         5     /* Zombie process */
140 #define G_PERM         6     /* No permission */
141 #define G_BUSY         7     /* Another process has control */
142 #define G_SYS          8     /* System process */
143 #define G_SELF         9     /* Process is self */
144 #define G_INTR        10    /* Interrupt received while grabbing */
145 #define G_LP64        11    /* Process is_LP64, self is_ILP32 */
146 #define G_FORMAT      12    /* File is not an ELF format core file */
147 #define G_ELF         13    /* Libelf error, elf_errno() is meaningful */
148 #define G_NOTE        14    /* Required PT_NOTE Phdr not present in core */
149 #define G_ISAINVAL    15    /* Wrong ELF machine type */
150 #define G_BADLWPS     16    /* Bad '/lwps' specification */
151 #define G_NOFD        17    /* No more file descriptors */

154 /* Flags accepted by Prelease */
155 #define PRELEASE_CLEAR 0x10  /* Clear all tracing flags */
156 #define PRELEASE_RETAIN 0x20 /* Retain final tracing flags */
157 #define PRELEASE_HANG  0x40  /* Leave the process stopped */
158 #define PRELEASE_KILL  0x80  /* Terminate the process */

160 typedef struct {           /* argument descriptor for system call (Psyscall) */
161     long    arg_value;     /* value of argument given to system call */
162     void    *arg_object;   /* pointer to object in controlling process */
163     char    arg_type;      /* AT_BYVAL, AT_BYREF */
164     char    arg_inout;     /* AI_INPUT, AI_OUTPUT, AI_INOUT */
165     ushort_t arg_size;    /* if AT_BYREF, size of object in bytes */
166 } argdes_t;

168 /* values for type */
169 #define AT_BYVAL       1
170 #define AT_BYREF       2

172 /* values for inout */
173 #define AI_INPUT       1
174 #define AI_OUTPUT      2
175 #define AI_INOUT       3

177 /* maximum number of syscall arguments */
178 #define MAXARGS       8

180 /* maximum size in bytes of a BYREF argument */
181 #define MAXARGL       (4*1024)

183 /*
184  * Ops vector definition for the Pgrab_ops().
185  */
186 typedef ssize_t (*pop_read_t)(struct ps_prochandle *, void *, size_t,
187     uintptr_t, void *);
188 typedef ssize_t (*pop_write_t)(struct ps_prochandle *, const void *, size_t,
189     uintptr_t, void *);
190 typedef int (*pop_read_maps_t)(struct ps_prochandle *, prmap_t **, ssize_t *,

```

```

191     void *);
192 typedef void (*pop_read_aux_t)(struct ps_prochandle *, auxv_t **, int *,
193     void *);
194 typedef int (*pop_cred_t)(struct ps_prochandle *, prcred_t *, int,
195     void *);
196 typedef int (*pop_priv_t)(struct ps_prochandle *, prpriv_t **, void *);
197 typedef int (*pop_secflags_t)(struct ps_prochandle *, prsecflags_t **, void *);
198 #endif /* ! codereview */
199 typedef const psinfo_t *(*pop_psinfo_t)(struct ps_prochandle *, psinfo_t *,
200     void *);
201 typedef void (*pop_status_t)(struct ps_prochandle *, pstatus_t *, void *);
202 typedef prheader_t *(*pop_lstatus_t)(struct ps_prochandle *, void *);
203 typedef prheader_t *(*pop_lpsinfo_t)(struct ps_prochandle *, void *);
204 typedef void (*pop_fini_t)(struct ps_prochandle *, void *);
205 typedef char *(*pop_platform_t)(struct ps_prochandle *, char *, size_t, void *);
206 typedef int (*pop_uname_t)(struct ps_prochandle *, struct utsname *, void *);
207 typedef char *(*pop_zonename_t)(struct ps_prochandle *, char *, size_t, void *);
208 typedef char *(*pop_execname_t)(struct ps_prochandle *, char *, size_t, void *);
209 #if defined(__i386) || defined(__amd64)
210 typedef int (*pop_ldt_t)(struct ps_prochandle *, struct ssd *, int, void *);
211 #endif

213 typedef struct ps_ops {
214     pop_read_t           pop_read;
215     pop_write_t          pop_write;
216     pop_read_maps_t     pop_read_maps;
217     pop_read_aux_t      pop_read_aux;
218     pop_cred_t           pop_cred;
219     pop_priv_t           pop_priv;
220     pop_psinfo_t         pop_psinfo;
221     pop_status_t         pop_status;
222     pop_lstatus_t        pop_lstatus;
223     pop_lpsinfo_t        pop_lpsinfo;
224     pop_fini_t           pop_fini;
225     pop_platform_t       pop_platform;
226     pop_uname_t          pop_uname;
227     pop_zonename_t       pop_zonename;
228     pop_execname_t       pop_execname;
229     pop_secflags_t       pop_secflags;
230 #endif /* ! codereview */
231 #if defined(__i386) || defined(__amd64)
232     pop_ldt_t             pop_ldt;
233 #endif
234 } ps_ops_t;

236 /*
237  * Function prototypes for routines in the process control package.
238  */
239 extern struct ps_prochandle *Pcreate(const char *, char *const *,
240     int *, char *, size_t);
241 extern struct ps_prochandle *Pxcree(const char *, char *const *,
242     char *const *, int *, char *, size_t);

244 extern const char *Pcreate_error(int);

246 extern struct ps_prochandle *Pgrab(pid_t, int, int *);
247 extern struct ps_prochandle *Pgrab_core(const char *, const char *, int, int *);
248 extern struct ps_prochandle *Pgrab_core(int, const char *, int *);
249 extern struct ps_prochandle *Pgrab_file(const char *, int *);
250 extern struct ps_prochandle *Pgrab_ops(pid_t, void *, const ps_ops_t *, int);
251 extern const char *Pgrab_error(int);

253 extern int    Preopen(struct ps_prochandle *);
254 extern void   Prelease(struct ps_prochandle *, int);
255 extern void   Pfree(struct ps_prochandle *);

```

```

257 extern int Pasfd(struct ps_prochandle *);
258 extern char *Pbrandname(struct ps_prochandle *, char *, size_t);
259 extern int Pctlfid(struct ps_prochandle *);
260 extern int Pcreate_agent(struct ps_prochandle *);
261 extern void Pdestroy_agent(struct ps_prochandle *);
262 extern int Pstopstatus(struct ps_prochandle *, long, uint_t);
263 extern int Pwait(struct ps_prochandle *, uint_t);
264 extern int Pstop(struct ps_prochandle *, uint_t);
265 extern int Pdstop(struct ps_prochandle *);
266 extern int Pstate(struct ps_prochandle *);
267 extern const psinfo_t *Ppsinfo(struct ps_prochandle *);
268 extern const pstatus_t *Pstatus(struct ps_prochandle *);
269 extern int Pcred(struct ps_prochandle *, prcred_t *, int);
270 extern int Psetcred(struct ps_prochandle *, const prcred_t *);
271 extern int Ppriv(struct ps_prochandle *, prpriv_t **);
272 extern void Ppriv_free(struct ps_prochandle *, prpriv_t *);
273 extern int Psetpriv(struct ps_prochandle *, prpriv_t *);
274 extern void *Pprivinfo(struct ps_prochandle *);
275 extern int Psetzoneid(struct ps_prochandle *, zoneid_t);
276 extern int Pgetareg(struct ps_prochandle *, int, prgreg_t *);
277 extern int Pputareg(struct ps_prochandle *, int, prgreg_t *);
278 extern int Psetrun(struct ps_prochandle *, int, int);
279 extern int Psecflags(struct ps_prochandle *, prsecflags_t **);
280 extern void Psecflags_free(prsecflags_t *);
281 #endif /* ! codereview */
282 extern ssize_t Pread(struct ps_prochandle *, void *, size_t, uintptr_t);
283 extern ssize_t Pread_string(struct ps_prochandle *, char *, size_t, uintptr_t);
284 extern ssize_t Pwrite(struct ps_prochandle *, const void *, size_t, uintptr_t);
285 extern int Pclearsig(struct ps_prochandle *);
286 extern int Pclearfault(struct ps_prochandle *);
287 extern int Psetbkpt(struct ps_prochandle *, uintptr_t, ulong_t);
288 extern int Pdelbkpt(struct ps_prochandle *, uintptr_t, ulong_t);
289 extern int Pxebkpt(struct ps_prochandle *, ulong_t);
290 extern int Psetwapt(struct ps_prochandle *, const prwatch_t *);
291 extern int Pdelwapt(struct ps_prochandle *, const prwatch_t *);
292 extern int Pxecwapt(struct ps_prochandle *, const prwatch_t *);
293 extern int Psetflags(struct ps_prochandle *, long);
294 extern int Punsetflags(struct ps_prochandle *, long);
295 extern int Psignal(struct ps_prochandle *, int, int);
296 extern int Pfault(struct ps_prochandle *, int, int);
297 extern int Psysentry(struct ps_prochandle *, int, int);
298 extern int Psysexit(struct ps_prochandle *, int, int);
299 extern void Psetsignal(struct ps_prochandle *, const sigset_t *);
300 extern void Psetfault(struct ps_prochandle *, const fltset_t *);
301 extern void Psetsysentry(struct ps_prochandle *, const sysset_t *);
302 extern void Psetsysexit(struct ps_prochandle *, const sysset_t *);

304 extern void Psync(struct ps_prochandle *);
305 extern int Psyscall(struct ps_prochandle *, sysret_t *,
306 int, uint_t, argdes_t *);
307 extern int Pisprocdir(struct ps_prochandle *, const char *);

309 /*
310 * Function prototypes for lwp-specific operations.
311 */
312 extern struct ps_lwphandle *Lgrab(struct ps_prochandle *, lwpid_t, int *);
313 extern const char *Lgrab_error(int);

315 extern struct ps_prochandle *Lprochandle(struct ps_lwphandle *);
316 extern void Lfree(struct ps_lwphandle *);

318 extern int Lctlfid(struct ps_lwphandle *);
319 extern int Lwait(struct ps_lwphandle *, uint_t);
320 extern int Lstop(struct ps_lwphandle *, uint_t);
321 extern int Ldstop(struct ps_lwphandle *);
322 extern int Lstate(struct ps_lwphandle *);

```

```

323 extern const lwpstatus_t *Lpsinfo(struct ps_lwphandle *);
324 extern const lwpstatus_t *Lstatus(struct ps_lwphandle *);
325 extern int Lgetareg(struct ps_lwphandle *, int, prgreg_t *);
326 extern int Lputareg(struct ps_lwphandle *, int, prgreg_t *);
327 extern int Lsetrun(struct ps_lwphandle *, int, int);
328 extern int Lclearsig(struct ps_lwphandle *);
329 extern int Lclearfault(struct ps_lwphandle *);
330 extern int Lxecbkpt(struct ps_lwphandle *, ulong_t);
331 extern int Lxecwapt(struct ps_lwphandle *, const prwatch_t *);
332 extern void Lsync(struct ps_lwphandle *);

334 extern int Lstack(struct ps_lwphandle *, stack_t *);
335 extern int Lmain_stack(struct ps_lwphandle *, stack_t *);
336 extern int Lalt_stack(struct ps_lwphandle *, stack_t *);

338 /*
339 * Function prototypes for system calls forced on the victim process.
340 */
341 extern int pr_open(struct ps_prochandle *, const char *, int, mode_t);
342 extern int pr_creat(struct ps_prochandle *, const char *, mode_t);
343 extern int pr_close(struct ps_prochandle *, int);
344 extern int pr_access(struct ps_prochandle *, const char *, int);
345 extern int pr_door_info(struct ps_prochandle *, int, struct door_info *);
346 extern void *pr_mmap(struct ps_prochandle *,
347 void *, size_t, int, int, off_t);
348 extern void *pr_zmap(struct ps_prochandle *,
349 void *, size_t, int, int);
350 extern int pr_munmap(struct ps_prochandle *, void *, size_t);
351 extern int pr_memcntl(struct ps_prochandle *,
352 caddr_t, size_t, int, caddr_t, int, int);
353 extern int pr_meminfo(struct ps_prochandle *, const uint64_t *addrs,
354 int addr_count, const uint_t *info, int info_count,
355 uint64_t *outdata, uint_t *validity);
356 extern int pr_sigaction(struct ps_prochandle *,
357 int, const struct sigaction *, struct sigaction *);
358 extern int pr_getitimer(struct ps_prochandle *,
359 int, struct itimerval *);
360 extern int pr_setitimer(struct ps_prochandle *,
361 int, const struct itimerval *, struct itimerval *);
362 extern int pr_ioctl(struct ps_prochandle *, int, int, void *, size_t);
363 extern int pr_fcntl(struct ps_prochandle *, int, int, void *);
364 extern int pr_stat(struct ps_prochandle *, const char *, struct stat *);
365 extern int pr_lstat(struct ps_prochandle *, const char *, struct stat *);
366 extern int pr_fstat(struct ps_prochandle *, int, struct stat *);
367 extern int pr_stat64(struct ps_prochandle *, const char *,
368 struct stat64 *);
369 extern int pr_lstat64(struct ps_prochandle *, const char *,
370 struct stat64 *);
371 extern int pr_fstat64(struct ps_prochandle *, int, struct stat64 *);
372 extern int pr_statvfs(struct ps_prochandle *, const char *, statvfs_t *);
373 extern int pr_fstatvfs(struct ps_prochandle *, int, statvfs_t *);
374 extern projid_t pr_getprojid(struct ps_prochandle *Pr);
375 extern taskid_t pr_gettaskid(struct ps_prochandle *Pr);
376 extern taskid_t pr_settaskid(struct ps_prochandle *Pr, projid_t project,
377 int flags);
378 extern zoneid_t pr_getzoneid(struct ps_prochandle *Pr);
379 extern int pr_getrctl(struct ps_prochandle *,
380 const char *, rctlblk_t *, rctlblk_t *, int);
381 extern int pr_setrctl(struct ps_prochandle *,
382 const char *, rctlblk_t *, rctlblk_t *, int);
383 extern int pr_getrlimit(struct ps_prochandle *,
384 int, struct rlimit *);
385 extern int pr_setrlimit(struct ps_prochandle *,
386 int, const struct rlimit *);
387 extern int pr_setprojctl(struct ps_prochandle *, const char *,
388 rctlblk_t *, size_t, int);

```

```

389 #if defined(_LARGEFILE64_SOURCE)
390 extern int pr_getrlimit64(struct ps_prochandle *,
391 int, struct rlimit64 *);
392 extern int pr_setrlimit64(struct ps_prochandle *,
393 int, const struct rlimit64 *);
394 #endif /* _LARGEFILE64_SOURCE */
395 extern int pr_lwp_exit(struct ps_prochandle *);
396 extern int pr_exit(struct ps_prochandle *, int);
397 extern int pr_waitid(struct ps_prochandle *,
398 idtype_t, id_t, siginfo_t *, int);
399 extern off_t pr_lseek(struct ps_prochandle *, int, off_t, int);
400 extern offset_t pr_llseek(struct ps_prochandle *, int, offset_t, int);
401 extern int pr_rename(struct ps_prochandle *, const char *, const char *);
402 extern int pr_link(struct ps_prochandle *, const char *, const char *);
403 extern int pr_unlink(struct ps_prochandle *, const char *);
404 extern int pr_getpeerucred(struct ps_prochandle *, int, ucred_t **);
405 extern int pr_getpeername(struct ps_prochandle *,
406 int, struct sockaddr *, socklen_t *);
407 extern int pr_getsockname(struct ps_prochandle *,
408 int, struct sockaddr *, socklen_t *);
409 extern int pr_getsockopt(struct ps_prochandle *,
410 int, int, int, void *, int *);
411 extern int pr_processor_bind(struct ps_prochandle *,
412 idtype_t, id_t, int, int *);
414 /*
415 * Function prototypes for accessing per-LWP register information.
416 */
417 extern int Plwp_getregs(struct ps_prochandle *, lwpid_t, prgregset_t);
418 extern int Plwp_setregs(struct ps_prochandle *, lwpid_t, const prgregset_t);
420 extern int Plwp_getfpregs(struct ps_prochandle *, lwpid_t, prfpregset_t *);
421 extern int Plwp_setfpregs(struct ps_prochandle *, lwpid_t,
422 const prfpregset_t *);
424 #if defined(__sparc)
426 extern int Plwp_getxregs(struct ps_prochandle *, lwpid_t, prxregset_t *);
427 extern int Plwp_setxregs(struct ps_prochandle *, lwpid_t, const prxregset_t *);
429 extern int Plwp_getgwindows(struct ps_prochandle *, lwpid_t, gwindows_t *);
431 #if defined(__sparcv9)
432 extern int Plwp_getasrs(struct ps_prochandle *, lwpid_t, asrset_t);
433 extern int Plwp_setasrs(struct ps_prochandle *, lwpid_t, const asrset_t);
434 #endif /* __sparcv9 */
436 #endif /* __sparc */
438 #if defined(__i386) || defined(__amd64)
439 extern int Pldt(struct ps_prochandle *, struct ssd *, int);
440 extern int proc_get_ldt(pid_t, struct ssd *, int);
441 #endif /* __i386 || __amd64 */
443 extern int Plwp_getpsinfo(struct ps_prochandle *, lwpid_t, lwpsinfo_t *);
444 extern int Plwp_getpymaster(struct ps_prochandle *, lwpid_t, psinfo_t *);
446 extern int Plwp_stack(struct ps_prochandle *, lwpid_t, stack_t *);
447 extern int Plwp_main_stack(struct ps_prochandle *, lwpid_t, stack_t *);
448 extern int Plwp_alt_stack(struct ps_prochandle *, lwpid_t, stack_t *);
450 /*
451 * LWP iteration interface; iterate over all active LWPs.
452 */
453 typedef int proc_lwp_f(void *, const lwpstatus_t *);
454 extern int Plwp_iter(struct ps_prochandle *, proc_lwp_f *, void *);

```

```

456 /*
457 * LWP iteration interface; iterate over all LWPs, active and zombie.
458 */
459 typedef int proc_lwp_all_f(void *, const lwpstatus_t *, const lwpsinfo_t *);
460 extern int Plwp_iter_all(struct ps_prochandle *, proc_lwp_all_f *, void *);
462 /*
463 * Process iteration interface; iterate over all non-system processes.
464 */
465 typedef int proc_walk_f(psinfo_t *, lwpsinfo_t *, void *);
466 extern int proc_walk(proc_walk_f *, void *, int);
468 #define PR_WALK_PROC 0 /* walk processes only */
469 #define PR_WALK_LWP 1 /* walk all lwps */
471 /*
472 * Determine if an lwp is in a set as returned from proc_arg_xgrab().
473 */
474 extern int proc_lwp_in_set(const char *, lwpid_t);
475 extern int proc_lwp_range_valid(const char *);
477 /*
478 * Symbol table interfaces.
479 */
481 /*
482 * Pseudo-names passed to Plookup_by_name() for well-known load objects.
483 * NOTE: It is required that PR_OBJ_EXEC and PR_OBJ_LDSDO exactly match
484 * the definitions of PS_OBJ_EXEC and PS_OBJ_LDSDO from <proc_service.h>.
485 */
486 #define PR_OBJ_EXEC ((const char *)0) /* search the executable file */
487 #define PR_OBJ_LDSDO ((const char *)1) /* search ld.so.1 */
488 #define PR_OBJ_EVERY ((const char *)-1) /* search every load object */
490 /*
491 * Special Lmid_t passed to Plookup_by_lmid() to search all link maps. The
492 * special values LM_ID_BASE and LM_ID_LDSDO from <link.h> may also be used.
493 * If PR_OBJ_EXEC is used as the object name, the lmid must be PR_LMID_EVERY
494 * or LM_ID_BASE in order to return a match. If PR_OBJ_LDSDO is used as the
495 * object name, the lmid must be PR_LMID_EVERY or LM_ID_LDSDO to return a match.
496 */
497 #define PR_LMID_EVERY ((Lmid_t)-1UL) /* search every link map */
499 /*
500 * 'object_name' is the name of a load object obtained from an
501 * iteration over the process's address space mappings (Pmapping_iter),
502 * or an iteration over the process's mapped objects (Pobject_iter),
503 * or else it is one of the special PR_OBJ_* values above.
504 */
505 extern int Plookup_by_name(struct ps_prochandle *,
506 const char *, const char *, GElf_Sym *);
508 extern int Plookup_by_addr(struct ps_prochandle *,
509 uintptr_t, char *, size_t, GElf_Sym *);
511 typedef struct prsyminfo {
512 const char *prs_object; /* object name */
513 const char *prs_name; /* symbol name */
514 lmid_t prs_lmid; /* link map id */
515 uint_t prs_id; /* symbol id */
516 uint_t prs_table; /* symbol table id */
517 } prsyminfo_t;
519 extern int Pxlookup_by_name(struct ps_prochandle *,
520 Lmid_t, const char *, const char *, GElf_Sym *, prsyminfo_t *);

```



```

522 extern int Pxlookup_by_addr(struct ps_prochandle *,
523     uintptr_t, char *, size_t, GElf_Sym *, prsyminfo_t *);
524 extern int Pxlookup_by_addr_resolved(struct ps_prochandle *,
525     uintptr_t, char *, size_t, GElf_Sym *, prsyminfo_t *);

527 typedef int proc_map_f(void *, const prmap_t *, const char *);

529 extern int Pmapping_iter(struct ps_prochandle *, proc_map_f *, void *);
530 extern int Pmapping_iter_resolved(struct ps_prochandle *, proc_map_f *, void *);
531 extern int Pobject_iter(struct ps_prochandle *, proc_map_f *, void *);
532 extern int Pobject_iter_resolved(struct ps_prochandle *, proc_map_f *, void *);

534 extern const prmap_t *Paddr_to_map(struct ps_prochandle *, uintptr_t);
535 extern const prmap_t *Paddr_to_text_map(struct ps_prochandle *, uintptr_t);
536 extern const prmap_t *Pname_to_map(struct ps_prochandle *, const char *);
537 extern const prmap_t *Plmid_to_map(struct ps_prochandle *,
538     Lmid_t, const char *);

540 extern const rd_loadobj_t *Paddr_to_loadobj(struct ps_prochandle *, uintptr_t);
541 extern const rd_loadobj_t *Pname_to_loadobj(struct ps_prochandle *,
542     const char *);
543 extern const rd_loadobj_t *Plmid_to_loadobj(struct ps_prochandle *,
544     Lmid_t, const char *);

546 extern ctf_file_t *Paddr_to_ctf(struct ps_prochandle *, uintptr_t);
547 extern ctf_file_t *Pname_to_ctf(struct ps_prochandle *, const char *);

549 extern char *Pplatform(struct ps_prochandle *, char *, size_t);
550 extern int Puname(struct ps_prochandle *, struct utsname *);
551 extern char *Pzonename(struct ps_prochandle *, char *, size_t);
552 extern char *Pfindobj(struct ps_prochandle *, const char *, char *, size_t);

554 extern char *Pexename(struct ps_prochandle *, char *, size_t);
555 extern char *Pobjname(struct ps_prochandle *, uintptr_t, char *, size_t);
556 extern char *Pobjname_resolved(struct ps_prochandle *, uintptr_t, char *,
557     size_t);
558 extern int Plmid(struct ps_prochandle *, uintptr_t, Lmid_t *);

560 typedef int proc_env_f(void *, struct ps_prochandle *, uintptr_t, const char *);
561 extern int Penv_iter(struct ps_prochandle *, proc_env_f *, void *);
562 extern char *Pgetenv(struct ps_prochandle *, const char *, char *, size_t);
563 extern long Pgetauxval(struct ps_prochandle *, int);
564 extern const auxv_t *Pgetauxvec(struct ps_prochandle *);

566 extern void Pset_procfs_path(const char *);

568 /*
569  * Symbol table iteration interface. The special lmid constants LM_ID_BASE,
570  * LM_ID_LDSO, and PR_LMID_EVERY may be used with Psymbol_iter_by_lmid.
571  */
572 typedef int proc_sym_f(void *, const GElf_Sym *, const char *);
573 typedef int proc_xsym_f(void *, const GElf_Sym *, const char *,
574     const prsyminfo_t *);

576 extern int Psymbol_iter(struct ps_prochandle *,
577     const char *, int, int, proc_sym_f *, void *);
578 extern int Psymbol_iter_by_addr(struct ps_prochandle *,
579     const char *, int, int, proc_sym_f *, void *);
580 extern int Psymbol_iter_by_name(struct ps_prochandle *,
581     const char *, int, int, proc_sym_f *, void *);

583 extern int Psymbol_iter_by_lmid(struct ps_prochandle *,
584     Lmid_t, const char *, int, int, proc_sym_f *, void *);

586 extern int Pxsymbol_iter(struct ps_prochandle *,

```

```

587     Lmid_t, const char *, int, int, proc_xsym_f *, void *);

589 /*
590  * 'which' selects which symbol table and can be one of the following.
591  */
592 #define PR_SYMTAB      1
593 #define PR_DYNSYM     2
594 /*
595  * 'type' selects the symbols of interest by binding and type. It is a bit-
596  * mask of one or more of the following flags, whose order MUST match the
597  * order of STB and STT constants in <sys/elf.h>.
598  */
599 #define BIND_LOCAL     0x0001
600 #define BIND_GLOBAL    0x0002
601 #define BIND_WEAK     0x0004
602 #define BIND_ANY (BIND_LOCAL|BIND_GLOBAL|BIND_WEAK)
603 #define TYPE_NOTYPE   0x0100
604 #define TYPE_OBJECT   0x0200
605 #define TYPE_FUNC     0x0400
606 #define TYPE_SECTION  0x0800
607 #define TYPE_FILE     0x1000
608 #define TYPE_ANY (TYPE_NOTYPE|TYPE_OBJECT|TYPE_FUNC|TYPE_SECTION|TYPE_FILE)

610 /*
611  * This returns the rtdb agent handle for the process.
612  * The handle will become invalid at the next successful exec() and
613  * must not be used beyond that point (see Preset_maps(), below).
614  */
615 extern rd_agent_t *Prd_agent(struct ps_prochandle *);

617 /*
618  * This should be called when an RD_DLACTION event with the
619  * RD_CONSISTENT state occurs via librtld_db's event mechanism.
620  * This makes libproc's address space mappings and symbol tables current.
621  * The variant Pupdate_syms() can be used to preload all symbol tables as well.
622  */
623 extern void Pupdate_maps(struct ps_prochandle *);
624 extern void Pupdate_syms(struct ps_prochandle *);

626 /*
627  * This must be called after the victim process performs a successful
628  * exec() if any of the symbol table interface functions have been called
629  * prior to that point. This is essential because an exec() invalidates
630  * all previous symbol table and address space mapping information.
631  * It is always safe to call, but if it is called other than after an
632  * exec() by the victim process it just causes unnecessary overhead.
633  *
634  * The rtdb agent handle obtained from a previous call to Prd_agent() is
635  * made invalid by Preset_maps() and Prd_agent() must be called again to get
636  * the new handle.
637  */
638 extern void Preset_maps(struct ps_prochandle *);

640 /*
641  * Given an address, Ppltdest() determines if this is part of a PLT, and if
642  * so returns a pointer to the symbol name that will be used for resolution.
643  * If the specified address is not part of a PLT, the function returns NULL.
644  */
645 extern const char *Ppltdest(struct ps_prochandle *, uintptr_t);

647 /*
648  * See comments for Pissyscall(), in Pisadep.h
649  */
650 extern int Pissyscall_prev(struct ps_prochandle *, uintptr_t, uintptr_t *);

652 /*

```

```

653 * Stack frame iteration interface.
654 */
655 typedef int proc_stack_f(void *, pgregset_t, uint_t, const long *);

657 extern int Pstack_iter(struct ps_prochandle *,
658     const pgregset_t, proc_stack_f *, void *);

660 /*
661 * The following functions define a set of passive interfaces: libproc provides
662 * default, empty definitions that are called internally. If a client wishes
663 * to override these definitions, it can simply provide its own version with
664 * the same signature that interposes on the libproc definition.
665 *
666 * If the client program wishes to report additional error information, it
667 * can provide its own version of Perror_printf.
668 *
669 * If the client program wishes to receive a callback after Pcreate forks
670 * but before it execs, it can provide its own version of Pcreate_callback.
671 */
672 extern void Perror_printf(struct ps_prochandle *P, const char *format, ...);
673 extern void Pcreate_callback(struct ps_prochandle *);

675 /*
676 * Remove unprintable characters from psinfo.pr_psargs and replace with
677 * whitespace characters so it is safe for printing.
678 */
679 extern void proc_unctrl_psinfo(psinfo_t *);

681 /*
682 * Utility functions for processing arguments which should be /proc files,
683 * pids, and/or core files. The returned error code can be passed to
684 * Pgrab_error() in order to convert it to an error string.
685 */
686 #define PR_ARG_PIDS    0x1    /* Allow pid and /proc file arguments */
687 #define PR_ARG_CORES  0x2    /* Allow core file arguments */

689 #define PR_ARG_ANY    (PR_ARG_PIDS | PR_ARG_CORES)

691 extern struct ps_prochandle *proc_arg_grab(const char *, int, int, int *);
692 extern struct ps_prochandle *proc_arg_xgrab(const char *, const char *, int,
693     int, int *, const char **);
694 extern pid_t proc_arg_psinfo(const char *, int, psinfo_t *, int *);
695 extern pid_t proc_arg_xpsinfo(const char *, int, psinfo_t *, int *,
696     const char **);

698 /*
699 * Utility functions for obtaining information via /proc without actually
700 * performing a Pcreate() or Pgrab():
701 */
702 extern int proc_get_auxv(pid_t, auxv_t *, int);
703 extern int proc_get_cred(pid_t, prcred_t *, int);
704 extern prpriv_t *proc_get_priv(pid_t);
705 extern void proc_free_priv(prpriv_t *);
706 extern int proc_get_psinfo(pid_t, psinfo_t *);
707 extern int proc_get_status(pid_t, pstatus_t *);
708 extern int proc_get_secflags(pid_t, prsecflags_t **);
709 #endif /* ! codereview */

711 /*
712 * Utility functions for debugging tools to convert numeric fault,
713 * signal, and system call numbers to symbolic names:
714 */
715 #define FLT2STR_MAX 32 /* max. string length of faults (like SIG2STR_MAX) */
716 #define SYS2STR_MAX 32 /* max. string length of syscalls (like SIG2STR_MAX) */

718 extern char *proc_fltname(int, char *, size_t);

```

```

719 extern char *proc_signame(int, char *, size_t);
720 extern char *proc_sysname(int, char *, size_t);

722 /*
723 * Utility functions for debugging tools to convert fault, signal, and system
724 * call names back to the numeric constants:
725 */
726 extern int proc_str2flt(const char *, int *);
727 extern int proc_str2sig(const char *, int *);
728 extern int proc_str2sys(const char *, int *);

730 /*
731 * Utility functions for debugging tools to convert a fault, signal or system
732 * call set to a string representation (e.g. "BUS,SEGV" or "open,close,read").
733 */
734 #define PRSIGBUFSZ    1024 /* buffer size for proc_sigset2str() */

736 extern char *procfltset2str(const fltset_t *, const char *, int,
737     char *, size_t);
738 extern char *procsigset2str(const sigset_t *, const char *, int,
739     char *, size_t);
740 extern char *procsysset2str(const sysset_t *, const char *, int,
741     char *, size_t);

743 extern int Pgcrcore(struct ps_prochandle *, const char *, core_content_t);
744 extern int Pfgcore(struct ps_prochandle *, int, core_content_t);
745 extern core_content_t Pcontent(struct ps_prochandle *);

747 /*
748 * Utility functions for debugging tools to convert a string representation of
749 * a fault, signal or system call set back to the numeric value of the
750 * corresponding set type.
751 */
752 extern char *proc_str2fltset(const char *, const char *, int, fltset_t *);
753 extern char *proc_str2sigset(const char *, const char *, int, sigset_t *);
754 extern char *proc_str2sysset(const char *, const char *, int, sysset_t *);

756 /*
757 * Utility functions for converting between strings and core_content_t.
758 */
759 #define PRCONTENTBUFSZ 80 /* buffer size for proc_content2str() */

761 extern int proc_str2content(const char *, core_content_t *);
762 extern int proc_content2str(core_content_t, char *, size_t);

764 /*
765 * Utility functions for buffering output to stdout, stderr while
766 * process is grabbed. Prevents deadlocks due to pfiles 'pgrep xterm'
767 * and other variants.
768 */
769 extern int proc_initstdio(void);
770 extern int proc_flushstdio(void);
771 extern int proc_finistdio(void);

773 /*
774 * Iterate over all open files.
775 */
776 typedef int proc_fdinfo_f(void *, prfdinfo_t *);
777 extern int Pfdinfo_iter(struct ps_prochandle *, proc_fdinfo_f *, void *);

779 #ifdef __cplusplus
780 }
781 #endif

783 #endif /* _LIBPROC_H */

```

```

*****
5709 Thu Jun 30 22:02:24 2016
new/usr/src/lib/libproc/common/mapfile-vers
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
_____unchanged_portion_omitted_____

68 SYMBOL_VERSION SUNWprivate_1.1 {
69   global:
70     Lalt_stack;
71     Lclearfault;
72     Lclearsig;
73     Lctlfld;
74     Ldstop;
75     Lfree;
76     Lgetareg;
77     Lgrab;
78     Lgrab_error;
79     _libproc_debug;
80     Lmain_stack;
81     Lprochandle;
82     Lpsinfo;
83     Lputareg;
84     Lsetrun;
85     Lstack;
86     Lstate;
87     Lstatus;
88     Lstop;
89     Lsync;
90     Lwait;
91     Lxecbkpt;
92     Lxecwapt;
93     Paddr_to_ctf;
94     Paddr_to_loadobj;
95     Paddr_to_map;
96     Paddr_to_text_map;
97     Pasfd;
98     Pclearfault;
99     Pclearsig;
100    Pcontent;
101    Pcreate;
102    Pcreate_agent;
103    Pcreate_callback;
104    Pcreate_error;
105    Pcred;
106    Pctlfld;
107    Pdelbkpt;
108    Pdelwapt;
109    Pdestroy_agent;
110    Pdstop;
111    Penv_iter;
112    Perror_printf;
113    Pexecname;
114    Pfault;
115    Pfgcore;
116    Pfggrab_core;
117    Pfree;
118    Pgc core;
119    Pgetareg;
120    Pgetauxval;
121    Pgetauxvec;
122    Pgetenv;
123    Pgrab;

```

```

124    Pgrab_core;
125    Pgrab_error;
126    Pgrab_file;
127    Pgrab_ops;
128    Pisprocdir;
129    Pissyscall;
130    Pissyscall_prev;
131    Plmid;
132    Plmid_to_ctf;
133    Plmid_to_loadobj;
134    Plmid_to_map;
135    Plookup_by_addr;
136    Plookup_by_name;
137    Plwp_alt_stack;
138    Plwp_getfpregs;
139    Plwp_getpsinfo;
140    Plwp_getregs;
141    Plwp_getspymaster;
142    Plwp_iter;
143    Plwp_iter_all;
144    Plwp_main_stack;
145    Plwp_setfpregs;
146    Plwp_setregs;
147    Plwp_stack;
148    Pmapping_iter;
149    Pmapping_iter_resolved;
150    Pname_to_ctf;
151    Pname_to_loadobj;
152    Pname_to_map;
153    Pobject_iter;
154    Pobject_iter_resolved;
155    Pobjname;
156    Pobjname_resolved;
157    Pplatform;
158    Ppltdest;
159    Ppriv;
160    Pprivinfo;
161    Ppriv_free;
162    Ppsinfo;
163    Pputareg;
164    pr_access;
165    pr_close;
166    pr_creat;
167    Prd_agent;
168    pr_door_info;
169    Pread;
170    Pread_string;
171    Prelease;
172    Preopen;
173    Preset_maps;
174    pr_exit;
175    pr_fcntl;
176    pr_fstat;
177    pr_fstat64;
178    pr_fstatvfs;
179    pr_getitimer;
180    pr_getpeername;
181    pr_getpeerucred;
182    pr_getprojid;
183    pr_getrctl;
184    pr_getrlimit;
185    pr_getrlimit64;
186    pr_getsockname;
187    pr_getsockopt;
188    pr_gettaskid;
189    pr_getzoneid;

```

```

190     pr_ioctl;
191     pr_link;
192     pr_llseek;
193     pr_lseek;
194     pr_lstat;
195     pr_lstat64;
196     pr_lwp_exit;
197     pr_memcntl;
198     pr_meminfo;
199     pr_mmap;
200     pr_munmap;
201     proc_arg_grab;
202     proc_arg_psinfo;
203     proc_arg_xgrab;
204     proc_arg_xpsinfo;
205     proc_content2str;
206     proc_finistdio;
207     proc_fltname;
208     proc_fltset2str;
209     proc_flushstdio;
210     proc_free_priv;
211     proc_get_auxv;
212     proc_get_cred;
213     proc_get_priv;
214     proc_get_psinfo;
215     proc_get_secflags;
216 #endif /* ! codereview */
217     proc_get_status;
218     proc_initstdio;
219     proc_lwp_in_set;
220     proc_lwp_range_valid;
221     proc_signame;
222     proc_sigset2str;
223     proc_str2content;
224     proc_str2flt;
225     proc_str2fltset;
226     proc_str2sig;
227     proc_str2sigset;
228     proc_str2sys;
229     proc_str2sysset;
230     proc_sysname;
231     proc_sysset2str;
232     proc_unctrl_psinfo;
233     proc_walk;
234     pr_open;
235     pr_processor_bind;
236     pr_rename;
237     pr_setitimer;
238     pr_setprojctl;
239     pr_setrctl;
240     pr_setrlimit;
241     pr_setrlimit64;
242     pr_settaskid;
243     pr_sigaction;
244     pr_stat;
245     pr_stat64;
246     pr_statvfs;
247     pr_unlink;
248     pr_waitid;
249     pr_zmap;
250     Pset_procfs_path;
251     Psetbkpt;
252     Psetcred;
253     Psetfault;
254     Psetflags;
255     Psetpriv;

```

```

256     Psetrun;
257     Psetsignal;
258     Psetsysentry;
259     Psetsysexit;
260     Psetwapt;
261     Psetzoneid;
262     Psignal;
263     ps_lcontinue;
264     ps_lgetfpregs;
265     ps_lgetregs;
266     ps_lsetfpregs;
267     ps_lsetregs;
268     ps_lstop;
269     ps_pauxv;
270     ps_pbrandname;
271     ps_pcontinue;
272     ps_pdmodel;
273     ps_pread { FLAGS = NODYNSORT }; # Alias of ps_pread
274     ps_pwrite { FLAGS = NODYNSORT }; # Alias of ps_pwrite
275     ps_pglobal_lookup;
276     ps_pglobal_sym;
277     ps_plog;
278     ps_pread;
279     ps_pstop;
280     ps_ptread { FLAGS = NODYNSORT }; # Alias of ps_pread
281     ps_ptwrite { FLAGS = NODYNSORT }; # Alias of ps_pwrite
282     ps_pwrite;
283     Psecflags;
284     Psecflags_free;
285 #endif /* ! codereview */
286     Pstack_iter;
287     Pstate;
288     Pstatus;
289     Pstop;
290     Pstopstatus;
291     Psymbol_iter;
292     Psymbol_iter_by_addr;
293     Psymbol_iter_by_lmid;
294     Psymbol_iter_by_name;
295     Psync;
296     Psyscall;
297     Psysentry;
298     Psysexit;
299     Puname;
300     Punsetflags;
301     Pupdate_maps;
302     Pupdate_syms;
303     Pwait;
304     Pwrite;
305     Pxcreate;
306     Pxecbkpt;
307     Pxecwapt;
308     Pxlookup_by_addr;
309     Pxlookup_by_addr_resolved;
310     Pxlookup_by_name;
311     Pxsymbol_iter;
312     Pzonename;
313     Pzonepath;
314     Pzoneroot;
315     Pfdinfo_iter;

317 $if _x86 && _ELF32
318     Pldt;
319     proc_get_ldt;
320     ps_lgetLDT;
321 $endif

```

```
323 $if _sparc
324     Plwp_getgwindows;
325     Plwp_getxregs;
326     Plwp_setxregs;
327     ps_lgetxregs;
328     ps_lgetxregsize;
329     ps_lsetxregs;

331 $if _ELF64
332     Plwp_getasrs;
333     Plwp_setasrs;
334 $endif
335 $endif

337     local:
338         *;
339 };
```

```

*****
5362 Thu Jun 30 22:02:24 2016
new/usr/src/lib/libproc/common/proc_get_info.c
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
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 /*
26 * Copyright 2015, Joyent, Inc.
27 */

29 #include <stdio.h>
30 #include <stdlib.h>
31 #include <unistd.h>
32 #include <fcntl.h>
33 #include <string.h>
34 #include <limits.h>
35 #include <sys/secflags.h>
36 #endif /* ! codereview */

38 #include "Pcontrol.h"

40 /*
41  * These several routines simply get the indicated /proc structures
42  * for a process identified by process ID. They are convenience
43  * functions for one-time operations. They do the mechanics of
44  * open() / read() / close() of the necessary /proc files so the
45  * caller's code can look relatively less cluttered.
46  */

48 /*
49  * 'ngroups' is the number of supplementary group entries allocated in
50  * the caller's cred structure. It should equal zero or one unless extra
51  * space has been allocated for the group list by the caller, like this:
52  *   credp = malloc(sizeof (prcred_t) + (ngroups - 1) * sizeof (gid_t));
53  */
54 int
55 proc_get_cred(pid_t pid, prcred_t *credp, int ngroups)
56 {
57     char fname[PATH_MAX];
58     int fd;

```

```

59     int rv = -1;
60     ssize_t minsize = sizeof (*credp) - sizeof (gid_t);
61     size_t size = minsize + ngroups * sizeof (gid_t);

63     (void) snprintf(fname, sizeof (fname), "%s/%d/cred",
64                    procfs_path, (int)pid);
65     if ((fd = open(fname, O_RDONLY)) >= 0) {
66         if (read(fd, credp, size) >= minsize)
67             rv = 0;
68         (void) close(fd);
69     }
70     return (rv);
71 }

73 int
74 proc_get_secflags(pid_t pid, prsecflags_t **psf)
75 {
76     char fname[PATH_MAX];
77     int fd;
78     int rv = -1;

80     if ((*psf = calloc(1, sizeof (prsecflags_t))) == NULL)
81         return (-1);

83     (void) snprintf(fname, sizeof (fname), "%s/%d/secflags",
84                    procfs_path, (int)pid);
85     if ((fd = open(fname, O_RDONLY)) >= 0) {
86         if (read(fd, *psf, sizeof (prsecflags_t)) ==
87             sizeof (prsecflags_t))
88             rv = 0;
89         (void) close(fd);
90     }
91     return (rv);
92 }

94 #endif /* ! codereview */
95 void
96 proc_free_priv(prpriv_t *priv)
97 {
98     free(priv);
99 }

101 /*
102  * Malloc and return a properly sized structure.
103  */
104 prpriv_t *
105 proc_get_priv(pid_t pid)
106 {
107     char fname[PATH_MAX];
108     int fd;
109     struct stat statb;
110     prpriv_t *rv = NULL;

112     (void) snprintf(fname, sizeof (fname), "%s/%d/priv",
113                    procfs_path, (int)pid);
114     if ((fd = open(fname, O_RDONLY)) >= 0) {
115         if (fstat(fd, &statb) != 0 ||
116             (rv = malloc(statb.st_size)) == NULL ||
117             read(fd, rv, statb.st_size) != statb.st_size) {
118             free(rv);
119             rv = NULL;
120         }
121         (void) close(fd);
122     }
123     return (rv);
124 }

```

```

126 #if defined(__i386) || defined(__amd64)
127 /*
128  * Fill in a pointer to a process LDT structure.
129  * The caller provides a buffer of size 'nldt * sizeof (struct ssd)';
130  * If pldt == NULL or nldt == 0, we return the number of existing LDT entries.
131  * Otherwise we return the actual number of LDT entries fetched (<= nldt).
132  */
133 int
134 proc_get_ldt(pid_t pid, struct ssd *pldt, int nldt)
135 {
136     char fname[PATH_MAX];
137     int fd;
138     struct stat statb;
139     size_t size;
140     ssize_t ssize;
141
142     (void) snprintf(fname, sizeof (fname), "%s/%d/ldt",
143                    procfs_path, (int)pid);
144     if ((fd = open(fname, O_RDONLY)) < 0)
145         return (-1);
146
147     if (pldt == NULL || nldt == 0) {
148         nldt = 0;
149         if (fstat(fd, &statb) == 0)
150             nldt = statb.st_size / sizeof (struct ssd);
151         (void) close(fd);
152         return (nldt);
153     }
154
155     size = nldt * sizeof (struct ssd);
156     if ((ssize = read(fd, pldt, size)) < 0)
157         nldt = -1;
158     else
159         nldt = ssize / sizeof (struct ssd);
160
161     (void) close(fd);
162     return (nldt);
163 }
164 #endif /* __i386 || __amd64 */
165
166 int
167 proc_get_psinfo(pid_t pid, psinfo_t *psp)
168 {
169     char fname[PATH_MAX];
170     int fd;
171     int rv = -1;
172
173     (void) snprintf(fname, sizeof (fname), "%s/%d/psinfo",
174                    procfs_path, (int)pid);
175     if ((fd = open(fname, O_RDONLY)) >= 0) {
176         if (read(fd, psp, sizeof (*psp)) == sizeof (*psp))
177             rv = 0;
178         (void) close(fd);
179     }
180     return (rv);
181 }
182
183 int
184 proc_get_status(pid_t pid, pstatus_t *psp)
185 {
186     char fname[PATH_MAX];
187     int fd;
188     int rv = -1;
189
190     (void) snprintf(fname, sizeof (fname), "%s/%d/status",

```

```

191     procfs_path, (int)pid);
192     if ((fd = open(fname, O_RDONLY)) >= 0) {
193         if (read(fd, psp, sizeof (*psp)) == sizeof (*psp))
194             rv = 0;
195         (void) close(fd);
196     }
197     return (rv);
198 }
199
200 /*
201  * Get the process's aux vector.
202  * 'naux' is the number of aux entries in the caller's buffer.
203  * We return the number of aux entries actually fetched from
204  * the process (less than or equal to 'naux') or -1 on failure.
205  */
206 int
207 proc_get_auxv(pid_t pid, auxv_t *pauxv, int naux)
208 {
209     char fname[PATH_MAX];
210     int fd;
211     int rv = -1;
212
213     (void) snprintf(fname, sizeof (fname), "%s/%d/auxv",
214                    procfs_path, (int)pid);
215     if ((fd = open(fname, O_RDONLY)) >= 0) {
216         if (rv = read(fd, pauxv, naux * sizeof (auxv_t)) >= 0)
217             rv /= sizeof (auxv_t);
218         (void) close(fd);
219     }
220     return (rv);
221 }

```

```

*****
98571 Thu Jun 30 22:02:25 2016
new/usr/src/lib/librestart/common/librestart.c
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
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 /*
23  * Copyright (c) 2004, 2010, Oracle and/or its affiliates. All rights reserved.
24  * Copyright (c) 2013, Joyent, Inc. All rights reserved.
25 */
26
27 #include <libintl.h>
28 #include <librestart.h>
29 #include <librestart_priv.h>
30 #include <libscf.h>
31 #include <libscf_priv.h>
32
33 #include <assert.h>
34 #include <ctype.h>
35 #include <dlfcn.h>
36 #include <errno.h>
37 #include <exec_attr.h>
38 #include <grp.h>
39 #include <libsysevent.h>
40 #include <libuutil.h>
41 #include <limits.h>
42 #include <link.h>
43 #include <malloc.h>
44 #include <pool.h>
45 #include <priv.h>
46 #include <project.h>
47 #include <pthread.h>
48 #include <pwd.h>
49 #include <secdb.h>
50 #include <signal.h>
51 #include <stdlib.h>
52 #include <string.h>
53 #include <syslog.h>
54 #include <sys/corect.h>
55 #include <sys/machelf.h>
56 #include <sys/secflags.h>
57 #endif /* ! codereview */
58 #include <sys/task.h>

```

```

59 #include <sys/types.h>
60 #include <time.h>
61 #include <unistd.h>
62 #include <ucontext.h>
63
64 #define min(a, b)          ((a) > (b) ? (b) : (a))
65
66 #define MKW_TRUE          ":true"
67 #define MKW_KILL         ":kill"
68 #define MKW_KILL_PROC   ":kill_process"
69
70 #define ALLOCFAIL        ((char *)"Allocation failure.")
71 #define RCBROKEN        ((char *)"Repository connection broken.")
72
73 #define MAX_COMMIT_RETRIES          10
74 #define MAX_COMMIT_RETRY_INT       (5 * 1000000) /* 5 seconds */
75 #define INITIAL_COMMIT_RETRY_INT   (10000) /* 1/100th second */
76
77 /*
78  * bad_fail() catches bugs in this and lower layers by reporting supposedly
79  * impossible function failures. The NDEBUG case keeps the strings out of the
80  * library but still calls abort() so we can root-cause from the core dump.
81  */
82 #ifndef NDEBUG
83 #define bad_fail(func, err) {
84     (void) fprintf(stderr,
85     "At %s:%d, %s() failed with unexpected error %d. Aborting.\n",
86     __FILE__, __LINE__, (func), (err));
87     abort();
88 }
89 #else
90 #define bad_fail(func, err) abort()
91 #endif
92
93 struct restart_event_handle {
94     char *reh_restarter_name;
95     char *reh_delegate_channel_name;
96     evchan_t *reh_delegate_channel;
97     char *reh_delegate_subscriber_id;
98     char *reh_master_channel_name;
99     evchan_t *reh_master_channel;
100     char *reh_master_subscriber_id;
101     int (*reh_handler)(restart_event_t *);
102 };
103
104 struct restart_event {
105     sysevent_t *re_sysevent;
106     restart_event_type_t re_type;
107     char *re_instance_name;
108     restart_event_handle_t re_event_handle;
109     restart_instance_state_t re_state;
110     restart_instance_state_t re_next_state;
111 };
112
113 /*
114  * Long reasons must all parse/read correctly in the following contexts:
115  *
116  * "A service instance transitioned state: %s."
117  * "A service failed: %s."
118  * "Reason: %s."
119  * "The service transitioned state (%s) and ..."
120  *
121  * With the exception of restart_str_none they must also fit the following
122  * moulds:
123  *
124  * "An instance transitioned because %s, and ..."

```



```

125 * "An instance transitioned to <new-state> because %s, and ..."
126 *
127 * Note that whoever is rendering the long message must provide the
128 * terminal punctuation - don't include it here. Similarly, do not
129 * provide an initial capital letter in reason-long.
130 *
131 * The long reason strings are Volatile - within the grammatical constraints
132 * above we may improve them as need be. The intention is that a consumer
133 * may blindly render the string along the lines of the above examples,
134 * but has no other guarantees as to the exact wording. Long reasons
135 * are localized.
136 *
137 * We define revisions of the set of short reason strings in use. Within
138 * a given revision, all short reasons are Committed. Consumers must check
139 * the revision in use before relying on the semantics of the short reason
140 * codes - if the version exceeds that which they are familiar with they should
141 * fail gracefully. Having checked for version compatability, a consumer
142 * is assured that
143 *
144 *     "short_reason_A iff semantic_A", provided:
145 *
146 *         . the restarter uses this short reason code at all,
147 *         . the short reason is not "none" (which a restarter could
148 *           specify for any transition semantics)
149 *
150 * To split/refine such a Committed semantic_A into further cases,
151 * we are required to bump the revision number. This should be an
152 * infrequent occurrence. If you bump the revision number you may
153 * need to make corresponding changes in any source that calls
154 * restarter_str_version (e.g., FMA event generation).
155 *
156 * To add additional reasons to the set you must also bump the version
157 * number.
158 */

160 /*
161 * The following describes revision 0 of the set of transition reasons.
162 * Read the preceding block comment before making any changes.
163 */
164 static const struct restarter_state_transition_reason restarter_str[] = {
165     /*
166      * Any transition for which the restarter has not provided a reason.
167      */
168     {
169         restarter_str_none,
170         "none",
171         "the restarter gave no reason"
172     },
173
174     /*
175      * A transition to maintenance state due to a
176      * 'svcadm mark maintenance <fmri>'. *Not* used if the libscf
177      * interface smf_maintain_instance(3SCF) is used to request maintenance.
178      */
179     {
180         restarter_str_administrative_request,
181         "administrative_request",
182         "maintenance was requested by an administrator"
183     },
184
185     /*
186      * A transition to maintenance state if a repository inconsistency
187      * exists when the service/instance state is first read by startd
188      * into the graph engine (this can also happen during startd restart).
189      */
190     {

```

```

191         restarter_str_bad_repo_state,
192         "bad_repo_state",
193         "an SMF repository inconsistency exists"
194     },
195
196     /*
197      * A transition 'maintenance -> uninitialized' resulting always
198      * from 'svcadm clear <fmri>'. *Not* used if the libscf interface
199      * smf_restore_instance(3SCF) is used.
200      */
201     {
202         restarter_str_clear_request,
203         "clear_request",
204         "maintenance clear was requested by an administrator"
205     },
206
207     /*
208      * A transition 'online -> offline' due to a process core dump.
209      */
210     {
211         restarter_str_ct_ev_core,
212         "ct_ev_core",
213         "a process dumped core"
214     },
215
216     /*
217      * A transition 'online -> offline' due to an empty process contract,
218      * i.e., the last process in a contract type service has exited.
219      */
220     {
221         restarter_str_ct_ev_exit,
222         "ct_ev_exit",
223         "all processes in the service have exited"
224     },
225
226     /*
227      * A transition 'online -> offline' due to a hardware error.
228      */
229     {
230         restarter_str_ct_ev_hwerr,
231         "ct_ev_hwerr",
232         "a process was killed due to uncorrectable hardware error"
233     },
234
235     /*
236      * A transition 'online -> offline' due to a process in the service
237      * having received a fatal signal originating from outside the
238      * service process contract.
239      */
240     {
241         restarter_str_ct_ev_signal,
242         "ct_ev_signal",
243         "a process received a fatal signal from outside the service"
244     },
245
246     /*
247      * A transition 'offline -> online' when all dependencies for the
248      * service have been met.
249      */
250     {
251         restarter_str_dependencies_satisfied,
252         "dependencies_satisfied",
253         "all dependencies have been satisfied"
254     },
255
256     /*

```

```

257  * A transition 'online -> offline' because some dependency for the
258  * service is no-longer met.
259  */
260  {
261      restarter_str_dependency_activity,
262      "dependency_activity",
263      "a dependency activity required a stop"
264  },
265
266  /*
267  * A transition to maintenance state due to a cycle in the
268  * service dependencies.
269  */
270  {
271      restarter_str_dependency_cycle,
272      "dependency_cycle",
273      "a dependency cycle exists"
274  },
275
276  /*
277  * A transition 'online -> offline -> disabled' due to a
278  * 'svcadm disable [-t] <fmri>' or smf_disable_instance(3SCF) call.
279  */
280  {
281      restarter_str_disable_request,
282      "disable_request",
283      "a disable was requested"
284  },
285
286  /*
287  * A transition 'disabled -> offline' due to a
288  * 'svcadm enable [-t] <fmri>' or smf_enable_instance(3SCF) call.
289  */
290  {
291      restarter_str_enable_request,
292      "enable_request",
293      "an enable was requested"
294  },
295
296  /*
297  * A transition to maintenance state when a method fails
298  * repeatedly for a retryable reason.
299  */
300  {
301      restarter_str_fault_threshold_reached,
302      "fault_threshold_reached",
303      "a method is failing in a retryable manner but too often"
304  },
305
306  /*
307  * A transition to uninitialized state when startd reads the service
308  * configuration and inserts it into the graph engine.
309  */
310  {
311      restarter_str_insert_in_graph,
312      "insert_in_graph",
313      "the instance was inserted in the graph"
314  },
315
316  /*
317  * A transition to maintenance state due to an invalid dependency
318  * declared for the service.
319  */
320  {
321      restarter_str_invalid_dependency,
322      "invalid_dependency",

```

```

323      "a service has an invalid dependency"
324  },
325
326  /*
327  * A transition to maintenance state because the service-declared
328  * restarter is invalid.
329  */
330  {
331      restarter_str_invalid_restarter,
332      "invalid_restarter",
333      "the service restarter is invalid"
334  },
335
336  /*
337  * A transition to maintenance state because a restarter method
338  * exited with one of SMF_EXIT_ERR_CONFIG, SMF_EXIT_ERR_NOSMF,
339  * SMF_EXIT_ERR_PERM, or SMF_EXIT_ERR_FATAL.
340  */
341  {
342      restarter_str_method_failed,
343      "method_failed",
344      "a start, stop or refresh method failed"
345  },
346
347  /*
348  * A transition 'uninitialized -> {disabled|offline}' after
349  * "insert_in_graph" to match the state configured in the
350  * repository.
351  */
352  {
353      restarter_str_per_configuration,
354      "per_configuration",
355      "the SMF repository configuration specifies this state"
356  },
357
358  /*
359  * Refresh requested - no state change.
360  */
361  {
362      restarter_str_refresh,
363      NULL,
364      "a refresh was requested (no change of state)"
365  },
366
367  /*
368  * A transition 'online -> offline -> online' due to a
369  * 'svcadm restart <fmri>' or equivalent libscf API call.
370  * Both the 'online -> offline' and 'offline -> online' transtions
371  * specify this reason.
372  */
373  {
374      restarter_str_restart_request,
375      "restart_request",
376      "a restart was requested"
377  },
378
379  /*
380  * A transition to maintenance state because the start method is
381  * being executed successfully but too frequently.
382  */
383  {
384      restarter_str_restarting_too_quickly,
385      "restarting_too_quickly",
386      "the instance is restarting too quickly"
387  },

```

```

389  /*
390  * A transition to maintenance state due a service requesting
391  * 'svcadm mark maintenance <fmri>' or equivalent libscf API call.
392  * A command line 'svcadm mark maintenance <fmri>' does not produce
393  * this reason - it produces administrative_request instead.
394  */
395  {
396      restarter_str_service_request,
397      "service_request",
398      "maintenance was requested by another service"
399  },
401  /*
402  * An instanced inserted into the graph at its existing state
403  * during a startd restart - no state change.
404  */
405  {
406      restarter_str_startd_restart,
407      NULL,
408      "the instance was inserted in the graph due to startd restart"
409  }
410 };

412 uint32_t
413 restarter_str_version(void)
414 {
415     return (RESTARTER_STRING_VERSION);
416 }

418 const char *
419 restarter_get_str_short(restarter_str_t key)
420 {
421     int i;
422     for (i = 0; i < sizeof (restarter_str) /
423          sizeof (struct restarter_state_transition_reason); i++)
424         if (key == restarter_str[i].str_key)
425             return (restarter_str[i].str_short);
426     return (NULL);
427 }

429 const char *
430 restarter_get_str_long(restarter_str_t key)
431 {
432     int i;
433     for (i = 0; i < sizeof (restarter_str) /
434          sizeof (struct restarter_state_transition_reason); i++)
435         if (key == restarter_str[i].str_key)
436             return (dgettext(TEXT_DOMAIN,
437                              restarter_str[i].str_long));
438     return (NULL);
439 }

441 /*
442 * A static no memory error message mc_error_t structure
443 * to be used in cases when memory errors are to be returned
444 * This avoids the need to attempt to allocate memory for the
445 * message, therefore getting into a cycle of no memory failures.
446 */
447 mc_error_t mc_nomem_err = {
448     0, ENOMEM, sizeof ("Out of memory") - 1, "Out of memory"
449 };

451 static const char * const allocfail = "Allocation failure.\n";
452 static const char * const rcbroken = "Repository connection broken.\n";

454 static int method_context_safety = 0; /* Can safely call pools/projects. */

```

```

456 int ndebug = 1;

458 /* PRINTFLIKE3 */
459 static mc_error_t *
460 mc_error_create(mc_error_t *e, int type, const char *format, ...)
461 {
462     mc_error_t *le;
463     va_list args;
464     int size;

466     /*
467     * If the type is ENOMEM and format is NULL, then
468     * go ahead and return the default nomem error.
469     * Otherwise, attempt to allocate the memory and if
470     * that fails then there is no reason to continue.
471     */
472     if (type == ENOMEM && format == NULL)
473         return (&mc_nomem_err);

475     if (e == NULL && (le = malloc(sizeof (mc_error_t))) == NULL)
476         return (&mc_nomem_err);
477     else
478         le = e;

480     le->type = type;
481     le->destroy = 1;
482     va_start(args, format);
483     size = vsnprintf(NULL, 0, format, args) + 1;
484     if (size >= RESTARTER_ERRMSGSZ) {
485         if ((le = realloc(e, sizeof (mc_error_t) +
486                        (size - RESTARTER_ERRMSGSZ))) == NULL) {
487             size = RESTARTER_ERRMSGSZ - 1;
488             le = e;
489         }
490     }

492     le->size = size;
493     (void) vsnprintf(le->msg, le->size, format, args);
494     va_end(args);

496     return (le);
497 }

499 void
500 restarter_mc_error_destroy(mc_error_t *mc_err)
501 {
502     if (mc_err == NULL)
503         return;

505     /*
506     * If the error messages was allocated then free.
507     */
508     if (mc_err->destroy) {
509         free(mc_err);
510     }
511 }

513 static void
514 free_restarter_event_handle(struct restarter_event_handle *h)
515 {
516     if (h == NULL)
517         return;

519     /*
520     * Just free the memory -- don't unbind the sysevent handle,

```

```

521  * as otherwise events may be lost if this is just a restarter
522  * restart.
523  */

525  if (h->reh_restarter_name != NULL)
526      free(h->reh_restarter_name);
527  if (h->reh_delegate_channel_name != NULL)
528      free(h->reh_delegate_channel_name);
529  if (h->reh_delegate_subscriber_id != NULL)
530      free(h->reh_delegate_subscriber_id);
531  if (h->reh_master_channel_name != NULL)
532      free(h->reh_master_channel_name);
533  if (h->reh_master_subscriber_id != NULL)
534      free(h->reh_master_subscriber_id);

536  free(h);
537 }

539 char *
540 _restarter_get_channel_name(const char *fmri, int type)
541 {
542     char *name;
543     char *chan_name = malloc(MAX_CHNAME_LEN);
544     char prefix_name[3];
545     int i;

547     if (chan_name == NULL)
548         return (NULL);

550     if (type == RESTARTER_CHANNEL_DELEGATE)
551         (void) strcpy(prefix_name, "d_");
552     else if (type == RESTARTER_CHANNEL_MASTER)
553         (void) strcpy(prefix_name, "m_");
554     else {
555         free(chan_name);
556         return (NULL);
557     }

559     /*
560     * Create a unique name
561     *
562     * Use the entire name, using a replacement of the /
563     * characters to get a better name.
564     *
565     * Remove the svc:/ from the beginning as this really
566     * isn't going to provide any uniqueness...
567     *
568     * An fmri name greater than MAX_CHNAME_LEN is going
569     * to be rejected as too long for the chan_name below
570     * in the snprintf call.
571     */
572     if ((name = strdup(strchr(fmri, '/') + 1)) == NULL) {
573         free(chan_name);
574         return (NULL);
575     }
576     i = 0;
577     while (name[i]) {
578         if (name[i] == '/') {
579             name[i] = '_';
580         }

582         i++;
583     }

585     /*
586     * Should check for [a-z],[A-Z],[0-9],.,_,-,:

```

```

587     */

589     if (snprintf(chan_name, MAX_CHNAME_LEN, "com.sun.scf:%s%s",
590         prefix_name, name) > MAX_CHNAME_LEN) {
591         free(chan_name);
592         chan_name = NULL;
593     }

595     free(name);
596     return (chan_name);
597 }

599 int
600 cb(sysevent_t *syse, void *cookie)
601 {
602     restarter_event_handle_t *h = (restarter_event_handle_t *)cookie;
603     restarter_event_t *e;
604     nvlist_t *attr_list = NULL;
605     int ret = 0;

607     e = uu_zalloc(sizeof (restarter_event_t));
608     if (e == NULL)
609         uu_die(allocfail);
610     e->re_event_handle = h;
611     e->re_sysevent = syse;

613     if (sysevent_get_attr_list(syse, &attr_list) != 0)
614         uu_die(allocfail);

616     if ((nvlist_lookup_uint32(attr_list, RESTARTER_NAME_TYPE,
617         &(e->re_type)) != 0) ||
618         (nvlist_lookup_string(attr_list,
619             RESTARTER_NAME_INSTANCE, &(e->re_instance_name)) != 0)) {
620         uu_warn("%s: Can't decode nvlist for event %p\n",
621             h->reh_restarter_name, (void *)syse);

623         ret = 0;
624     } else {
625         ret = h->reh_handler(e);
626     }

628     uu_free(e);
629     nvlist_free(attr_list);
630     return (ret);
631 }

633 /*
634 * restarter_bind_handle(uint32_t, char *, int (*)(restarter_event_t *), int,
635 * restarter_event_handle_t **)
636 *
637 * Bind to a delegated restarter event channel.
638 * Each delegated restarter gets its own channel for resource management.
639 *
640 * Returns 0 on success or
641 * ENOTSUP version mismatch
642 * EINVAL restarter_name or event_handle is NULL
643 * ENOMEM out of memory, too many channels, or too many subscriptions
644 * EBUSY sysevent_evcbind() could not establish binding
645 * EFAULT internal sysevent_evcbind()/sysevent_evcbind() error
646 * EMFILE out of file descriptors
647 * EPERM insufficient privilege for sysevent_evcbind()
648 * EEXIST already subscribed
649 */
650 int
651 restarter_bind_handle(uint32_t version, const char *restarter_name,
652     int (*event_handler)(restarter_event_t *), int flags,

```

```

653 restarter_event_handle_t **rehp)
654 {
655     restarter_event_handle_t *h;
656     size_t sz;
657     int err;

659     if (version != RESTARTER_EVENT_VERSION)
660         return (ENOTSUP);

662     if (restarter_name == NULL || event_handler == NULL)
663         return (EINVAL);

665     if (flags & RESTARTER_FLAG_DEBUG)
666         ndebug++;

668     if ((h = uu_zalloc(sizeof (restarter_event_handle_t))) == NULL)
669         return (ENOMEM);

671     h->reh_delegate_subscriber_id = malloc(MAX_SUBID_LEN);
672     h->reh_master_subscriber_id = malloc(MAX_SUBID_LEN);
673     h->reh_restarter_name = strdup(restarter_name);
674     if (h->reh_delegate_subscriber_id == NULL ||
675         h->reh_master_subscriber_id == NULL ||
676         h->reh_restarter_name == NULL) {
677         free_restarter_event_handle(h);
678         return (ENOMEM);
679     }

681     sz = strlcpy(h->reh_delegate_subscriber_id, "del", MAX_SUBID_LEN);
682     assert(sz < MAX_SUBID_LEN);
683     sz = strlcpy(h->reh_master_subscriber_id, "master", MAX_SUBID_LEN);
684     assert(sz < MAX_SUBID_LEN);

686     h->reh_delegate_channel_name =
687         restarter_get_channel_name(restarter_name,
688             RESTARTER_CHANNEL_DELEGATE);
689     h->reh_master_channel_name =
690         restarter_get_channel_name(restarter_name,
691             RESTARTER_CHANNEL_MASTER);

693     if (h->reh_delegate_channel_name == NULL ||
694         h->reh_master_channel_name == NULL) {
695         free_restarter_event_handle(h);
696         return (ENOMEM);
697     }

699     if (sysevent_etc_bind(h->reh_delegate_channel_name,
700         &h->reh_delegate_channel, EVCH_CREAT|EVCH_HOLD_PEND) != 0) {
701         err = errno;
702         assert(err != EINVAL);
703         assert(err != ENOENT);
704         free_restarter_event_handle(h);
705         return (err);
706     }

708     if (sysevent_etc_bind(h->reh_master_channel_name,
709         &h->reh_master_channel, EVCH_CREAT|EVCH_HOLD_PEND) != 0) {
710         err = errno;
711         assert(err != EINVAL);
712         assert(err != ENOENT);
713         free_restarter_event_handle(h);
714         return (err);
715     }

717     h->reh_handler = event_handler;

```

```

719     assert(strlen(restarter_name) <= MAX_CLASS_LEN - 1);
720     assert(strlen(h->reh_delegate_subscriber_id) <= MAX_SUBID_LEN - 1);
721     assert(strlen(h->reh_master_subscriber_id) <= MAX_SUBID_LEN - 1);

723     if (sysevent_etc_subscribe(h->reh_delegate_channel,
724         h->reh_delegate_subscriber_id, EC_ALL, cb, h, EVCH_SUB_KEEP) != 0) {
725         err = errno;
726         assert(err != EINVAL);
727         free_restarter_event_handle(h);
728         return (err);
729     }

731     *rehp = h;
732     return (0);
733 }

735 restarter_event_handle_t *
736 restarter_event_get_handle(restarter_event_t *e)
737 {
738     assert(e != NULL && e->re_event_handle != NULL);
739     return (e->re_event_handle);
740 }

742 restarter_event_type_t
743 restarter_event_get_type(restarter_event_t *e)
744 {
745     assert(e != NULL);
746     return (e->re_type);
747 }

749 ssize_t
750 restarter_event_get_instance(restarter_event_t *e, char *inst, size_t sz)
751 {
752     assert(e != NULL && inst != NULL);
753     return ((ssize_t)strlcpy(inst, e->re_instance_name, sz));
754 }

756 int
757 restarter_event_get_current_states(restarter_event_t *e,
758     restarter_instance_state_t *state, restarter_instance_state_t *next_state)
759 {
760     if (e == NULL)
761         return (-1);
762     *state = e->re_state;
763     *next_state = e->re_next_state;
764     return (0);
765 }

767 /*
768  * restarter_event_publish_retry() is a wrapper around sysevent_etc_publish().
769  * In case, the event cannot be sent at the first attempt (sysevent_etc_publish
770  * returned EAGAIN - sysevent queue full), this function retries a few time
771  * and return ENOSPC if it reaches the retry limit.
772  *
773  * The arguments to this function map the arguments of sysevent_etc_publish().
774  *
775  * On success, return 0. On error, return
776  *
777  * EFAULT - internal sysevent_etc_publish() error
778  * ENOMEM - internal sysevent_etc_publish() error
779  * EBADF - scp is invalid (sysevent_etc_publish() returned EINVAL)
780  * ENOSPC - sysevent queue full (sysevent_etc_publish() returned EAGAIN)
781  */
782 int
783 restarter_event_publish_retry(evchan_t *scp, const char *class,
784     const char *subclass, const char *vendor, const char *pub_name,

```

```

785     nvlist_t *attr_list, uint32_t flags)
786 {
787     int retries, ret;
788     useconds_t retry_int = INITIAL_COMMIT_RETRY_INT;
789
790     for (retries = 0; retries < MAX_COMMIT_RETRIES; retries++) {
791         ret = sysevent_evc_publish(scp, class, subclass, vendor,
792             pub_name, attr_list, flags);
793         if (ret == 0)
794             break;
795
796         switch (ret) {
797             case EAGAIN:
798                 /* Queue is full */
799                 (void) usleep(retry_int);
800
801                 retry_int = min(retry_int * 2, MAX_COMMIT_RETRY_INT);
802                 break;
803
804             case EINVAL:
805                 ret = EBADF;
806                 /* FALLTHROUGH */
807
808             case EFAULT:
809             case ENOMEM:
810                 return (ret);
811
812             case EOVERFLOW:
813             default:
814                 /* internal error - abort */
815                 bad_fail("sysevent_evc_publish", ret);
816         }
817     }
818
819     if (retries == MAX_COMMIT_RETRIES)
820         ret = ENOSPC;
821
822     return (ret);
823 }
824
825 /*
826  * Commit the state, next state, and auxiliary state into the repository.
827  * Let the graph engine know about the state change and error.  On success,
828  * return 0.  On error, return
829  * EPROTO - librestart compiled against different libscf
830  * ENOMEM - out of memory
831  * - repository server out of resources
832  * ENOTACTIVE - repository server not running
833  * ECONNABORTED - repository connection established, but then broken
834  * - unknown libscf error
835  * ENOENT - inst does not exist in the repository
836  * EPERM - insufficient permissions
837  * EACCESS - backend access denied
838  * EROFS - backend is readonly
839  * EFAULT - internal sysevent_evc_publish() error
840  * EBADF - h is invalid (sysevent_evc_publish() returned EINVAL)
841  * ENOSPC - sysevent queue full (sysevent_evc_publish() returned EAGAIN)
842  */
843 int
844 restarter_set_states(restarter_event_handle_t *h, const char *inst,
845     restarter_instance_state_t cur_state,
846     restarter_instance_state_t new_cur_state,
847     restarter_instance_state_t next_state,
848     restarter_instance_state_t new_next_state, restarter_error_t e,
849     restarter_str_t aux)
850 {

```

```

851     nvlist_t *attr;
852     scf_handle_t *scf_h;
853     instance_data_t id;
854     int ret = 0;
855     const char *p = restarter_get_str_short(aux);
856
857     assert(h->reh_master_channel != NULL);
858     assert(h->reh_master_channel_name != NULL);
859     assert(h->reh_master_subscriber_id != NULL);
860
861     if ((scf_h = scf_handle_create(SCF_VERSION)) == NULL) {
862         switch (scf_error()) {
863             case SCF_ERROR_VERSION_MISMATCH:
864                 return (EPROTO);
865
866             case SCF_ERROR_NO_MEMORY:
867                 return (ENOMEM);
868
869             default:
870                 bad_fail("scf_handle_create", scf_error());
871         }
872     }
873
874     if (scf_handle_bind(scf_h) == -1) {
875         scf_handle_destroy(scf_h);
876         switch (scf_error()) {
877             case SCF_ERROR_NO_SERVER:
878                 return (ENOTACTIVE);
879
880             case SCF_ERROR_NO_RESOURCES:
881                 return (ENOMEM);
882
883             case SCF_ERROR_INVALID_ARGUMENT:
884             case SCF_ERROR_IN_USE:
885             default:
886                 bad_fail("scf_handle_bind", scf_error());
887         }
888     }
889
890     if (nvlist_alloc(&attr, NV_UNIQUE_NAME, 0) != 0 ||
891         nvlist_add_int32(attr, RESTARTER_NAME_STATE, new_cur_state) != 0 ||
892         nvlist_add_int32(attr, RESTARTER_NAME_NEXT_STATE, new_next_state)
893         != 0 ||
894         nvlist_add_int32(attr, RESTARTER_NAME_ERROR, e) != 0 ||
895         nvlist_add_string(attr, RESTARTER_NAME_INSTANCE, inst) != 0 ||
896         nvlist_add_int32(attr, RESTARTER_NAME_REASON, aux) != 0) {
897         ret = ENOMEM;
898     } else {
899         id.i_fmri = inst;
900         id.i_state = cur_state;
901         id.i_next_state = next_state;
902
903         ret = _restarter_commit_states(scf_h, &id, new_cur_state,
904             new_next_state, p);
905
906         if (ret == 0) {
907             ret = restarter_event_publish_retry(
908                 h->reh_master_channel, "master", "state_change",
909                 "com.sun", "librestart", attr, EVCH_NOSLEEP);
910         }
911     }
912
913     nvlist_free(attr);
914     (void) scf_handle_unbind(scf_h);
915     scf_handle_destroy(scf_h);

```

```

917     return (ret);
918 }

920 restarter_instance_state_t
921 restarter_string_to_state(char *string)
922 {
923     assert(string != NULL);

925     if (strcmp(string, SCF_STATE_STRING_NONE) == 0)
926         return (RESTARTER_STATE_NONE);
927     else if (strcmp(string, SCF_STATE_STRING_UNINIT) == 0)
928         return (RESTARTER_STATE_UNINIT);
929     else if (strcmp(string, SCF_STATE_STRING_MAINT) == 0)
930         return (RESTARTER_STATE_MAINT);
931     else if (strcmp(string, SCF_STATE_STRING_OFFLINE) == 0)
932         return (RESTARTER_STATE_OFFLINE);
933     else if (strcmp(string, SCF_STATE_STRING_DISABLED) == 0)
934         return (RESTARTER_STATE_DISABLED);
935     else if (strcmp(string, SCF_STATE_STRING_ONLINE) == 0)
936         return (RESTARTER_STATE_ONLINE);
937     else if (strcmp(string, SCF_STATE_STRING_DEGRADED) == 0)
938         return (RESTARTER_STATE_DEGRADED);
939     else {
940         return (RESTARTER_STATE_NONE);
941     }
942 }

944 ssize_t
945 restarter_state_to_string(restarter_instance_state_t state, char *string,
946                          size_t len)
947 {
948     assert(string != NULL);

950     if (state == RESTARTER_STATE_NONE)
951         return ((ssize_t)strcpy(string, SCF_STATE_STRING_NONE, len));
952     else if (state == RESTARTER_STATE_UNINIT)
953         return ((ssize_t)strcpy(string, SCF_STATE_STRING_UNINIT, len));
954     else if (state == RESTARTER_STATE_MAINT)
955         return ((ssize_t)strcpy(string, SCF_STATE_STRING_MAINT, len));
956     else if (state == RESTARTER_STATE_OFFLINE)
957         return ((ssize_t)strcpy(string, SCF_STATE_STRING_OFFLINE,
958                                 len));
959     else if (state == RESTARTER_STATE_DISABLED)
960         return ((ssize_t)strcpy(string, SCF_STATE_STRING_DISABLED,
961                                 len));
962     else if (state == RESTARTER_STATE_ONLINE)
963         return ((ssize_t)strcpy(string, SCF_STATE_STRING_ONLINE, len));
964     else if (state == RESTARTER_STATE_DEGRADED)
965         return ((ssize_t)strcpy(string, SCF_STATE_STRING_DEGRADED,
966                                 len));
967     else
968         return ((ssize_t)strcpy(string, "unknown", len));
969 }

971 /*
972  * Sets pg to the name property group of s_inst.  If it doesn't exist, it is
973  * added.
974  *
975  * Fails with
976  *   ECONNABORTED - repository disconnection or unknown libscf error
977  *   EBADF - inst is not set
978  *   ECANCELED - inst is deleted
979  *   EPERM - permission is denied
980  *   EACCES - backend denied access
981  *   EROFS - backend readonly
982  */

```

```

983 static int
984 instance_get_or_add_pg(scf_instance_t *inst, const char *name,
985                       const char *type, uint32_t flags, scf_propertygroup_t *pg)
986 {
987     again:
988     if (scf_instance_get_pg(inst, name, pg) == 0)
989         return (0);

991     switch (scf_error()) {
992     case SCF_ERROR_CONNECTION_BROKEN:
993     default:
994         return (ECONNABORTED);

996     case SCF_ERROR_NOT_SET:
997         return (EBADF);

999     case SCF_ERROR_DELETED:
1000         return (ECANCELED);

1002     case SCF_ERROR_NOT_FOUND:
1003         break;

1005     case SCF_ERROR_HANDLE_MISMATCH:
1006     case SCF_ERROR_INVALID_ARGUMENT:
1007         bad_fail("scf_instance_get_pg", scf_error());
1008     }

1010     if (scf_instance_add_pg(inst, name, type, flags, pg) == 0)
1011         return (0);

1013     switch (scf_error()) {
1014     case SCF_ERROR_CONNECTION_BROKEN:
1015     default:
1016         return (ECONNABORTED);

1018     case SCF_ERROR_DELETED:
1019         return (ECANCELED);

1021     case SCF_ERROR_EXISTS:
1022         goto again;

1024     case SCF_ERROR_PERMISSION_DENIED:
1025         return (EPERM);

1027     case SCF_ERROR_BACKEND_ACCESS:
1028         return (EACCES);

1030     case SCF_ERROR_BACKEND_READONLY:
1031         return (EROFS);

1033     case SCF_ERROR_HANDLE_MISMATCH:
1034     case SCF_ERROR_INVALID_ARGUMENT:
1035     case SCF_ERROR_NOT_SET:
1036         bad_fail("scf_instance_add_pg", scf_error());
1037     }

1039     return (0);
1040 }

1042 /*
1043  * Fails with
1044  *   ECONNABORTED
1045  *   ECANCELED - pg was deleted
1046  */
1047 static int
1048 tx_set_value(scf_transaction_t *tx, scf_transaction_entry_t *ent,

```

```

1049     const char *pname, scf_type_t ty, scf_value_t *val)
1050 {
1051     int r;
1052
1053     for (;;) {
1054         if (scf_transaction_property_change_type(tx, ent, pname,
1055             ty) == 0)
1056             break;
1057
1058         switch (scf_error()) {
1059             case SCF_ERROR_CONNECTION_BROKEN:
1060             default:
1061                 return (ECONNABORTED);
1062
1063             case SCF_ERROR_DELETED:
1064                 return (ECANCELED);
1065
1066             case SCF_ERROR_NOT_FOUND:
1067                 break;
1068
1069             case SCF_ERROR_HANDLE_MISMATCH:
1070             case SCF_ERROR_INVALID_ARGUMENT:
1071             case SCF_ERROR_IN_USE:
1072             case SCF_ERROR_NOT_SET:
1073                 bad_fail("scf_transaction_property_change_type",
1074                     scf_error());
1075         }
1076
1077         if (scf_transaction_property_new(tx, ent, pname, ty) == 0)
1078             break;
1079
1080         switch (scf_error()) {
1081             case SCF_ERROR_CONNECTION_BROKEN:
1082             default:
1083                 return (ECONNABORTED);
1084
1085             case SCF_ERROR_DELETED:
1086                 return (ECANCELED);
1087
1088             case SCF_ERROR_EXISTS:
1089                 break;
1090
1091             case SCF_ERROR_HANDLE_MISMATCH:
1092             case SCF_ERROR_INVALID_ARGUMENT:
1093             case SCF_ERROR_IN_USE:
1094             case SCF_ERROR_NOT_SET:
1095                 bad_fail("scf_transaction_property_new", scf_error());
1096         }
1097     }
1098
1099     r = scf_entry_add_value(ent, val);
1100     assert(r == 0);
1101
1102     return (0);
1103 }
1104
1105 /*
1106 * Commit new_state, new_next_state, and aux to the repository for id.  If
1107 * successful, also set id's state and next-state as given, and return 0.
1108 * Fails with
1109 *   ENOMEM - out of memory
1110 *   ECONNABORTED - repository connection broken
1111 *   - unknown libscf error
1112 *   EINVAL - id->i_fmri is invalid or not an instance FMRI
1113 *   ENOENT - id->i_fmri does not exist
1114 *   EPERM - insufficient permissions

```

```

1115 *   EACCES - backend access denied
1116 *   EROFS - backend is readonly
1117 */
1118 int
1119 _restarter_commit_states(scf_handle_t *h, instance_data_t *id,
1120     restarter_instance_state_t new_state,
1121     restarter_instance_state_t new_state_next, const char *aux)
1122 {
1123     char str_state[MAX_SCF_STATE_STRING_SZ];
1124     char str_new_state[MAX_SCF_STATE_STRING_SZ];
1125     char str_state_next[MAX_SCF_STATE_STRING_SZ];
1126     char str_new_state_next[MAX_SCF_STATE_STRING_SZ];
1127     int ret = 0, r;
1128     struct timeval now;
1129     ssize_t sz;
1130
1131     scf_transaction_t *t = NULL;
1132     scf_transaction_entry_t *t_state = NULL, *t_state_next = NULL;
1133     scf_transaction_entry_t *t_stime = NULL, *t_aux = NULL;
1134     scf_value_t *v_state = NULL, *v_state_next = NULL, *v_stime = NULL;
1135     scf_value_t *v_aux = NULL;
1136     scf_instance_t *s_inst = NULL;
1137     scf_propertygroup_t *pg = NULL;
1138
1139     assert(new_state != RESTARTER_STATE_NONE);
1140
1141     if ((s_inst = scf_instance_create(h)) == NULL ||
1142         (pg = scf_pg_create(h)) == NULL ||
1143         (t = scf_transaction_create(h)) == NULL ||
1144         (t_state = scf_entry_create(h)) == NULL ||
1145         (t_state_next = scf_entry_create(h)) == NULL ||
1146         (t_stime = scf_entry_create(h)) == NULL ||
1147         (t_aux = scf_entry_create(h)) == NULL ||
1148         (v_state = scf_value_create(h)) == NULL ||
1149         (v_state_next = scf_value_create(h)) == NULL ||
1150         (v_stime = scf_value_create(h)) == NULL ||
1151         (v_aux = scf_value_create(h)) == NULL) {
1152         ret = ENOMEM;
1153         goto out;
1154     }
1155
1156     sz = restarter_state_to_string(new_state, str_new_state,
1157         sizeof (str_new_state));
1158     assert(sz < sizeof (str_new_state));
1159     sz = restarter_state_to_string(new_state_next, str_new_state_next,
1160         sizeof (str_new_state_next));
1161     assert(sz < sizeof (str_new_state_next));
1162     sz = restarter_state_to_string(id->i_state, str_state,
1163         sizeof (str_state));
1164     assert(sz < sizeof (str_state));
1165     sz = restarter_state_to_string(id->i_next_state, str_state_next,
1166         sizeof (str_state_next));
1167     assert(sz < sizeof (str_state_next));
1168
1169     ret = gettimeofday(&now, NULL);
1170     assert(ret != -1);
1171
1172     if (scf_handle_decode_fmri(h, id->i_fmri, NULL, NULL, s_inst,
1173         NULL, NULL, SCF_DECODE_FMRI_EXACT) == -1) {
1174         switch (scf_error()) {
1175             case SCF_ERROR_CONNECTION_BROKEN:
1176             default:
1177                 ret = ECONNABORTED;
1178                 break;
1179
1180             case SCF_ERROR_INVALID_ARGUMENT:

```



```

1181         case SCF_ERROR_CONSTRAINT_VIOLATED:
1182             ret = EINVAL;
1183             break;
1185         case SCF_ERROR_NOT_FOUND:
1186             ret = ENOENT;
1187             break;
1189         case SCF_ERROR_HANDLE_MISMATCH:
1190             bad_fail("scf_handle_decode_fmri", scf_error());
1191         }
1192         goto out;
1193     }
1196     if (scf_value_set_astring(v_state, str_new_state) != 0 ||
1197         scf_value_set_astring(v_state_next, str_new_state_next) != 0)
1198         bad_fail("scf_value_set_astring", scf_error());
1200     if (aux) {
1201         if (scf_value_set_astring(v_aux, aux) != 0)
1202             bad_fail("scf_value_set_astring", scf_error());
1203     }
1205     if (scf_value_set_time(v_stime, now.tv_sec, now.tv_usec * 1000) != 0)
1206         bad_fail("scf_value_set_time", scf_error());
1208 add_pg:
1209     switch (r = instance_get_or_add_pg(s_inst, SCF_PG_RESTARTER,
1210         SCF_PG_RESTARTER_TYPE, SCF_PG_RESTARTER_FLAGS, pg)) {
1211     case 0:
1212         break;
1214     case ECONNABORTED:
1215     case EPERM:
1216     case EACCES:
1217     case EROFS:
1218         ret = r;
1219         goto out;
1221     case ECANCELED:
1222         ret = ENOENT;
1223         goto out;
1225     case EBADF:
1226     default:
1227         bad_fail("instance_get_or_add_pg", r);
1228     }
1230     for (;;) {
1231         if (scf_transaction_start(t, pg) != 0) {
1232             switch (scf_error()) {
1233             case SCF_ERROR_CONNECTION_BROKEN:
1234             default:
1235                 ret = ECONNABORTED;
1236                 goto out;
1238             case SCF_ERROR_NOT_SET:
1239                 goto add_pg;
1241             case SCF_ERROR_PERMISSION_DENIED:
1242                 ret = EPERM;
1243                 goto out;
1245             case SCF_ERROR_BACKEND_ACCESS:
1246                 ret = EACCES;

```

```

1247         goto out;
1249         case SCF_ERROR_BACKEND_READONLY:
1250             ret = EROFS;
1251             goto out;
1253         case SCF_ERROR_HANDLE_MISMATCH:
1254         case SCF_ERROR_IN_USE:
1255             bad_fail("scf_transaction_start", scf_error());
1256         }
1257     }
1259     if ((r = tx_set_value(t, t_state, SCF_PROPERTY_STATE,
1260         SCF_TYPE_ASTRING, v_state)) != 0 ||
1261         (r = tx_set_value(t, t_state_next, SCF_PROPERTY_NEXT_STATE,
1262         SCF_TYPE_ASTRING, v_state_next)) != 0 ||
1263         (r = tx_set_value(t, t_stime, SCF_PROPERTY_STATE_TIMESTAMP,
1264         SCF_TYPE_TIME, v_stime)) != 0) {
1265         switch (r) {
1266         case ECONNABORTED:
1267             ret = ECONNABORTED;
1268             goto out;
1270         case ECANCELED:
1271             scf_transaction_reset(t);
1272             goto add_pg;
1274         default:
1275             bad_fail("tx_set_value", r);
1276         }
1277     }
1279     if (aux) {
1280         if ((r = tx_set_value(t, t_aux, SCF_PROPERTY_AUX_STATE,
1281         SCF_TYPE_ASTRING, v_aux)) != 0) {
1282             switch (r) {
1283             case ECONNABORTED:
1284                 ret = ECONNABORTED;
1285                 goto out;
1287             case ECANCELED:
1288                 scf_transaction_reset(t);
1289                 goto add_pg;
1291             default:
1292                 bad_fail("tx_set_value", r);
1293             }
1294         }
1295     }
1297     ret = scf_transaction_commit(t);
1298     if (ret == 1)
1299         break;
1300     if (ret == -1) {
1301         switch (scf_error()) {
1302         case SCF_ERROR_CONNECTION_BROKEN:
1303         default:
1304             ret = ECONNABORTED;
1305             goto out;
1307         case SCF_ERROR_PERMISSION_DENIED:
1308             ret = EPERM;
1309             goto out;
1311         case SCF_ERROR_BACKEND_ACCESS:
1312             ret = EACCES;

```

```

1313         goto out;
1315     case SCF_ERROR_BACKEND_READONLY:
1316         ret = EROFS;
1317         goto out;
1319     case SCF_ERROR_NOT_SET:
1320         bad_fail("scf_transaction_commit", scf_error());
1321     }
1322 }
1324     scf_transaction_reset(t);
1325     if (scf_pg_update(pg) == -1) {
1326         switch (scf_error()) {
1327             case SCF_ERROR_CONNECTION_BROKEN:
1328             default:
1329                 ret = ECONNABORTED;
1330                 goto out;
1332         case SCF_ERROR_NOT_SET:
1333             goto add_pg;
1334         }
1335     }
1336 }
1338     id->i_state = new_state;
1339     id->i_next_state = new_state_next;
1340     ret = 0;
1342 out:
1343     scf_transaction_destroy(t);
1344     scf_entry_destroy(t_state);
1345     scf_entry_destroy(t_state_next);
1346     scf_entry_destroy(t_stime);
1347     scf_entry_destroy(t_aux);
1348     scf_value_destroy(v_state);
1349     scf_value_destroy(v_state_next);
1350     scf_value_destroy(v_stime);
1351     scf_value_destroy(v_aux);
1352     scf_pg_destroy(pg);
1353     scf_instance_destroy(s_inst);
1355     return (ret);
1356 }
1358 /*
1359  * Fails with
1360  * EINVAL - type is invalid
1361  * ENOMEM
1362  * ECONNABORTED - repository connection broken
1363  * EBADF - s_inst is not set
1364  * ECANCELED - s_inst is deleted
1365  * EPERM - permission denied
1366  * EACCES - backend access denied
1367  * EROFS - backend readonly
1368  */
1369 int
1370 restarter_remove_contract(scf_instance_t *s_inst, ctid_t contract_id,
1371     restarter_contract_type_t type)
1372 {
1373     scf_handle_t *h;
1374     scf_transaction_t *t = NULL;
1375     scf_transaction_entry_t *t_cid = NULL;
1376     scf_propertygroup_t *pg = NULL;
1377     scf_property_t *prop = NULL;
1378     scf_value_t *val;

```

```

1379     scf_iter_t *iter = NULL;
1380     const char *pname;
1381     int ret = 0, primary;
1382     uint64_t c;
1384     switch (type) {
1385     case RESTARTER_CONTRACT_PRIMARY:
1386         primary = 1;
1387         break;
1388     case RESTARTER_CONTRACT_TRANSIENT:
1389         primary = 0;
1390         break;
1391     default:
1392         return (EINVAL);
1393     }
1395     h = scf_instance_handle(s_inst);
1397     pg = scf_pg_create(h);
1398     prop = scf_property_create(h);
1399     iter = scf_iter_create(h);
1400     t = scf_transaction_create(h);
1402     if (pg == NULL || prop == NULL || iter == NULL || t == NULL) {
1403         ret = ENOMEM;
1404         goto remove_contract_cleanup;
1405     }
1407 add:
1408     scf_transaction_destroy_children(t);
1409     ret = instance_get_or_add_pg(s_inst, SCF_PG_RESTARTER,
1410         SCF_PG_RESTARTER_TYPE, SCF_PG_RESTARTER_FLAGS, pg);
1411     if (ret != 0)
1412         goto remove_contract_cleanup;
1414     pname = primary? SCF_PROPERTY_CONTRACT :
1415         SCF_PROPERTY_TRANSIENT_CONTRACT;
1417     for (;;) {
1418         if (scf_transaction_start(t, pg) != 0) {
1419             switch (scf_error()) {
1420             case SCF_ERROR_CONNECTION_BROKEN:
1421             default:
1422                 ret = ECONNABORTED;
1423                 goto remove_contract_cleanup;
1425             case SCF_ERROR_DELETED:
1426                 goto add;
1428             case SCF_ERROR_PERMISSION_DENIED:
1429                 ret = EPERM;
1430                 goto remove_contract_cleanup;
1432             case SCF_ERROR_BACKEND_ACCESS:
1433                 ret = EACCES;
1434                 goto remove_contract_cleanup;
1436             case SCF_ERROR_BACKEND_READONLY:
1437                 ret = EROFS;
1438                 goto remove_contract_cleanup;
1440             case SCF_ERROR_HANDLE_MISMATCH:
1441             case SCF_ERROR_IN_USE:
1442             case SCF_ERROR_NOT_SET:
1443                 bad_fail("scf_transaction_start", scf_error());
1444             }

```

```

1445     }
1447     t_cid = scf_entry_create(h);
1449     if (scf_pg_get_property(pg, pname, prop) == 0) {
1450 replace:
1451         if (scf_transaction_property_change_type(t, t_cid,
1452             pname, SCF_TYPE_COUNT) != 0) {
1453             switch (scf_error()) {
1454             case SCF_ERROR_CONNECTION_BROKEN:
1455             default:
1456                 ret = ECONNABORTED;
1457                 goto remove_contract_cleanup;
1459
1460             case SCF_ERROR_DELETED:
1461                 scf_entry_destroy(t_cid);
1462                 goto add;
1463
1464             case SCF_ERROR_NOT_FOUND:
1465                 goto new;
1466
1467             case SCF_ERROR_HANDLE_MISMATCH:
1468             case SCF_ERROR_INVALID_ARGUMENT:
1469             case SCF_ERROR_IN_USE:
1470             case SCF_ERROR_NOT_SET:
1471                 bad_fail(
1472                     "scf_transaction_property_changetype",
1473                     scf_error());
1474             }
1476         if (scf_property_is_type(prop, SCF_TYPE_COUNT) == 0) {
1477             if (scf_iter_property_values(iter, prop) != 0) {
1478                 switch (scf_error()) {
1479                 case SCF_ERROR_CONNECTION_BROKEN:
1480                 default:
1481                     ret = ECONNABORTED;
1482                     goto remove_contract_cleanup;
1484
1485                 case SCF_ERROR_NOT_SET:
1486                 case SCF_ERROR_HANDLE_MISMATCH:
1487                     bad_fail(
1488                         "scf_iter_property_values",
1489                         scf_error());
1490                 }
1492 next_val:
1493                 val = scf_value_create(h);
1494                 if (val == NULL) {
1495                     assert(scf_error() ==
1496                         SCF_ERROR_NO_MEMORY);
1497                     ret = ENOMEM;
1498                     goto remove_contract_cleanup;
1499                 }
1501                 ret = scf_iter_next_value(iter, val);
1502                 if (ret == -1) {
1503                     switch (scf_error()) {
1504                     case SCF_ERROR_CONNECTION_BROKEN:
1505                     default:
1506                         ret = ECONNABORTED;
1507                         goto remove_contract_cleanup;
1508
1509                     case SCF_ERROR_DELETED:
1510                         scf_value_destroy(val);
1511                         goto add;

```

```

1512             case SCF_ERROR_HANDLE_MISMATCH:
1513             case SCF_ERROR_INVALID_ARGUMENT:
1514             case SCF_ERROR_PERMISSION_DENIED:
1515             default:
1516                 bad_fail("scf_iter_next_value",
1517                     scf_error());
1518             }
1519         }
1521         if (ret == 1) {
1522             ret = scf_value_get_count(val, &c);
1523             assert(ret == 0);
1525             if (c != contract_id) {
1526                 ret = scf_entry_add_value(t_cid,
1527                     val);
1528                 assert(ret == 0);
1529             } else {
1530                 scf_value_destroy(val);
1531             }
1533             goto next_val;
1534         }
1536         scf_value_destroy(val);
1537     } else {
1538         switch (scf_error()) {
1539         case SCF_ERROR_CONNECTION_BROKEN:
1540         default:
1541             ret = ECONNABORTED;
1542             goto remove_contract_cleanup;
1544
1545         case SCF_ERROR_TYPE_MISMATCH:
1546             break;
1547
1548         case SCF_ERROR_INVALID_ARGUMENT:
1549         case SCF_ERROR_NOT_SET:
1550             bad_fail("scf_property_is_type",
1551                 scf_error());
1552         }
1553     } else {
1554         switch (scf_error()) {
1555         case SCF_ERROR_CONNECTION_BROKEN:
1556         default:
1557             ret = ECONNABORTED;
1558             goto remove_contract_cleanup;
1560
1561         case SCF_ERROR_DELETED:
1562             scf_entry_destroy(t_cid);
1563             goto add;
1564
1565         case SCF_ERROR_NOT_FOUND:
1566             break;
1567
1568         case SCF_ERROR_HANDLE_MISMATCH:
1569         case SCF_ERROR_INVALID_ARGUMENT:
1570         case SCF_ERROR_NOT_SET:
1571             bad_fail("scf_pg_get_property", scf_error());
1572         }
1573 new:
1574         if (scf_transaction_property_new(t, t_cid, pname,
1575             SCF_TYPE_COUNT) != 0) {
1576             switch (scf_error()) {

```

```

1577         case SCF_ERROR_CONNECTION_BROKEN:
1578         default:
1579             ret = ECONNABORTED;
1580             goto remove_contract_cleanup;

1582         case SCF_ERROR_DELETED:
1583             scf_entry_destroy(t_cid);
1584             goto add;

1586         case SCF_ERROR_EXISTS:
1587             goto replace;

1589         case SCF_ERROR_HANDLE_MISMATCH:
1590         case SCF_ERROR_INVALID_ARGUMENT:
1591         case SCF_ERROR_NOT_SET:
1592             bad_fail("scf_transaction_property_new",
1593                    scf_error());
1594     }
1595 }
1596 }

1598 ret = scf_transaction_commit(t);
1599 if (ret == -1) {
1600     switch (scf_error()) {
1601     case SCF_ERROR_CONNECTION_BROKEN:
1602     default:
1603         ret = ECONNABORTED;
1604         goto remove_contract_cleanup;

1606     case SCF_ERROR_DELETED:
1607         goto add;

1609     case SCF_ERROR_PERMISSION_DENIED:
1610         ret = EPERM;
1611         goto remove_contract_cleanup;

1613     case SCF_ERROR_BACKEND_ACCESS:
1614         ret = EACCES;
1615         goto remove_contract_cleanup;

1617     case SCF_ERROR_BACKEND_READONLY:
1618         ret = EROFS;
1619         goto remove_contract_cleanup;

1621     case SCF_ERROR_NOT_SET:
1622         bad_fail("scf_transaction_commit", scf_error());
1623     }
1624 }
1625 if (ret == 1) {
1626     ret = 0;
1627     break;
1628 }

1630 scf_transaction_destroy_children(t);
1631 if (scf_pg_update(pg) == -1) {
1632     switch (scf_error()) {
1633     case SCF_ERROR_CONNECTION_BROKEN:
1634     default:
1635         ret = ECONNABORTED;
1636         goto remove_contract_cleanup;

1638     case SCF_ERROR_DELETED:
1639         goto add;

1641     case SCF_ERROR_NOT_SET:
1642         bad_fail("scf_pg_update", scf_error());

```

```

1643     }
1644 }
1645 }

1647 remove_contract_cleanup:
1648     scf_transaction_destroy_children(t);
1649     scf_transaction_destroy(t);
1650     scf_iter_destroy(iter);
1651     scf_property_destroy(prop);
1652     scf_pg_destroy(pg);

1654     return (ret);
1655 }

1657 /*
1658  * Fails with
1659  * EINVAL - type is invalid
1660  * ENOMEM
1661  * ECONNABORTED - repository disconnection
1662  * EBADF - s_inst is not set
1663  * ECANCELED - s_inst is deleted
1664  * EPERM
1665  * EACCES
1666  * EROFS
1667  */
1668 int
1669 restarter_store_contract(scf_instance_t *s_inst, ctid_t contract_id,
1670                         restarter_contract_type_t type)
1671 {
1672     scf_handle_t *h;
1673     scf_transaction_t *t = NULL;
1674     scf_transaction_entry_t *t_cid = NULL;
1675     scf_value_t *val;
1676     scf_propertygroup_t *pg = NULL;
1677     scf_property_t *prop = NULL;
1678     scf_iter_t *iter = NULL;
1679     const char *pname;
1680     int ret = 0, primary;

1682     if (type == RESTARTER_CONTRACT_PRIMARY)
1683         primary = 1;
1684     else if (type == RESTARTER_CONTRACT_TRANSIENT)
1685         primary = 0;
1686     else
1687         return (EINVAL);

1689     h = scf_instance_handle(s_inst);

1691     pg = scf_pg_create(h);
1692     prop = scf_property_create(h);
1693     iter = scf_iter_create(h);
1694     t = scf_transaction_create(h);

1696     if (pg == NULL || prop == NULL || iter == NULL || t == NULL) {
1697         ret = ENOMEM;
1698         goto out;
1699     }

1701 add:
1702     scf_transaction_destroy_children(t);
1703     ret = instance_get_or_add_pg(s_inst, SCF_PG_RESTARTER,
1704                                 SCF_PG_RESTARTER_TYPE, SCF_PG_RESTARTER_FLAGS, pg);
1705     if (ret != 0)
1706         goto out;

1708     pname = primary ? SCF_PROPERTY_CONTRACT :

```

```

1709     SCF_PROPERTY_TRANSIENT_CONTRACT;
1710
1711     for (;;) {
1712         if (scf_transaction_start(t, pg) != 0) {
1713             switch (scf_error()) {
1714                 case SCF_ERROR_CONNECTION_BROKEN:
1715                 default:
1716                     ret = ECONNABORTED;
1717                     goto out;
1718
1719                 case SCF_ERROR_DELETED:
1720                     goto add;
1721
1722                 case SCF_ERROR_PERMISSION_DENIED:
1723                     ret = EPERM;
1724                     goto out;
1725
1726                 case SCF_ERROR_BACKEND_ACCESS:
1727                     ret = EACCES;
1728                     goto out;
1729
1730                 case SCF_ERROR_BACKEND_READONLY:
1731                     ret = EROFS;
1732                     goto out;
1733
1734                 case SCF_ERROR_HANDLE_MISMATCH:
1735                 case SCF_ERROR_IN_USE:
1736                 case SCF_ERROR_NOT_SET:
1737                     bad_fail("scf_transaction_start", scf_error());
1738             }
1739         }
1740
1741         t_cid = scf_entry_create(h);
1742         if (t_cid == NULL) {
1743             ret = ENOMEM;
1744             goto out;
1745         }
1746
1747         if (scf_pg_get_property(pg, pname, prop) == 0) {
1748             replace:
1749             if (scf_transaction_property_change_type(t, t_cid,
1750                 pname, SCF_TYPE_COUNT) != 0) {
1751                 switch (scf_error()) {
1752                     case SCF_ERROR_CONNECTION_BROKEN:
1753                     default:
1754                         ret = ECONNABORTED;
1755                         goto out;
1756
1757                     case SCF_ERROR_DELETED:
1758                         scf_entry_destroy(t_cid);
1759                         goto add;
1760
1761                     case SCF_ERROR_NOT_FOUND:
1762                         goto new;
1763
1764                     case SCF_ERROR_HANDLE_MISMATCH:
1765                     case SCF_ERROR_INVALID_ARGUMENT:
1766                     case SCF_ERROR_IN_USE:
1767                     case SCF_ERROR_NOT_SET:
1768                         bad_fail(
1769                             "scf_transaction_propert_change_type",
1770                             scf_error());
1771                 }
1772             }
1773
1774             if (scf_property_is_type(prop, SCF_TYPE_COUNT) == 0) {

```

```

1775         if (scf_iter_property_values(iter, prop) != 0) {
1776             switch (scf_error()) {
1777                 case SCF_ERROR_CONNECTION_BROKEN:
1778                 default:
1779                     ret = ECONNABORTED;
1780                     goto out;
1781
1782                 case SCF_ERROR_NOT_SET:
1783                 case SCF_ERROR_HANDLE_MISMATCH:
1784                     bad_fail(
1785                         "scf_iter_property_values",
1786                         scf_error());
1787             }
1788         }
1789
1790     next_val:
1791     val = scf_value_create(h);
1792     if (val == NULL) {
1793         assert(scf_error() ==
1794             SCF_ERROR_NO_MEMORY);
1795         ret = ENOMEM;
1796         goto out;
1797     }
1798
1799     ret = scf_iter_next_value(iter, val);
1800     if (ret == -1) {
1801         switch (scf_error()) {
1802             case SCF_ERROR_CONNECTION_BROKEN:
1803             default:
1804                 ret = ECONNABORTED;
1805                 goto out;
1806
1807             case SCF_ERROR_DELETED:
1808                 scf_value_destroy(val);
1809                 goto add;
1810
1811             case SCF_ERROR_HANDLE_MISMATCH:
1812             case SCF_ERROR_INVALID_ARGUMENT:
1813             case SCF_ERROR_PERMISSION_DENIED:
1814                 bad_fail(
1815                     "scf_iter_next_value",
1816                     scf_error());
1817         }
1818     }
1819
1820     if (ret == 1) {
1821         ret = scf_entry_add_value(t_cid, val);
1822         assert(ret == 0);
1823
1824         goto next_val;
1825     }
1826
1827     scf_value_destroy(val);
1828 } else {
1829     switch (scf_error()) {
1830         case SCF_ERROR_CONNECTION_BROKEN:
1831         default:
1832             ret = ECONNABORTED;
1833             goto out;
1834
1835         case SCF_ERROR_TYPE_MISMATCH:
1836             break;
1837
1838         case SCF_ERROR_INVALID_ARGUMENT:
1839         case SCF_ERROR_NOT_SET:
1840             bad_fail("scf_property_is_type",

```

```

1841         scf_error());
1842     }
1843     } else {
1844     }
1845     switch (scf_error()) {
1846     case SCF_ERROR_CONNECTION_BROKEN:
1847     default:
1848         ret = ECONNABORTED;
1849         goto out;
1851     case SCF_ERROR_DELETED:
1852         scf_entry_destroy(t_cid);
1853         goto add;
1855     case SCF_ERROR_NOT_FOUND:
1856         break;
1858     case SCF_ERROR_HANDLE_MISMATCH:
1859     case SCF_ERROR_INVALID_ARGUMENT:
1860     case SCF_ERROR_NOT_SET:
1861         bad_fail("scf_pg_get_property", scf_error());
1862     }
1864 new:
1865     if (scf_transaction_property_new(t, t_cid, pname,
1866         SCF_TYPE_COUNT) != 0) {
1867         switch (scf_error()) {
1868         case SCF_ERROR_CONNECTION_BROKEN:
1869         default:
1870             ret = ECONNABORTED;
1871             goto out;
1873         case SCF_ERROR_DELETED:
1874             scf_entry_destroy(t_cid);
1875             goto add;
1877         case SCF_ERROR_EXISTS:
1878             goto replace;
1880         case SCF_ERROR_HANDLE_MISMATCH:
1881         case SCF_ERROR_INVALID_ARGUMENT:
1882         case SCF_ERROR_NOT_SET:
1883             bad_fail("scf_transaction_property_new",
1884                 scf_error());
1885         }
1886     }
1887 }
1889 val = scf_value_create(h);
1890 if (val == NULL) {
1891     assert(scf_error() == SCF_ERROR_NO_MEMORY);
1892     ret = ENOMEM;
1893     goto out;
1894 }
1896 scf_value_set_count(val, contract_id);
1897 ret = scf_entry_add_value(t_cid, val);
1898 assert(ret == 0);
1900 ret = scf_transaction_commit(t);
1901 if (ret == -1) {
1902     switch (scf_error()) {
1903     case SCF_ERROR_CONNECTION_BROKEN:
1904     default:
1905         ret = ECONNABORTED;
1906         goto out;

```

```

1908     case SCF_ERROR_DELETED:
1909         goto add;
1911     case SCF_ERROR_PERMISSION_DENIED:
1912         ret = EPERM;
1913         goto out;
1915     case SCF_ERROR_BACKEND_ACCESS:
1916         ret = EACCES;
1917         goto out;
1919     case SCF_ERROR_BACKEND_READONLY:
1920         ret = EROFS;
1921         goto out;
1923     case SCF_ERROR_NOT_SET:
1924         bad_fail("scf_transaction_commit", scf_error());
1925     }
1926     }
1927     if (ret == 1) {
1928         ret = 0;
1929         break;
1930     }
1932     scf_transaction_destroy_children(t);
1933     if (scf_pg_update(pg) == -1) {
1934         switch (scf_error()) {
1935         case SCF_ERROR_CONNECTION_BROKEN:
1936         default:
1937             ret = ECONNABORTED;
1938             goto out;
1940         case SCF_ERROR_DELETED:
1941             goto add;
1943         case SCF_ERROR_NOT_SET:
1944             bad_fail("scf_pg_update", scf_error());
1945         }
1946     }
1947 }
1949 out:
1950     scf_transaction_destroy_children(t);
1951     scf_transaction_destroy(t);
1952     scf_iter_destroy(iter);
1953     scf_property_destroy(prop);
1954     scf_pg_destroy(pg);
1956     return (ret);
1957 }
1959 int
1960 restarter_rm_libs_loadable()
1961 {
1962     void *libhndl;
1964     if (method_context_safety)
1965         return (1);
1967     if ((libhndl = dlopen("libpool.so", RTLD_LAZY | RTLD_LOCAL)) == NULL)
1968         return (0);
1970     (void) dlclose(libhndl);
1972     if ((libhndl = dlopen("libproject.so", RTLD_LAZY | RTLD_LOCAL)) == NULL)

```

```

1973         return (0);
1975     (void) dlclose(libhndl);
1977     method_context_safety = 1;
1979     return (1);
1980 }

1982 static int
1983 get_astring_val(scf_propertygroup_t *pg, const char *name, char *buf,
1984               size_t bufsz, scf_property_t *prop, scf_value_t *val)
1985 {
1986     ssize_t szret;

1988     if (pg == NULL)
1989         return (-1);

1991     if (scf_pg_get_property(pg, name, prop) != SCF_SUCCESS) {
1992         if (scf_error() == SCF_ERROR_CONNECTION_BROKEN)
1993             uu_die(rcbroken);
1994         return (-1);
1995     }

1997     if (scf_property_get_value(prop, val) != SCF_SUCCESS) {
1998         if (scf_error() == SCF_ERROR_CONNECTION_BROKEN)
1999             uu_die(rcbroken);
2000         return (-1);
2001     }

2003     szret = scf_value_get_astring(val, buf, bufsz);

2005     return (szret >= 0 ? 0 : -1);
2006 }

2008 static int
2009 get_boolean_val(scf_propertygroup_t *pg, const char *name, uint8_t *b,
2010               scf_property_t *prop, scf_value_t *val)
2011 {
2012     if (scf_pg_get_property(pg, name, prop) != SCF_SUCCESS) {
2013         if (scf_error() == SCF_ERROR_CONNECTION_BROKEN)
2014             uu_die(rcbroken);
2015         return (-1);
2016     }

2018     if (scf_property_get_value(prop, val) != SCF_SUCCESS) {
2019         if (scf_error() == SCF_ERROR_CONNECTION_BROKEN)
2020             uu_die(rcbroken);
2021         return (-1);
2022     }

2024     if (scf_value_get_boolean(val, b))
2025         return (-1);

2027     return (0);
2028 }

2030 /*
2031  * Try to load mcp->pwd, if it isn't already.
2032  * Fails with
2033  *   ENOMEM - malloc() failed
2034  *   ENOENT - no entry found
2035  *   EIO - I/O error
2036  *   EMFILE - process out of file descriptors
2037  *   ENFILE - system out of file handles
2038  */

```

```

2039 static int
2040 lookup_pwd(struct method_context *mcp)
2041 {
2042     struct passwd *pwdp;

2044     if (mcp->pwbuf != NULL && mcp->pwd.pw_uid == mcp->uid)
2045         return (0);

2047     if (mcp->pwbuf == NULL) {
2048         mcp->pwbufsz = sysconf(_SC_GETPW_R_SIZE_MAX);
2049         assert(mcp->pwbufsz >= 0);
2050         mcp->pwbuf = malloc(mcp->pwbufsz);
2051         if (mcp->pwbuf == NULL)
2052             return (ENOMEM);
2053     }

2055     do {
2056         errno = 0;
2057         pwdp = getpwuid_r(mcp->uid, &mcp->pwd, mcp->pwbuf,
2058                         mcp->pwbufsz);
2059     } while (pwdp == NULL && errno == EINTR);
2060     if (pwdp != NULL)
2061         return (0);

2063     free(mcp->pwbuf);
2064     mcp->pwbuf = NULL;

2066     switch (errno) {
2067     case 0:
2068     default:
2069         /*
2070          * Until bug 5065780 is fixed, getpwuid_r() can fail with
2071          * ENOENT, particularly on the miniroot. Since the
2072          * documentation is inaccurate, we'll return ENOENT for unknown
2073          * errors.
2074          */
2075         return (ENOENT);

2077     case EIO:
2078     case EMFILE:
2079     case ENFILE:
2080         return (errno);

2082     case ERANGE:
2083         bad_fail("getpwuid_r", errno);
2084         /* NOTREACHED */
2085     }
2086 }

2088 /*
2089  * Get the user id for str. Returns 0 on success or
2090  *   ERANGE the uid is too big
2091  *   EINVAL the string starts with a digit, but is not a valid uid
2092  *   ENOMEM out of memory
2093  *   ENOENT no passwd entry for str
2094  *   EIO an I/O error has occurred
2095  *   EMFILE/ENFILE out of file descriptors
2096  */
2097 int
2098 get_uid(const char *str, struct method_context *ci, uid_t *uidp)
2099 {
2100     if (isdigit(str[0])) {
2101         uid_t uid;
2102         char *cp;

2104         errno = 0;

```

```

2105         uid = strtol(str, &cp, 10);
2107
2108         if (uid == 0 && errno != 0) {
2109             assert(errno != EINVAL);
2110             return (errno);
2111         }
2112
2113         for (; *cp != '\0'; ++cp)
2114             if (*cp != ' ' || *cp != '\t')
2115                 return (EINVAL);
2116
2117         if (uid > UID_MAX)
2118             return (EINVAL);
2119
2120         *uidp = uid;
2121         return (0);
2122     } else {
2123         struct passwd *pwdp;
2124
2125         if (ci->pwbuf == NULL) {
2126             ci->pwbufsz = sysconf(_SC_GETPW_R_SIZE_MAX);
2127             ci->pwbuf = malloc(ci->pwbufsz);
2128             if (ci->pwbuf == NULL)
2129                 return (ENOMEM);
2130         }
2131
2132         do {
2133             errno = 0;
2134             pwdp =
2135                 getpwnam_r(str, &ci->pwd, ci->pwbuf, ci->pwbufsz);
2136         } while (pwdp == NULL && errno == EINTR);
2137
2138         if (pwdp != NULL) {
2139             *uidp = ci->pwd.pw_uid;
2140             return (0);
2141         } else {
2142             free(ci->pwbuf);
2143             ci->pwbuf = NULL;
2144             switch (errno) {
2145                 case 0:
2146                     return (ENOENT);
2147
2148                 case ENOENT:
2149                 case EIO:
2150                 case EMFILE:
2151                 case ENFILE:
2152                     return (errno);
2153
2154                 case ERANGE:
2155                 default:
2156                     bad_fail("getpwnam_r", errno);
2157                     /* NOTREACHED */
2158             }
2159         }
2160     }
2161 }
2162
2163 gid_t
2164 get_gid(const char *str)
2165 {
2166     if (isdigit(str[0])) {
2167         gid_t gid;
2168         char *cp;
2169
2170         errno = 0;
2171         gid = strtol(str, &cp, 10);

```

```

2172         if (gid == 0 && errno != 0)
2173             return ((gid_t)-1);
2174
2175         for (; *cp != '\0'; ++cp)
2176             if (*cp != ' ' || *cp != '\t')
2177                 return ((gid_t)-1);
2178
2179         return (gid);
2180     } else {
2181         struct group grp, *ret;
2182         char *buffer;
2183         size_t buflen;
2184
2185         buflen = sysconf(_SC_GETGR_R_SIZE_MAX);
2186         buffer = malloc(buflen);
2187         if (buffer == NULL)
2188             uu_die(allocfail);
2189
2190         errno = 0;
2191         ret = getgrnam_r(str, &grp, buffer, buflen);
2192         free(buffer);
2193
2194         return (ret == NULL ? (gid_t)-1 : grp.gr_gid);
2195     }
2196 }
2197
2198 /*
2199  * Fails with
2200  * ENOMEM - out of memory
2201  * ENOENT - no passwd entry
2202  * EIO - no project entry
2203  * EIO - an I/O error occurred
2204  * EMFILE - the process is out of file descriptors
2205  * ENFILE - the system is out of file handles
2206  * ERANGE - the project id is out of range
2207  * EINVAL - str is invalid
2208  * E2BIG - the project entry was too big
2209  * -1 - the name service switch is misconfigured
2210  */
2211 int
2212 get_projid(const char *str, struct method_context *cip)
2213 {
2214     int ret;
2215     void *buf;
2216     const size_t bufsz = PROJECT_BUFSZ;
2217     struct project proj, *pp;
2218
2219     if (strcmp(str, ":default") == 0) {
2220         if (cip->uid == 0) {
2221             /* Don't change project for root services */
2222             cip->project = NULL;
2223             return (0);
2224         }
2225     }
2226
2227     switch (ret = lookup_pwd(cip)) {
2228     case 0:
2229         break;
2230
2231     case ENOMEM:
2232     case ENOENT:
2233     case EIO:
2234     case EMFILE:
2235     case ENFILE:
2236         return (ret);

```



```

2237         default:
2238             bad_fail("lookup_pwd", ret);
2239     }

2241     buf = malloc(bufsz);
2242     if (buf == NULL)
2243         return (ENOMEM);

2245     do {
2246         errno = 0;
2247         pp = getdefaultproj(cip->pwd.pw_name, &proj, buf,
2248             bufsz);
2249     } while (pp == NULL && errno == EINTR);

2251     /* to be continued ... */
2252 } else {
2253     projid_t projid;
2254     char *cp;

2256     if (!isdigit(str[0])) {
2257         cip->project = strdup(str);
2258         return (cip->project != NULL ? 0 : ENOMEM);
2259     }

2261     errno = 0;
2262     projid = strtol(str, &cp, 10);

2264     if (projid == 0 && errno != 0) {
2265         assert(errno == ERANGE);
2266         return (errno);
2267     }

2269     for (; *cp != '\0'; ++cp)
2270         if (*cp != ' ' || *cp != '\t')
2271             return (EINVAL);

2273     if (projid > MAXPROJID)
2274         return (ERANGE);

2276     buf = malloc(bufsz);
2277     if (buf == NULL)
2278         return (ENOMEM);

2280     do {
2281         errno = 0;
2282         pp = getprojbyid(projid, &proj, buf, bufsz);
2283     } while (pp == NULL && errno == EINTR);
2284 }

2286 if (pp) {
2287     cip->project = strdup(pp->pj_name);
2288     free(buf);
2289     return (cip->project != NULL ? 0 : ENOMEM);
2290 }

2292 free(buf);

2294 switch (errno) {
2295 case 0:
2296     return (ENOENT);

2298 case EIO:
2299 case EMFILE:
2300 case ENFILE:
2301     return (errno);

```

```

2303     case ERANGE:
2304         return (E2BIG);

2306     default:
2307         return (-1);
2308     }
2309 }

2311 /*
2312  * Parse the supp_groups property value and populate ci->groups. Returns
2313  * EINVAL (get_gid() failed for one of the components), E2BIG (the property has
2314  * more than NGROUPS_MAX-1 groups), or 0 on success.
2315  */
2316 int
2317 get_groups(char *str, struct method_context *ci)
2318 {
2319     char *cp, *end, *next;
2320     uint_t i;

2322     const char * const whitespace = " \t";
2323     const char * const illegal = ", \t";

2325     if (str[0] == '\0') {
2326         ci->ngroups = 0;
2327         return (0);
2328     }

2330     for (cp = str, i = 0; *cp != '\0'; ) {
2331         /* skip whitespace */
2332         cp += strspn(cp, whitespace);

2334         /* find the end */
2335         end = cp + strcspn(cp, illegal);

2337         /* skip whitespace after end */
2338         next = end + strspn(end, whitespace);

2340         /* if there's a comma, it separates the fields */
2341         if (*next == ',')
2342             ++next;

2344         *end = '\0';

2346         if ((ci->groups[i] = get_gid(cp)) == (gid_t)-1) {
2347             ci->ngroups = 0;
2348             return (EINVAL);
2349         }

2351         ++i;
2352         if (i > NGROUPS_MAX - 1) {
2353             ci->ngroups = 0;
2354             return (E2BIG);
2355         }

2357         cp = next;
2358     }

2360     ci->ngroups = i;
2361     return (0);
2362 }

2365 /*
2366  * Return an error message structure containing the error message
2367  * with context, and the error so the caller can make a decision
2368  * on what to do next.

```

```

2369 *
2370 * Because get_ids uses the mc_error_create() function which can
2371 * reallocate the merr, this function must return the merr pointer
2372 * in case it was reallocated.
2373 */
2374 static mc_error_t *
2375 get_profile(scf_propertygroup_t *methpg, scf_propertygroup_t *instpg,
2376            scf_property_t *prop, scf_value_t *val, const char *cmdline,
2377            struct method_context *ci, mc_error_t *merr)
2378 {
2379     char *buf = ci->vbuf;
2380     ssize_t buf_sz = ci->vbuf_sz;
2381     char cmd[PATH_MAX];
2382     char *cp, *value;
2383     const char *cmdp;
2384     execattr_t *eap;
2385     mc_error_t *err = merr;
2386     int r;
2387
2388     if (!(get_astring_val(methpg, SCF_PROPERTY_PROFILE, buf, buf_sz, prop,
2389                         val) == 0 || get_astring_val(instpg, SCF_PROPERTY_PROFILE, buf,
2390                         buf_sz, prop, val) == 0))
2391         return (mc_error_create(merr, scf_error(),
2392                                "Method context requires a profile, but the \"%s\" "
2393                                "property could not be read. scf_error is %s",
2394                                SCF_PROPERTY_PROFILE, scf_strerror(scf_error())));
2395
2396     /* Extract the command from the command line. */
2397     cp = strpbrk(cmdline, " \t");
2398
2399     if (cp == NULL) {
2400         cmdp = cmdline;
2401     } else {
2402         (void) strncpy(cmd, cmdline, cp - cmdline);
2403         cmd[cp - cmdline] = '\0';
2404         cmdp = cmd;
2405     }
2406
2407     /* Require that cmdp[0] == '/'? */
2408
2409     eap = getexecprof(buf, KV_COMMAND, cmdp, GET_ONE);
2410     if (eap == NULL)
2411         return (mc_error_create(merr, ENOENT,
2412                                "Could not find the execution profile \"%s\", "
2413                                "command %s.", buf, cmdp));
2414
2415     /* Based on pexec.c */
2416
2417     /* Get the euid first so we don't override ci->pwd for the uid. */
2418     if ((value = kva_match(eap->attr, EXECATTR_EUID_KW)) != NULL) {
2419         if ((r = get_uid(value, ci, &ci->euid)) != 0) {
2420             ci->euid = (uid_t)-1;
2421             err = mc_error_create(merr, r,
2422                                "Could not interpret profile euid value \"%s\", "
2423                                "from the execution profile \"%s\", error %d.",
2424                                value, buf, r);
2425             goto out;
2426         }
2427     }
2428
2429     if ((value = kva_match(eap->attr, EXECATTR_UID_KW)) != NULL) {
2430         if ((r = get_uid(value, ci, &ci->uid)) != 0) {
2431             ci->euid = ci->uid = (uid_t)-1;
2432             err = mc_error_create(merr, r,
2433                                "Could not interpret profile uid value \"%s\", "
2434                                "from the execution profile \"%s\", error %d.",

```

```

2435         value, buf, r);
2436         goto out;
2437     }
2438     ci->euid = ci->uid;
2439 }
2440
2441 if ((value = kva_match(eap->attr, EXECATTR_GID_KW)) != NULL) {
2442     ci->egid = ci->gid = get_gid(value);
2443     if (ci->gid == (gid_t)-1) {
2444         err = mc_error_create(merr, EINVAL,
2445                                "Could not interpret profile gid value \"%s\", "
2446                                "from the execution profile \"%s\".", value, buf);
2447         goto out;
2448     }
2449 }
2450
2451 if ((value = kva_match(eap->attr, EXECATTR_EGID_KW)) != NULL) {
2452     ci->egid = get_gid(value);
2453     if (ci->egid == (gid_t)-1) {
2454         err = mc_error_create(merr, EINVAL,
2455                                "Could not interpret profile egid value \"%s\", "
2456                                "from the execution profile \"%s\".", value, buf);
2457         goto out;
2458     }
2459 }
2460
2461 if ((value = kva_match(eap->attr, EXECATTR_LPRIV_KW)) != NULL) {
2462     ci->lpriv_set = priv_str_to_set(value, "", NULL);
2463     if (ci->lpriv_set == NULL) {
2464         if (errno != EINVAL)
2465             err = mc_error_create(merr, ENOMEM,
2466                                    ALLOCFAIL);
2467         else
2468             err = mc_error_create(merr, EINVAL,
2469                                    "Could not interpret profile "
2470                                    "limitprivs value \"%s\", from "
2471                                    "the execution profile \"%s\".",
2472                                    value, buf);
2473         goto out;
2474     }
2475 }
2476
2477 if ((value = kva_match(eap->attr, EXECATTR_IPRIV_KW)) != NULL) {
2478     ci->priv_set = priv_str_to_set(value, "", NULL);
2479     if (ci->priv_set == NULL) {
2480         if (errno != EINVAL)
2481             err = mc_error_create(merr, ENOMEM,
2482                                    ALLOCFAIL);
2483         else
2484             err = mc_error_create(merr, EINVAL,
2485                                    "Could not interpret profile privs value "
2486                                    "\"%s\", from the execution profile "
2487                                    "\"%s\".", value, buf);
2488         goto out;
2489     }
2490 }
2491
2492 out:
2493     free_execattr(eap);
2494
2495     return (err);
2496 }
2497
2498 /*
2499 * Return an error message structure containing the error message
2500 * with context, and the error so the caller can make a decision

```

```

2501 * on what to do next.
2502 *
2503 * Because get_ids uses the mc_error_create() function which can
2504 * reallocate the merr, this function must return the merr pointer
2505 * in case it was reallocated.
2506 */
2507 static mc_error_t *
2508 get_ids(scf_propertygroup_t *methpg, scf_propertygroup_t *instpg,
2509        scf_property_t *prop, scf_value_t *val, struct method_context *ci,
2510        mc_error_t *merr)
2511 {
2512     char *vbuf = ci->vbuf;
2513     ssize_t vbuf_sz = ci->vbuf_sz;
2514     int r;
2515
2516     /*
2517     * This should never happen because the caller should fall through
2518     * another path of just setting the ids to defaults, instead of
2519     * attempting to get the ids here.
2520     */
2521     if (methpg == NULL && instpg == NULL)
2522         return (mc_error_create(merr, ENOENT,
2523                                "No property groups to get ids from."));
2524
2525     if (!(get_astring_val(methpg, SCF_PROPERTY_USER,
2526                          vbuf, vbuf_sz, prop, val) == 0 || get_astring_val(instpg,
2527                                   SCF_PROPERTY_USER, vbuf, vbuf_sz, prop,
2528                                   val) == 0))
2529         return (mc_error_create(merr, ENOENT,
2530                                "Could not get \"%s\" property.", SCF_PROPERTY_USER));
2531
2532     if ((r = get_uid(vbuf, ci, &ci->uid) != 0) {
2533         ci->uid = (uid_t)-1;
2534         return (mc_error_create(merr, r,
2535                                "Could not interpret \"%s\" property value \"%s\", \"
2536                                \"error %d.\", SCF_PROPERTY_USER, vbuf, r));
2537     }
2538
2539     if (!(get_astring_val(methpg, SCF_PROPERTY_GROUP, vbuf, vbuf_sz, prop,
2540                          val) == 0 || get_astring_val(instpg, SCF_PROPERTY_GROUP, vbuf,
2541                                   vbuf_sz, prop, val) == 0)) {
2542         if (scf_error() == SCF_ERROR_NOT_FOUND) {
2543             (void) strcpy(vbuf, ":default");
2544         } else {
2545             return (mc_error_create(merr, ENOENT,
2546                                    "Could not get \"%s\" property.",
2547                                    SCF_PROPERTY_GROUP));
2548         }
2549     }
2550
2551     if (strcmp(vbuf, ":default") != 0) {
2552         ci->gid = get_gid(vbuf);
2553         if (ci->gid == (gid_t)-1) {
2554             return (mc_error_create(merr, ENOENT,
2555                                    "Could not interpret \"%s\" property value \"%s\".",
2556                                    SCF_PROPERTY_GROUP, vbuf));
2557         }
2558     } else {
2559         switch (r = lookup_pwd(ci)) {
2560         case 0:
2561             ci->gid = ci->pwd.pw_gid;
2562             break;
2563
2564         case ENOENT:
2565             ci->gid = (gid_t)-1;
2566             return (mc_error_create(merr, ENOENT,

```

```

2567         "No passwd entry for uid \"%d\".", ci->uid));
2568
2569     case ENOMEM:
2570         return (mc_error_create(merr, ENOMEM,
2571                                "Out of memory."));
2572
2573     case EIO:
2574     case EMFILE:
2575     case ENFILE:
2576         return (mc_error_create(merr, ENFILE,
2577                                "getpwuid_r() failed, error %d.", r));
2578
2579     default:
2580         bad_fail("lookup_pwd", r);
2581     }
2582 }
2583
2584 if (!(get_astring_val(methpg, SCF_PROPERTY_SUPP_GROUPS, vbuf, vbuf_sz,
2585                      prop, val) == 0 || get_astring_val(instpg,
2586                                   SCF_PROPERTY_SUPP_GROUPS, vbuf, vbuf_sz, prop, val) == 0)) {
2587     if (scf_error() == SCF_ERROR_NOT_FOUND) {
2588         (void) strcpy(vbuf, ":default");
2589     } else {
2590         return (mc_error_create(merr, ENOENT,
2591                                "Could not get supplemental groups (\"%s\") \"
2592                                \"property.\", SCF_PROPERTY_SUPP_GROUPS));
2593     }
2594 }
2595
2596 if (strcmp(vbuf, ":default") != 0) {
2597     switch (r = get_groups(vbuf, ci)) {
2598     case 0:
2599         break;
2600
2601     case EINVAL:
2602         return (mc_error_create(merr, EINVAL,
2603                                "Could not interpret supplemental groups (\"%s\") \"
2604                                \"property value \"%s\".", SCF_PROPERTY_SUPP_GROUPS,
2605                                vbuf));
2606
2607     case E2BIG:
2608         return (mc_error_create(merr, E2BIG,
2609                                "Too many supplemental groups values in \"%s\".",
2610                                vbuf));
2611
2612     default:
2613         bad_fail("get_groups", r);
2614     }
2615 } else {
2616     ci->ngroups = -1;
2617 }
2618
2619 if (!(get_astring_val(methpg, SCF_PROPERTY_PRIVILEGES, vbuf, vbuf_sz,
2620                      prop, val) == 0 || get_astring_val(instpg, SCF_PROPERTY_PRIVILEGES,
2621                                   vbuf, vbuf_sz, prop, val) == 0)) {
2622     if (scf_error() == SCF_ERROR_NOT_FOUND) {
2623         (void) strcpy(vbuf, ":default");
2624     } else {
2625         return (mc_error_create(merr, ENOENT,
2626                                "Could not get \"%s\" property.\",
2627                                SCF_PROPERTY_PRIVILEGES));
2628     }
2629 }
2630
2631 /*
2632 * For default privs, we need to keep priv_set == NULL, as

```

```

2633     * we use this test elsewhere.
2634     */
2635     if (strcmp(vbuf, ":default") != 0) {
2636         ci->priv_set = priv_str_to_set(vbuf, ",", NULL);
2637         if (ci->priv_set == NULL) {
2638             if (errno != EINVAL) {
2639                 return (mc_error_create(merr, ENOMEM,
2640                     ALLOCFAIL));
2641             } else {
2642                 return (mc_error_create(merr, EINVAL,
2643                     "Could not interpret \"%s\" "
2644                     "property value \"%s\".",
2645                     SCF_PROPERTY_PRIVILEGES, vbuf));
2646             }
2647         }
2648     }
2649
2650     if (!(get_astring_val(methpg, SCF_PROPERTY_LIMIT_PRIVILEGES, vbuf,
2651         vbuf_sz, prop, val) == 0 || get_astring_val(instpg,
2652         SCF_PROPERTY_LIMIT_PRIVILEGES, vbuf, vbuf_sz, prop, val) == 0)) {
2653         if (scf_error() == SCF_ERROR_NOT_FOUND) {
2654             (void) strcpy(vbuf, ":default");
2655         } else {
2656             return (mc_error_create(merr, ENOENT,
2657                 "Could not get \"%s\" property.",
2658                 SCF_PROPERTY_LIMIT_PRIVILEGES));
2659         }
2660     }
2661
2662     if (strcmp(vbuf, ":default") == 0)
2663         /*
2664         * L must default to all privileges so root NPA services see
2665         * iE = all. "zone" is all privileges available in the current
2666         * zone, equivalent to "all" in the global zone.
2667         */
2668         (void) strcpy(vbuf, "zone");
2669
2670     ci->lpriv_set = priv_str_to_set(vbuf, ",", NULL);
2671     if (ci->lpriv_set == NULL) {
2672         if (errno != EINVAL) {
2673             return (mc_error_create(merr, ENOMEM, ALLOCFAIL));
2674         } else {
2675             return (mc_error_create(merr, EINVAL,
2676                 "Could not interpret \"%s\" property value \"%s\".",
2677                 SCF_PROPERTY_LIMIT_PRIVILEGES, vbuf));
2678         }
2679     }
2680
2681     return (merr);
2682 }
2683
2684 static int
2685 get_environment(scf_handle_t *h, scf_propertygroup_t *pg,
2686     struct method_context *mcp, scf_property_t *prop, scf_value_t *val)
2687 {
2688     scf_iter_t *iter;
2689     scf_type_t type;
2690     size_t i = 0;
2691     int ret;
2692
2693     if (scf_pg_get_property(pg, SCF_PROPERTY_ENVIRONMENT, prop) != 0) {
2694         if (scf_error() == SCF_ERROR_NOT_FOUND)
2695             return (ENOENT);
2696         return (scf_error());
2697     }
2698     if (scf_property_type(prop, &type) != 0)

```

```

2699         return (scf_error());
2700     if (type != SCF_TYPE_ASTRING)
2701         return (EINVAL);
2702     if ((iter = scf_iter_create(h)) == NULL)
2703         return (scf_error());
2704
2705     if (scf_iter_property_values(iter, prop) != 0) {
2706         ret = scf_error();
2707         scf_iter_destroy(iter);
2708         return (ret);
2709     }
2710
2711     mcp->env_sz = 10;
2712
2713     if ((mcp->env = uu_zalloc(sizeof (*mcp->env) * mcp->env_sz)) == NULL) {
2714         ret = ENOMEM;
2715         goto out;
2716     }
2717
2718     while ((ret = scf_iter_next_value(iter, val)) == 1) {
2719         ret = scf_value_get_as_string(val, mcp->vbuf, mcp->vbuf_sz);
2720         if (ret == -1) {
2721             ret = scf_error();
2722             goto out;
2723         }
2724
2725         if ((mcp->env[i] = strdup(mcp->vbuf)) == NULL) {
2726             ret = ENOMEM;
2727             goto out;
2728         }
2729
2730         if (++i == mcp->env_sz) {
2731             char **env;
2732             mcp->env_sz *= 2;
2733             env = uu_zalloc(sizeof (*mcp->env) * mcp->env_sz);
2734             if (env == NULL) {
2735                 ret = ENOMEM;
2736                 goto out;
2737             }
2738             (void) memcpy(env, mcp->env,
2739                 sizeof (*mcp->env) * (mcp->env_sz / 2));
2740             free(mcp->env);
2741             mcp->env = env;
2742         }
2743     }
2744
2745     if (ret == -1)
2746         ret = scf_error();
2747
2748 out:
2749     scf_iter_destroy(iter);
2750     return (ret);
2751 }
2752
2753 /*
2754 * Fetch method context information from the repository, allocate and fill
2755 * a method_context structure, return it in *mcp, and return NULL.
2756 *
2757 * If no method_context is defined, original init context is provided, where
2758 * the working directory is '/', and uid/gid are 0/0. But if a method_context
2759 * is defined at any level the smf_method(5) method_context defaults are used.
2760 *
2761 * Return an error message structure containing the error message
2762 * with context, and the error so the caller can make a decision
2763 * on what to do next.
2764 *

```

```

2765 * Error Types :
2766 *   E2BIG           Too many values or entry is too big
2767 *   EINVAL         Invalid value
2768 *   EIO            an I/O error has occurred
2769 *   ENOENT        no entry for value
2770 *   ENOMEM        out of memory
2771 *   ENOTSUP       Version mismatch
2772 *   ERANGE        value is out of range
2773 *   EMFILE/ENFILE out of file descriptors
2774 *
2775 *   SCF_ERROR_BACKEND_ACCESS
2776 *   SCF_ERROR_CONNECTION_BROKEN
2777 *   SCF_ERROR_DELETED
2778 *   SCF_ERROR_CONSTRAINT_VIOLATED
2779 *   SCF_ERROR_HANDLE_DESTROYED
2780 *   SCF_ERROR_INTERNAL
2781 *   SCF_ERROR_INVALID_ARGUMENT
2782 *   SCF_ERROR_NO_MEMORY
2783 *   SCF_ERROR_NO_RESOURCES
2784 *   SCF_ERROR_NOT_BOUND
2785 *   SCF_ERROR_NOT_FOUND
2786 *   SCF_ERROR_NOT_SET
2787 *   SCF_ERROR_TYPE_MISMATCH
2788 *
2789 */
2790 mc_error_t *
2791 restarter_get_method_context(uint_t version, scf_instance_t *inst,
2792 scf_snapshot_t *snap, const char *mname, const char *cmdline,
2793 struct method_context **mcpp)
2794 {
2795     scf_handle_t *h;
2796     scf_propertygroup_t *methpg = NULL;
2797     scf_propertygroup_t *instpg = NULL;
2798     scf_propertygroup_t *pg = NULL;
2799     scf_property_t *prop = NULL;
2800     scf_value_t *val = NULL;
2801     scf_type_t ty;
2802     uint8_t use_profile;
2803     int ret = 0;
2804     int mc_used = 0;
2805     mc_error_t *err = NULL;
2806     struct method_context *cip;
2807
2808     if ((err = malloc(sizeof (mc_error_t))) == NULL)
2809         return (mc_error_create(NULL, ENOMEM, NULL));
2810
2811     /* Set the type to zero to track if an error occurred. */
2812     err->type = 0;
2813
2814     if (version != RESTARTER_METHOD_CONTEXT_VERSION)
2815         return (mc_error_create(err, ENOTSUP,
2816 "Invalid client version %d. (Expected %d)",
2817 version, RESTARTER_METHOD_CONTEXT_VERSION));
2818
2819     /* Get the handle before we allocate anything. */
2820     h = scf_instance_handle(inst);
2821     if (h == NULL)
2822         return (mc_error_create(err, scf_error(),
2823 scf_strerror(scf_error())));
2824
2825     cip = malloc(sizeof (*cip));
2826     if (cip == NULL)
2827         return (mc_error_create(err, ENOMEM, ALLOCFAIL));
2828
2829     (void) memset(cip, 0, sizeof (*cip));
2830     cip->uid = (uid_t)-1;

```

```

2831     cip->euid = (uid_t)-1;
2832     cip->gid = (gid_t)-1;
2833     cip->egid = (gid_t)-1;
2834
2835     cip->vbuf_sz = scf_limit(SCF_LIMIT_MAX_VALUE_LENGTH);
2836     assert(cip->vbuf_sz >= 0);
2837     cip->vbuf = malloc(cip->vbuf_sz);
2838     if (cip->vbuf == NULL) {
2839         free(cip);
2840         return (mc_error_create(err, ENOMEM, ALLOCFAIL));
2841     }
2842
2843     if ((instpg = scf_pg_create(h)) == NULL ||
2844         (methpg = scf_pg_create(h)) == NULL ||
2845         (prop = scf_property_create(h)) == NULL ||
2846         (val = scf_value_create(h)) == NULL) {
2847         err = mc_error_create(err, scf_error(),
2848 "Failed to create repository object: %s",
2849 "Failed to create repository object: %s\n",
2850 scf_strerror(scf_error()));
2851         goto out;
2852     }
2853
2854     /*
2855      * The method environment, and the credentials/profile data,
2856      * may be found either in the pg for the method (methpg),
2857      * or in the instance/service SCF_PG_METHOD_CONTEXT pg (named
2858      * instpg below).
2859      */
2860     if (scf_instance_get_pg_composed(inst, snap, mname, methpg) !=
2861         SCF_SUCCESS) {
2862         err = mc_error_create(err, scf_error(), "Unable to get the "
2863 "\"%s\" method, %s", mname, scf_strerror(scf_error()));
2864         goto out;
2865     }
2866
2867     if (scf_instance_get_pg_composed(inst, snap, SCF_PG_METHOD_CONTEXT,
2868 instpg) != SCF_SUCCESS) {
2869         if (scf_error() != SCF_ERROR_NOT_FOUND) {
2870             err = mc_error_create(err, scf_error(),
2871 "Unable to retrieve the \"%s\" property group, %s",
2872 SCF_PG_METHOD_CONTEXT, scf_strerror(scf_error()));
2873             goto out;
2874         }
2875         scf_pg_destroy(instpg);
2876         instpg = NULL;
2877     } else {
2878         mc_used++;
2879     }
2880
2881     ret = get_environment(h, methpg, cip, prop, val);
2882     if (ret == ENOENT && instpg != NULL) {
2883         ret = get_environment(h, instpg, cip, prop, val);
2884     }
2885
2886     switch (ret) {
2887     case 0:
2888         mc_used++;
2889         break;
2890     case ENOENT:
2891         break;
2892     case ENOMEM:
2893         err = mc_error_create(err, ret, "Out of memory.");
2894         goto out;
2895     case EINVAL:

```

```

2896         err = mc_error_create(err, ret, "Invalid method environment.");
2897         goto out;
2898     default:
2899         err = mc_error_create(err, ret,
2900             "Get method environment failed: %s", scf_strerror(ret));
2901         "Get method environment failed : %s\n", scf_strerror(ret));
2902         goto out;
2903     }
2904     pg = methpg;
2905
2906     ret = scf_pg_get_property(pg, SCF_PROPERTY_USE_PROFILE, prop);
2907     if (ret && scf_error() == SCF_ERROR_NOT_FOUND && instpg != NULL) {
2908         pg = NULL;
2909         ret = scf_pg_get_property(instpg, SCF_PROPERTY_USE_PROFILE,
2910             prop);
2911     }
2912
2913     if (ret) {
2914         switch (scf_error()) {
2915         case SCF_ERROR_NOT_FOUND:
2916             /* No profile context: use default credentials */
2917             cip->uid = 0;
2918             cip->gid = 0;
2919             break;
2920
2921         case SCF_ERROR_CONNECTION_BROKEN:
2922             err = mc_error_create(err, SCF_ERROR_CONNECTION_BROKEN,
2923                 RCBROKEN);
2924             goto out;
2925
2926         case SCF_ERROR_DELETED:
2927             err = mc_error_create(err, SCF_ERROR_NOT_FOUND,
2928                 "Could not find property group \"%s\"",
2929                 pg == NULL ? SCF_PG_METHOD_CONTEXT : mname);
2930             goto out;
2931
2932         case SCF_ERROR_HANDLE_MISMATCH:
2933         case SCF_ERROR_INVALID_ARGUMENT:
2934         case SCF_ERROR_NOT_SET:
2935             default:
2936                 bad_fail("scf_pg_get_property", scf_error());
2937         }
2938     } else {
2939         if (scf_property_type(prop, &ty) != SCF_SUCCESS) {
2940             ret = scf_error();
2941             switch (ret) {
2942             case SCF_ERROR_CONNECTION_BROKEN:
2943                 err = mc_error_create(err,
2944                     SCF_ERROR_CONNECTION_BROKEN, RCBROKEN);
2945                 break;
2946
2947             case SCF_ERROR_DELETED:
2948                 err = mc_error_create(err,
2949                     SCF_ERROR_NOT_FOUND,
2950                     "Could not find property group \"%s\"",
2951                     pg == NULL ? SCF_PG_METHOD_CONTEXT : mname);
2952                 break;
2953
2954             case SCF_ERROR_NOT_SET:
2955             default:
2956                 bad_fail("scf_property_type", ret);
2957             }
2958
2959             goto out;
2960         }

```

```

2962         if (ty != SCF_TYPE_BOOLEAN) {
2963             err = mc_error_create(err,
2964                 SCF_ERROR_TYPE_MISMATCH,
2965                 "\"%s\" property is not boolean in property group \"
2966                 \"%s\".", SCF_PROPERTY_USE_PROFILE,
2967                 pg == NULL ? SCF_PG_METHOD_CONTEXT : mname);
2968             goto out;
2969         }
2970
2971         if (scf_property_get_value(prop, val) != SCF_SUCCESS) {
2972             ret = scf_error();
2973             switch (ret) {
2974             case SCF_ERROR_CONNECTION_BROKEN:
2975                 err = mc_error_create(err,
2976                     SCF_ERROR_CONNECTION_BROKEN, RCBROKEN);
2977                 break;
2978
2979             case SCF_ERROR_CONSTRAINT_VIOLATED:
2980                 err = mc_error_create(err,
2981                     SCF_ERROR_CONSTRAINT_VIOLATED,
2982                     "\"%s\" property has multiple values.",
2983                     SCF_PROPERTY_USE_PROFILE);
2984                 break;
2985
2986             case SCF_ERROR_NOT_FOUND:
2987                 err = mc_error_create(err,
2988                     SCF_ERROR_NOT_FOUND,
2989                     "\"%s\" property has no values.",
2990                     SCF_PROPERTY_USE_PROFILE);
2991                 break;
2992             default:
2993                 bad_fail("scf_property_get_value", ret);
2994             }
2995
2996             goto out;
2997         }
2998
2999         mc_used++;
3000         ret = scf_value_get_boolean(val, &use_profile);
3001         assert(ret == SCF_SUCCESS);
3002
3003         /* get ids & privileges */
3004         if (use_profile)
3005             err = get_profile(pg, instpg, prop, val, cmdline,
3006                 cip, err);
3007         else
3008             err = get_ids(pg, instpg, prop, val, cip, err);
3009
3010         if (err->type != 0)
3011             goto out;
3012     }
3013
3014     /* get working directory */
3015     if ((methpg != NULL && scf_pg_get_property(methpg,
3016         SCF_PROPERTY_WORKING_DIRECTORY, prop) == SCF_SUCCESS) ||
3017         (instpg != NULL && scf_pg_get_property(instpg,
3018         SCF_PROPERTY_WORKING_DIRECTORY, prop) == SCF_SUCCESS)) {
3019         if (scf_property_get_value(prop, val) != SCF_SUCCESS) {
3020             ret = scf_error();
3021             switch (ret) {
3022             case SCF_ERROR_CONNECTION_BROKEN:
3023                 err = mc_error_create(err, ret, RCBROKEN);
3024                 break;
3025
3026             case SCF_ERROR_CONSTRAINT_VIOLATED:

```

```

3027         err = mc_error_create(err, ret,
3028             "\\%s\\" property has multiple values.",
3029             SCF_PROPERTY_WORKING_DIRECTORY);
3030         break;

3032     case SCF_ERROR_NOT_FOUND:
3033         err = mc_error_create(err, ret,
3034             "\\%s\\" property has no values.",
3035             SCF_PROPERTY_WORKING_DIRECTORY);
3036         break;

3038     default:
3039         bad_fail("scf_property_get_value", ret);
3040     }

3042     goto out;
3043 }

3045     mc_used++;
3046     ret = scf_value_get_astring(val, cip->vbuf, cip->vbuf_sz);
3047     assert(ret != -1);
3048 } else {
3049     ret = scf_error();
3050     switch (ret) {
3051     case SCF_ERROR_NOT_FOUND:
3052         /* okay if missing. */
3053         (void) strcpy(cip->vbuf, ":default");
3054         break;

3056     case SCF_ERROR_CONNECTION_BROKEN:
3057         err = mc_error_create(err, ret, RCBROKEN);
3058         goto out;

3060     case SCF_ERROR_DELETED:
3061         err = mc_error_create(err, ret,
3062             "Property group could not be found");
3063         goto out;

3065     case SCF_ERROR_HANDLE_MISMATCH:
3066     case SCF_ERROR_INVALID_ARGUMENT:
3067     case SCF_ERROR_NOT_SET:
3068     default:
3069         bad_fail("scf_pg_get_property", ret);
3070     }
3071 }

3073 if (strcmp(cip->vbuf, ":default") == 0 ||
3074     strcmp(cip->vbuf, ":home") == 0) {
3075     switch (ret = lookup_pwd(cip)) {
3076     case 0:
3077         break;

3079     case ENOMEM:
3080         err = mc_error_create(err, ret, "Out of memory.");
3081         goto out;

3083     case ENOENT:
3084     case EIO:
3085     case EMFILE:
3086     case ENFILE:
3087         err = mc_error_create(err, ret,
3088             "Could not get passwd entry.");
3089         goto out;

3091     default:
3092         bad_fail("lookup_pwd", ret);

```

```

3093     }

3095     cip->working_dir = strdup(cip->pwd.pw_dir);
3096     if (cip->working_dir == NULL) {
3097         err = mc_error_create(err, ENOMEM, ALLOCFAIL);
3098         goto out;
3099     }
3100 } else {
3101     cip->working_dir = strdup(cip->vbuf);
3102     if (cip->working_dir == NULL) {
3103         err = mc_error_create(err, ENOMEM, ALLOCFAIL);
3104         goto out;
3105     }
3106 }

3108 /* get security flags */
3109 if ((methpg != NULL && scf_pg_get_property(methpg,
3110     SCF_PROPERTY_SECFLAGS, prop) == SCF_SUCCESS) ||
3111     (instpg != NULL && scf_pg_get_property(instpg,
3112     SCF_PROPERTY_SECFLAGS, prop) == SCF_SUCCESS)) {
3113     if (scf_property_get_value(prop, val) != SCF_SUCCESS) {
3114         ret = scf_error();
3115         switch (ret) {
3116         case SCF_ERROR_CONNECTION_BROKEN:
3117             err = mc_error_create(err, ret, RCBROKEN);
3118             break;

3120         case SCF_ERROR_CONSTRAINT_VIOLATED:
3121             err = mc_error_create(err, ret,
3122                 "\\%s\\" property has multiple values.",
3123                 SCF_PROPERTY_SECFLAGS);
3124             break;

3126         case SCF_ERROR_NOT_FOUND:
3127             err = mc_error_create(err, ret,
3128                 "\\%s\\" property has no values.",
3129                 SCF_PROPERTY_SECFLAGS);
3130             break;

3132         default:
3133             bad_fail("scf_property_get_value", ret);
3134         }
3135     }
3136 } else {
3137     (void) strcpy(cip->vbuf, ":default", cip->vbuf_sz);
3138 }
3139     ret = scf_value_get_astring(val, cip->vbuf,
3140     cip->vbuf_sz);
3141     assert(ret != -1);
3142     mc_used++;
3143 } else {
3144     ret = scf_error();
3145     switch (ret) {
3146     case SCF_ERROR_NOT_FOUND:
3147         /* okay if missing. */
3148         (void) strcpy(cip->vbuf, ":default", cip->vbuf_sz);
3149         break;

3151     case SCF_ERROR_CONNECTION_BROKEN:
3152         err = mc_error_create(err, ret, RCBROKEN);
3153         goto out;

3155     case SCF_ERROR_DELETED:
3156         err = mc_error_create(err, ret,
3157             "Property group could not be found");
3158         goto out;

```

```

3160         case SCF_ERROR_HANDLE_MISMATCH:
3161         case SCF_ERROR_INVALID_ARGUMENT:
3162         case SCF_ERROR_NOT_SET:
3163         default:
3164             bad_fail("scf_pg_get_property", ret);
3165     }
3166 }

3169 if (scf_default_secflags(h, &cip->def_secflags) != 0) {
3170     err = mc_error_create(err, EINVAL, "couldn't fetch "
3171         "default security-flags");
3172     goto out;
3173 }

3175 if (strcmp(cip->vbuf, ":default") == 0) {
3176     if (secflags_parse(&cip->def_secflags.psf_inherit, "default",
3177         &cip->secflag_delta) != 0) {
3178         err = mc_error_create(err, EINVAL, "couldn't parse "
3179             "security flags: %s", cip->vbuf);
3180         goto out;
3181     }
3182 } else {
3183     if (secflags_parse(&cip->def_secflags.psf_inherit, cip->vbuf,
3184         &cip->secflag_delta) != 0) {
3185         err = mc_error_create(err, EINVAL, "couldn't parse "
3186             "security flags: %s", cip->vbuf);
3187         goto out;
3188     }
3189 }

3191 #endif /* ! codereview */
3192 /* get (optional) corefile pattern */
3193 if ((methpg != NULL && scf_pg_get_property(methpg,
3194     SCF_PROPERTY_COREFILE_PATTERN, prop) == SCF_SUCCESS) ||
3195     (instpg != NULL && scf_pg_get_property(instpg,
3196     SCF_PROPERTY_COREFILE_PATTERN, prop) == SCF_SUCCESS)) {
3197     if (scf_property_get_value(prop, val) != SCF_SUCCESS) {
3198         ret = scf_error();
3199         switch (ret) {
3200         case SCF_ERROR_CONNECTION_BROKEN:
3201             err = mc_error_create(err, ret, RCBROKEN);
3202             break;

3204         case SCF_ERROR_CONSTRAINT_VIOLATED:
3205             err = mc_error_create(err, ret,
3206                 "\"%s\" property has multiple values.",
3207                 SCF_PROPERTY_COREFILE_PATTERN);
3208             break;

3210         case SCF_ERROR_NOT_FOUND:
3211             err = mc_error_create(err, ret,
3212                 "\"%s\" property has no values.",
3213                 SCF_PROPERTY_COREFILE_PATTERN);
3214             break;

3216         default:
3217             bad_fail("scf_property_get_value", ret);
3218     }

3220 } else {

3222     ret = scf_value_get_astring(val, cip->vbuf,
3223         cip->vbuf_sz);
3224     assert(ret != -1);

```

```

3226         cip->corefile_pattern = strdup(cip->vbuf);
3227         if (cip->corefile_pattern == NULL) {
3228             err = mc_error_create(err, ENOMEM, ALLOCFAIL);
3229             goto out;
3230         }
3231     }

3233     mc_used++;
3234 } else {
3235     ret = scf_error();
3236     switch (ret) {
3237     case SCF_ERROR_NOT_FOUND:
3238         /* okay if missing. */
3239         break;

3241     case SCF_ERROR_CONNECTION_BROKEN:
3242         err = mc_error_create(err, ret, RCBROKEN);
3243         goto out;

3245     case SCF_ERROR_DELETED:
3246         err = mc_error_create(err, ret,
3247             "Property group could not be found");
3248         goto out;

3250     case SCF_ERROR_HANDLE_MISMATCH:
3251     case SCF_ERROR_INVALID_ARGUMENT:
3252     case SCF_ERROR_NOT_SET:
3253     default:
3254         bad_fail("scf_pg_get_property", ret);
3255     }

3258 if (restarter_rm_libs_loadable()) {
3259     /* get project */
3260     if ((methpg != NULL && scf_pg_get_property(methpg,
3261         SCF_PROPERTY_PROJECT, prop) == SCF_SUCCESS) ||
3262         (instpg != NULL && scf_pg_get_property(instpg,
3263         SCF_PROPERTY_PROJECT, prop) == SCF_SUCCESS)) {
3264         if (scf_property_get_value(prop, val) != SCF_SUCCESS) {
3265             ret = scf_error();
3266             switch (ret) {
3267             case SCF_ERROR_CONNECTION_BROKEN:
3268                 err = mc_error_create(err, ret,
3269                     RCBROKEN);
3270                 break;

3272             case SCF_ERROR_CONSTRAINT_VIOLATED:
3273                 err = mc_error_create(err, ret,
3274                     "\"%s\" property has multiple "
3275                     "values.", SCF_PROPERTY_PROJECT);
3276                 break;

3278             case SCF_ERROR_NOT_FOUND:
3279                 err = mc_error_create(err, ret,
3280                     "\"%s\" property has no values.",
3281                     SCF_PROPERTY_PROJECT);
3282                 break;

3284             default:
3285                 bad_fail("scf_property_get_value", ret);
3286         }

3288         (void) strcpy(cip->vbuf, ":default");
3289     } else {
3290         ret = scf_value_get_astring(val, cip->vbuf,

```



```

3291         cip->vbuf_sz);
3292         assert(ret != -1);
3293     }
3294
3295     mc_used++;
3296 } else {
3297     (void) strcpy(cip->vbuf, ":default");
3298 }
3299
3300 switch (ret = get_projid(cip->vbuf, cip)) {
3301 case 0:
3302     break;
3303
3304 case ENOMEM:
3305     err = mc_error_create(err, ret, "Out of memory.");
3306     goto out;
3307
3308 case ENOENT:
3309     err = mc_error_create(err, ret,
3310         "Missing passwd or project entry for \"%s\".",
3311         cip->vbuf);
3312     goto out;
3313
3314 case EIO:
3315     err = mc_error_create(err, ret, "I/O error.");
3316     goto out;
3317
3318 case EMFILE:
3319 case ENFILE:
3320     err = mc_error_create(err, ret,
3321         "Out of file descriptors.");
3322     goto out;
3323
3324 case -1:
3325     err = mc_error_create(err, ret,
3326         "Name service switch is misconfigured.");
3327     goto out;
3328
3329 case ERANGE:
3330 case E2BIG:
3331     err = mc_error_create(err, ret,
3332         "Project ID \"%s\" too big.", cip->vbuf);
3333     goto out;
3334
3335 case EINVAL:
3336     err = mc_error_create(err, ret,
3337         "Project ID \"%s\" is invalid.", cip->vbuf);
3338     goto out;
3339
3340 default:
3341     bad_fail("get_projid", ret);
3342 }
3343
3344 /* get resource pool */
3345 if ((methpg != NULL && scf_pg_get_property(methpg,
3346     SCF_PROPERTY_RESOURCE_POOL, prop) == SCF_SUCCESS) ||
3347     (instpg != NULL && scf_pg_get_property(instpg,
3348     SCF_PROPERTY_RESOURCE_POOL, prop) == SCF_SUCCESS)) {
3349     if (scf_property_get_value(prop, val) != SCF_SUCCESS) {
3350         ret = scf_error();
3351         switch (ret) {
3352             case SCF_ERROR_CONNECTION_BROKEN:
3353                 err = mc_error_create(err, ret,
3354                     RCBROKEN);
3355                 break;

```

```

3357     case SCF_ERROR_CONSTRAINT_VIOLATED:
3358         err = mc_error_create(err, ret,
3359             "\"%s\" property has multiple "
3360             "values.",
3361             SCF_PROPERTY_RESOURCE_POOL);
3362         break;
3363
3364     case SCF_ERROR_NOT_FOUND:
3365         err = mc_error_create(err, ret,
3366             "\"%s\" property has no "
3367             "values.",
3368             SCF_PROPERTY_RESOURCE_POOL);
3369         break;
3370
3371     default:
3372         bad_fail("scf_property_get_value", ret);
3373 }
3374
3375 (void) strcpy(cip->vbuf, ":default");
3376 } else {
3377     ret = scf_value_get_astring(val, cip->vbuf,
3378         cip->vbuf_sz);
3379     assert(ret != -1);
3380 }
3381
3382 mc_used++;
3383 } else {
3384     ret = scf_error();
3385     switch (ret) {
3386     case SCF_ERROR_NOT_FOUND:
3387         /* okay if missing. */
3388         (void) strcpy(cip->vbuf, ":default");
3389         break;
3390
3391     case SCF_ERROR_CONNECTION_BROKEN:
3392         err = mc_error_create(err, ret, RCBROKEN);
3393         goto out;
3394
3395     case SCF_ERROR_DELETED:
3396         err = mc_error_create(err, ret,
3397             "property group could not be found.");
3398         goto out;
3399
3400     case SCF_ERROR_HANDLE_MISMATCH:
3401     case SCF_ERROR_INVALID_ARGUMENT:
3402     case SCF_ERROR_NOT_SET:
3403     default:
3404         bad_fail("scf_pg_get_property", ret);
3405     }
3406 }
3407
3408 if (strcmp(cip->vbuf, ":default") != 0) {
3409     cip->resource_pool = strdup(cip->vbuf);
3410     if (cip->resource_pool == NULL) {
3411         err = mc_error_create(err, ENOMEM, ALLOCFAIL);
3412         goto out;
3413     }
3414 }
3415 }
3416
3417 /*
3418  * A method_context was not used for any configurable
3419  * elements or attributes, so reset and use the simple
3420  * defaults that provide historic init behavior.
3421  */
3422 if (mc_used == 0) {

```

```

3423         free(cip->pwbuf);
3424         free(cip->vbuf);
3425         free(cip->working_dir);

3427         (void) memset(cip, 0, sizeof (*cip));
3428         cip->uid = 0;
3429         cip->gid = 0;
3430         cip->euid = (uid_t)-1;
3431         cip->egid = (gid_t)-1;

3433         if (scf_default_secflags(h, &cip->def_secflags) != 0) {
3434             err = mc_error_create(err, EINVAL, "couldn't fetch "
3435                 "default security-flags");
3436             goto out;
3437         }

3439         if (secflags_parse(&cip->def_secflags.psf_inherit, "default",
3440             &cip->secflag_delta) != 0) {
3441             err = mc_error_create(err, EINVAL, "couldn't parse "
3442                 "security flags: %s", cip->vbuf);
3443             goto out;
3444         }
3445 #endif /* ! codereview */
3446     }

3448     *mcpp = cip;

3450 out:
3451     (void) scf_value_destroy(val);
3452     scf_property_destroy(prop);
3453     scf_pg_destroy(instpg);
3454     scf_pg_destroy(methpg);

3456     if (cip->pwbuf != NULL) {
3457         free(cip->pwbuf);
3458         cip->pwbuf = NULL;
3459     }

3461     free(cip->vbuf);

3463     if (err->type != 0) {
3464         restarter_free_method_context(cip);
3465     } else {
3466         restarter_mc_error_destroy(err);
3467         err = NULL;
3468     }

3470     return (err);
3471 }

3473 /*
3474  * Modify the current process per the given method_context.  On success, returns
3475  * 0.  Note that the environment is not modified by this function to include the
3476  * environment variables in cip->env.
3477  *
3478  * On failure, sets *fp to NULL or the name of the function which failed,
3479  * and returns one of the following error codes.  The words in parentheses are
3480  * the values to which *fp may be set for the error case.
3481  * ENOMEM - malloc() failed
3482  * EIO - an I/O error occurred (getpwuid_r, chdir)
3483  * EMFILE - process is out of file descriptors (getpwuid_r)
3484  * ENFILE - system is out of file handles (getpwuid_r)
3485  * EINVAL - gid or egid is out of range (setregid)
3486  *          ngroups is too big (setgroups)
3487  *          project's project id is bad (setproject)
3488  *          uid or euid is out of range (setreuid)

```

```

3489  *          poolname is invalid (pool_set_binding)
3490  * EPERM - insufficient privilege (setregid, initgroups, setgroups, setppriv,
3491  *         setproject, setreuid, settaskid)
3492  * ENOENT - uid has a passwd entry but no shadow entry
3493  *         working_dir does not exist (chdir)
3494  *         uid has no passwd entry
3495  *         the pool could not be found (pool_set_binding)
3496  * EFAULT - lpriv_set or priv_set has a bad address (setppriv)
3497  *         working_dir has a bad address (chdir)
3498  * EACCES - could not access working_dir (chdir)
3499  *         in a TASK_FINAL task (setproject, settaskid)
3500  *         no resource pool accepting default binding exists (setproject)
3501  * ELOOP - too many symbolic links in working_dir (chdir)
3502  * ENAMETOOLONG - working_dir is too long (chdir)
3503  * ENOLINK - working_dir is on an inaccessible remote machine (chdir)
3504  * ENOTDIR - working_dir is not a directory (chdir)
3505  * ESRCH - uid is not a user of project (setproject)
3506  *         project is invalid (setproject)
3507  *         the resource pool specified for project is unknown (setproject)
3508  * EBADF - the configuration for the pool is invalid (pool_set_binding)
3509  * -1 - core_set_process_path() failed (core_set_process_path)
3510  *         a resource control assignment failed (setproject)
3511  *         a system error occurred during pool_set_binding (pool_set_binding)
3512  */
3513 int
3514 restarter_set_method_context(struct method_context *cip, const char **fp)
3515 {
3516     pid_t mypid = -1;
3517     int r, ret;
3518     secflagdelta_t delta = {0};
3519 #endif /* ! codereview */

3521     cip->pwbuf = NULL;
3522     *fp = NULL;

3524     if (cip->gid != (gid_t)-1) {
3525         if (setregid(cip->gid,
3526             cip->egid != (gid_t)-1 ? cip->egid : cip->gid) != 0) {
3527             *fp = "setregid";

3529             ret = errno;
3530             assert(ret == EINVAL || ret == EPERM);
3531             goto out;
3532         }
3533     } else {
3534         if (cip->pwbuf == NULL) {
3535             switch (ret = lookup_pwd(cip)) {
3536                 case 0:
3537                     break;

3539                     case ENOMEM:
3540                     case ENOENT:
3541                         *fp = NULL;
3542                         goto out;

3544                     case EIO:
3545                     case EMFILE:
3546                     case ENFILE:
3547                         *fp = "getpwuid_r";
3548                         goto out;

3550             default:
3551                 bad_fail("lookup_pwd", ret);
3552             }
3553     }

```

```

3555         if (setregid(cip->pwd.pw_gid,
3556             cip->egid != (gid_t)-1 ?
3557             cip->egid : cip->pwd.pw_gid) != 0) {
3558             *fp = "setregid";
3559
3560             ret = errno;
3561             assert(ret == EINVAL || ret == EPERM);
3562             goto out;
3563         }
3564     }
3565
3566     if (cip->ngroups == -1) {
3567         if (cip->pwbuf == NULL) {
3568             switch (ret = lookup_pwd(cip)) {
3569                 case 0:
3570                     break;
3571
3572                 case ENOMEM:
3573                 case ENOENT:
3574                     *fp = NULL;
3575                     goto out;
3576
3577                 case EIO:
3578                 case EMFILE:
3579                 case ENFILE:
3580                     *fp = "getpwuid_r";
3581                     goto out;
3582
3583                 default:
3584                     bad_fail("lookup_pwd", ret);
3585             }
3586         }
3587
3588         /* Ok if cip->gid == -1 */
3589         if (initgroups(cip->pwd.pw_name, cip->gid) != 0) {
3590             *fp = "initgroups";
3591             ret = errno;
3592             assert(ret == EPERM);
3593             goto out;
3594         }
3595     } else if (cip->ngroups > 0 &&
3596                setgroups(cip->ngroups, cip->groups) != 0) {
3597         *fp = "setgroups";
3598
3599         ret = errno;
3600         assert(ret == EINVAL || ret == EPERM);
3601         goto out;
3602     }
3603
3604     if (cip->corefile_pattern != NULL) {
3605         mypid = getpid();
3606
3607         if (core_set_process_path(cip->corefile_pattern,
3608             strlen(cip->corefile_pattern) + 1, mypid) != 0) {
3609             *fp = "core_set_process_path";
3610             ret = -1;
3611             goto out;
3612         }
3613     }
3614
3615     delta.psd_ass_active = B_TRUE;
3616     secflags_copy(&delta.psd_assign, &cip->def_secflags.psf_inherit);
3617     if (psecflags(P_PID, P_MYID, PSF_INHERIT,
3618         &delta) != 0) {
3619         *fp = "psecflags (inherit defaults)";

```

```

3621         ret = errno;
3622         goto out;
3623     }
3624
3625     if (psecflags(P_PID, P_MYID, PSF_INHERIT,
3626         &cip->secflag_delta) != 0) {
3627         *fp = "psecflags (inherit)";
3628         ret = errno;
3629         goto out;
3630     }
3631
3632     secflags_copy(&delta.psd_assign, &cip->def_secflags.psf_lower);
3633     if (psecflags(P_PID, P_MYID, PSF_LOWER,
3634         &delta) != 0) {
3635         *fp = "psecflags (lower)";
3636         ret = errno;
3637         goto out;
3638     }
3639
3640     secflags_copy(&delta.psd_assign, &cip->def_secflags.psf_upper);
3641     if (psecflags(P_PID, P_MYID, PSF_UPPER,
3642         &delta) != 0) {
3643         *fp = "psecflags (upper)";
3644         ret = errno;
3645         goto out;
3646     }
3647
3648 #endif /* ! codereview */
3649     if (restarter_rm_libs_loadable()) {
3650         if (cip->project == NULL) {
3651             if (settaskid(getprojid(), TASK_NORMAL) == -1) {
3652                 switch (errno) {
3653                     case EACCES:
3654                     case EPERM:
3655                         *fp = "settaskid";
3656                         ret = errno;
3657                         goto out;
3658
3659                     case EINVAL:
3660                     default:
3661                         bad_fail("settaskid", errno);
3662                 }
3663             }
3664         } else {
3665             switch (ret = lookup_pwd(cip)) {
3666                 case 0:
3667                     break;
3668
3669                 case ENOMEM:
3670                 case ENOENT:
3671                     *fp = NULL;
3672                     goto out;
3673
3674                 case EIO:
3675                 case EMFILE:
3676                 case ENFILE:
3677                     *fp = "getpwuid_r";
3678                     goto out;
3679
3680                 default:
3681                     bad_fail("lookup_pwd", ret);
3682             }
3683
3684             *fp = "setproject";
3685
3686             switch (setproject(cip->project, cip->pwd.pw_name,

```

```

3687         TASK_NORMAL)) {
3688     case 0:
3689         break;
3691     case SETPROJ_ERR_TASK:
3692     case SETPROJ_ERR_POOL:
3693         ret = errno;
3694         goto out;
3696     default:
3697         ret = -1;
3698         goto out;
3699     }
3700 }
3702 if (cip->resource_pool != NULL) {
3703     if (mypid == -1)
3704         mypid = getpid();
3706     *fp = "pool_set_binding";
3708     if (pool_set_binding(cip->resource_pool, P_PID,
3709         mypid) != PO_SUCCESS) {
3710         switch (pool_error()) {
3711             case POE_INVALID_SEARCH:
3712                 ret = ENOENT;
3713                 break;
3715             case POE_BADPARAM:
3716                 ret = EINVAL;
3717                 break;
3719             case POE_INVALID_CONF:
3720                 ret = EBADF;
3721                 break;
3723             case POE_SYSTEM:
3724                 ret = -1;
3725                 break;
3727             default:
3728                 bad_fail("pool_set_binding",
3729                     pool_error());
3730         }
3732         goto out;
3733     }
3734 }
3735 }
3737 /*
3738  * Now, we have to assume our ID. If the UID is 0, we want it to be
3739  * privilege-aware, otherwise the limit set gets used instead of E/P.
3740  * We can do this by setting P as well, which keeps
3741  * PA status (see priv_can_clear_PA()).
3742  */
3744 *fp = "setppriv";
3746 if (cip->lpriv_set != NULL) {
3747     if (setppriv(PRIV_SET, PRIV_LIMIT, cip->lpriv_set) != 0) {
3748         ret = errno;
3749         assert(ret == EFAULT || ret == EPERM);
3750         goto out;
3751     }
3752 }

```

```

3753 if (cip->priv_set != NULL) {
3754     if (setppriv(PRIV_SET, PRIV_INHERITABLE, cip->priv_set) != 0) {
3755         ret = errno;
3756         assert(ret == EFAULT || ret == EPERM);
3757         goto out;
3758     }
3759 }
3761 /*
3762  * If the limit privset is already set, then must be privilege
3763  * aware. Otherwise, don't assume anything, and force privilege
3764  * aware status.
3765  */
3767 if (cip->lpriv_set == NULL && cip->priv_set != NULL) {
3768     ret = setpflags(PRIV_AWARE, 1);
3769     assert(ret == 0);
3770 }
3772 *fp = "setreuid";
3773 if (setreuid(cip->uid,
3774     cip->euid != (uid_t)-1 ? cip->euid : cip->uid) != 0) {
3775     ret = errno;
3776     assert(ret == EINVAL || ret == EPERM);
3777     goto out;
3778 }
3780 *fp = "setppriv";
3781 if (cip->priv_set != NULL) {
3782     if (setppriv(PRIV_SET, PRIV_PERMITTED, cip->priv_set) != 0) {
3783         ret = errno;
3784         assert(ret == EFAULT || ret == EPERM);
3785         goto out;
3786     }
3787 }
3789 /*
3790  * The last thing to do is chdir to the specified working directory.
3791  * This should come after the uid switching as only the user might
3792  * have access to the specified directory.
3793  */
3794 if (cip->working_dir != NULL) {
3795     do {
3796         r = chdir(cip->working_dir);
3797     } while (r != 0 && errno == EINTR);
3798     if (r != 0) {
3799         *fp = "chdir";
3800         ret = errno;
3801         goto out;
3802     }
3803 }
3805 ret = 0;
3806 out:
3807 free(cip->pwbuf);
3808 cip->pwbuf = NULL;
3809 return (ret);
3810 }
3812 void
3813 restarter_free_method_context(struct method_context *mcp)
3814 {
3815     size_t i;
3817     if (mcp->lpriv_set != NULL)
3818         priv_freerset(mcp->lpriv_set);

```

```

3819     if (mcp->priv_set != NULL)
3820         priv_freeset(mcp->priv_set);

3822     if (mcp->env != NULL) {
3823         for (i = 0; i < mcp->env_sz; i++)
3824             free(mcp->env[i]);
3825         free(mcp->env);
3826     }

3828     free(mcp->working_dir);
3829     free(mcp->corefile_pattern);
3830     free(mcp->project);
3831     free(mcp->resource_pool);
3832     free(mcp);
3833 }

3835 /*
3836  * Method keyword functions
3837  */

3839 int
3840 restarter_is_null_method(const char *meth)
3841 {
3842     return (strcmp(meth, MKW_TRUE) == 0);
3843 }

3845 static int
3846 is_kill_method(const char *method, const char *kill_str,
3847               size_t kill_str_len)
3848 {
3849     const char *cp;
3850     int sig;

3852     if (strncmp(method, kill_str, kill_str_len) != 0 ||
3853         (method[kill_str_len] != '\0' &&
3854          !isspace(method[kill_str_len])))
3855         return (-1);

3857     cp = method + kill_str_len;
3858     while (*cp != '\0' && isspace(*cp))
3859         ++cp;

3861     if (*cp == '\0')
3862         return (SIGTERM);

3864     if (*cp != '-')
3865         return (-1);

3867     return (str2sig(cp + 1, &sig) == 0 ? sig : -1);
3868 }

3870 int
3871 restarter_is_kill_proc_method(const char *method)
3872 {
3873     return (is_kill_method(method, MKW_KILL_PROC,
3874                           sizeof (MKW_KILL_PROC) - 1));
3875 }

3877 int
3878 restarter_is_kill_method(const char *method)
3879 {
3880     return (is_kill_method(method, MKW_KILL, sizeof (MKW_KILL) - 1));
3881 }

3883 /*
3884  * Stubs for now.

```

```

3885  */

3887 /* ARGSUSED */
3888 int
3889 restarter_event_get_enabled(restarter_event_t *e)
3890 {
3891     return (-1);
3892 }

3894 /* ARGSUSED */
3895 uint64_t
3896 restarter_event_get_seq(restarter_event_t *e)
3897 {
3898     return (-1);
3899 }

3901 /* ARGSUSED */
3902 void
3903 restarter_event_get_time(restarter_event_t *e, hrttime_t *time)
3904 {
3905 }

3907 /*
3908  * Check for and validate fmri specified in restarter_actions/auxiliary_fmri
3909  * 0 - Success
3910  * 1 - Failure
3911  */
3912 int
3913 restarter_inst_validate_ractions_aux_fmri(scf_instance_t *inst)
3914 {
3915     scf_handle_t *h;
3916     scf_propertygroup_t *pg;
3917     scf_property_t *prop;
3918     scf_value_t *val;
3919     char *aux_fmri;
3920     size_t size = scf_limit(SCF_LIMIT_MAX_VALUE_LENGTH);
3921     int ret = 1;

3923     if ((aux_fmri = malloc(size)) == NULL)
3924         return (1);

3926     h = scf_instance_handle(inst);

3928     pg = scf_pg_create(h);
3929     prop = scf_property_create(h);
3930     val = scf_value_create(h);
3931     if (pg == NULL || prop == NULL || val == NULL)
3932         goto out;

3934     if (instance_get_or_add_pg(inst, SCF_PG_RESTARTER_ACTIONS,
3935                               SCF_PG_RESTARTER_ACTIONS_TYPE, SCF_PG_RESTARTER_ACTIONS_FLAGS,
3936                               pg) != SCF_SUCCESS)
3937         goto out;

3939     if (get_astring_val(pg, SCF_PROPERTY_AUX_FMRI, aux_fmri, size,
3940                       prop, val) != SCF_SUCCESS)
3941         goto out;

3943     if (scf_parse_fmri(aux_fmri, NULL, NULL, NULL, NULL, NULL,
3944                       NULL) != SCF_SUCCESS)
3945         goto out;

3947     ret = 0;

3949 out:
3950     free(aux_fmri);

```

```

3951     scf_value_destroy(val);
3952     scf_property_destroy(prop);
3953     scf_pg_destroy(pg);
3954     return (ret);
3955 }

3957 /*
3958  * Get instance's boolean value in restarter_actions/auxiliary_tty
3959  * Return -1 on failure
3960  */
3961 int
3962 restarter_inst_ractions_from_tty(scf_instance_t *inst)
3963 {
3964     scf_handle_t *h;
3965     scf_propertygroup_t *pg;
3966     scf_property_t *prop;
3967     scf_value_t *val;
3968     uint8_t has_tty;
3969     int ret = -1;

3971     h = scf_instance_handle(inst);
3972     pg = scf_pg_create(h);
3973     prop = scf_property_create(h);
3974     val = scf_value_create(h);
3975     if (pg == NULL || prop == NULL || val == NULL)
3976         goto out;

3978     if (instance_get_or_add_pg(inst, SCF_PG_RESTARTER_ACTIONS,
3979         SCF_PG_RESTARTER_ACTIONS_TYPE, SCF_PG_RESTARTER_ACTIONS_FLAGS,
3980         pg) != SCF_SUCCESS)
3981         goto out;

3983     if (get_boolean_val(pg, SCF_PROPERTY_AUX_TTY, &has_tty, prop,
3984         val) != SCF_SUCCESS)
3985         goto out;

3987     ret = has_tty;

3989 out:
3990     scf_value_destroy(val);
3991     scf_property_destroy(prop);
3992     scf_pg_destroy(pg);
3993     return (ret);
3994 }

3996 /*
3997  * If the instance's dump-on-restart property exists, remove it and return true,
3998  * otherwise return false.
3999  */
4000 int
4001 restarter_inst_dump(scf_instance_t *inst)
4002 {
4003     scf_handle_t *h;
4004     scf_propertygroup_t *pg;
4005     scf_property_t *prop;
4006     scf_value_t *val;
4007     int ret = 0;

4009     h = scf_instance_handle(inst);
4010     pg = scf_pg_create(h);
4011     prop = scf_property_create(h);
4012     val = scf_value_create(h);
4013     if (pg == NULL || prop == NULL || val == NULL)
4014         goto out;

4016     if (scf_instance_get_pg(inst, SCF_PG_RESTARTER_ACTIONS, pg) !=

```

```

4017     SCF_SUCCESS) {
4018         if (scf_error() == SCF_ERROR_CONNECTION_BROKEN)
4019             uu_die(rcbroken);
4020         goto out;
4021     }

4023     if (scf_pg_get_property(pg, SCF_PROPERTY_DODUMP, prop) != SCF_SUCCESS) {
4024         if (scf_error() == SCF_ERROR_CONNECTION_BROKEN)
4025             uu_die(rcbroken);
4026         goto out;
4027     }

4029     ret = 1;

4031     if (scf_instance_delete_prop(inst, SCF_PG_RESTARTER_ACTIONS,
4032         SCF_PROPERTY_DODUMP) != SCF_SUCCESS) {
4033         if (scf_error() == SCF_ERROR_CONNECTION_BROKEN)
4034             uu_die(rcbroken);
4035         goto out;
4036     }

4038 out:
4039     scf_value_destroy(val);
4040     scf_property_destroy(prop);
4041     scf_pg_destroy(pg);
4042     return (ret);
4043 }

4045 static int
4046 restarter_inst_set_astring_prop(scf_instance_t *inst, const char *pgname,
4047     const char *pgtype, uint32_t pgflags, const char *pname, const char *str)
4048 {
4049     scf_handle_t *h;
4050     scf_propertygroup_t *pg;
4051     scf_transaction_t *t;
4052     scf_transaction_entry_t *e;
4053     scf_value_t *v;
4054     int ret = 1, r;

4056     h = scf_instance_handle(inst);

4058     pg = scf_pg_create(h);
4059     t = scf_transaction_create(h);
4060     e = scf_entry_create(h);
4061     v = scf_value_create(h);
4062     if (pg == NULL || t == NULL || e == NULL || v == NULL)
4063         goto out;

4065     if (instance_get_or_add_pg(inst, pgname, pgtype, pgflags, pg))
4066         goto out;

4068     if (scf_value_set_astring(v, str) != SCF_SUCCESS)
4069         goto out;

4071     for (;;) {
4072         if (scf_transaction_start(t, pg) != 0)
4073             goto out;

4075         if (tx_set_value(t, e, pname, SCF_TYPE_ASTRING, v) != 0)
4076             goto out;

4078         if ((r = scf_transaction_commit(t)) == 1)
4079             break;

4081         if (r == -1)
4082             goto out;

```

```

4084         scf_transaction_reset(t);
4085         if (scf_pg_update(pg) == -1)
4086             goto out;
4087     }
4088     ret = 0;

4090 out:
4091     scf_transaction_destroy(t);
4092     scf_entry_destroy(e);
4093     scf_value_destroy(v);
4094     scf_pg_destroy(pg);

4096     return (ret);
4097 }

4099 int
4100 restarter_inst_set_aux_fmri(scf_instance_t *inst)
4101 {
4102     scf_handle_t *h;
4103     scf_propertygroup_t *pg;
4104     scf_property_t *prop;
4105     scf_value_t *val;
4106     char *aux_fmri;
4107     size_t size = scf_limit(SCF_LIMIT_MAX_VALUE_LENGTH);
4108     int ret = 1;

4110     if ((aux_fmri = malloc(size)) == NULL)
4111         return (1);

4113     h = scf_instance_handle(inst);

4115     pg = scf_pg_create(h);
4116     prop = scf_property_create(h);
4117     val = scf_value_create(h);
4118     if (pg == NULL || prop == NULL || val == NULL)
4119         goto out;

4121     /*
4122      * Get auxiliary_fmri value from restarter_actions pg
4123      */
4124     if (instance_get_or_add_pg(inst, SCF_PG_RESTARTER_ACTIONS,
4125                               SCF_PG_RESTARTER_ACTIONS_TYPE, SCF_PG_RESTARTER_ACTIONS_FLAGS,
4126                               pg) != SCF_SUCCESS)
4127         goto out;

4129     if (get_astring_val(pg, SCF_PROPERTY_AUX_FMRI, aux_fmri, size,
4130                       prop, val) != SCF_SUCCESS)
4131         goto out;

4133     /*
4134      * Populate restarter/auxiliary_fmri with the obtained fmri.
4135      */
4136     ret = restarter_inst_set_astring_prop(inst, SCF_PG_RESTARTER,
4137                                         SCF_PG_RESTARTER_TYPE, SCF_PG_RESTARTER_FLAGS,
4138                                         SCF_PROPERTY_AUX_FMRI, aux_fmri);

4140 out:
4141     free(aux_fmri);
4142     scf_value_destroy(val);
4143     scf_property_destroy(prop);
4144     scf_pg_destroy(pg);
4145     return (ret);
4146 }

4148 int

```

```

4149 restarter_inst_reset_aux_fmri(scf_instance_t *inst)
4150 {
4151     return (scf_instance_delete_prop(inst,
4152                                     SCF_PG_RESTARTER, SCF_PROPERTY_AUX_FMRI));
4153 }

4155 int
4156 restarter_inst_reset_ractions_aux_fmri(scf_instance_t *inst)
4157 {
4158     return (scf_instance_delete_prop(inst,
4159                                     SCF_PG_RESTARTER_ACTIONS, SCF_PROPERTY_AUX_FMRI));
4160 }

```

```

*****
11591 Thu Jun 30 22:02:27 2016
new/usr/src/lib/librestart/common/librestart.h
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
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 (c) 2004, 2010, Oracle and/or its affiliates. All rights reserved.
23 */

25 #ifndef _LIBRESTART_H
26 #define _LIBRESTART_H

28 #include <libsysevent.h>
29 #include <libcontract.h>
30 #include <libscf.h>
31 #include <limits.h>
32 #include <priv.h>
33 #include <pwd.h>
34 #include <sys/types.h>
35 #include <sys/secflags.h>
36 #endif /* ! codereview */

38 #ifdef __cplusplus
39 extern "C" {
40 #endif

42 /*
43  * There are 3 parts to librestart.
44  * 1) The event protocol from the master restarter to its delegates.
45  * 2) A functional interface for updating the repository.
46  * 3) Convenience functions for common restarter tasks.
47  *
48  * Event protocol
49  * We need a reliable event protocol, as there's no way to define
50  * restarter events as idempotent.
51  *
52  * Currently using sysevent channels as the reliable event implementation.
53  * This could change if the implementation proves unsuitable, but
54  * the API defined here should abstract anything but a change in
55  * the fundamental event model.
56  *
57  * We offer functions to tease apart the event rather than generic
58  * nvpair interfaces. This is because each event type has a well-

```

```

59  * defined set of fields.
60 */

62 /*
63  * Some of the functions have external contracted consumers, review contracts
64  * when making incompatible changes.
65 */

67 typedef struct restarter_event_handle restarter_event_handle_t;
68 typedef struct restarter_event restarter_event_t;

70 typedef uint32_t restarter_event_type_t;

72 /*
73  * Define an event protocol version. In theory, we could use this in
74  * the future to support delegated restarters which use an older
75  * protocol. In practice, increment RESTARTER_EVENT_VERSION whenever the
76  * protocol might have changed.
77 */
78 #define RESTARTER_EVENT_VERSION 5

80 #define RESTARTER_FLAG_DEBUG 1

82 #define RESTARTER_ERRMSGSZ 1024

84 /*
85  * Event types
86  * RESTARTER_EVENT_TYPE_ADD_INSTANCE
87  * responsible for a new (stopped) instance
88  * RESTARTER_EVENT_TYPE_REMOVE_INSTANCE
89  * no longer responsible for this instance; stop it and return
90  * RESTARTER_EVENT_TYPE_ENABLE
91  * no guarantee that dependencies are met; see
92  * RESTARTER_EVENT_TYPE_START
93  * RESTARTER_EVENT_TYPE_DISABLE
94  * no guarantee that instance was running
95  * RESTARTER_EVENT_TYPE_ADMIN_DEGRADED
96  * RESTARTER_EVENT_TYPE_ADMIN_REFRESH
97  * RESTARTER_EVENT_TYPE_ADMIN_RESTART
98  * RESTARTER_EVENT_TYPE_ADMIN_MAINT_OFF
99  * RESTARTER_EVENT_TYPE_ADMIN_MAINT_ON
100 * RESTARTER_EVENT_TYPE_ADMIN_MAINT_ON_IMMEDIATE
101 * RESTARTER_EVENT_TYPE_ADMIN_MAINT_OFF
102 * RESTARTER_EVENT_TYPE_STOP
103 * dependencies are, or are becoming, unsatisfied
104 * RESTARTER_EVENT_TYPE_START
105 * dependencies have become satisfied
106 * RESTARTER_EVENT_TYPE_DEPENDENCY_CYCLE
107 * instance caused a dependency cycle
108 * RESTARTER_EVENT_TYPE_INVALID_DEPENDENCY
109 * instance has an invalid dependency
110 */

112 #define RESTARTER_EVENT_TYPE_INVALID 0
113 #define RESTARTER_EVENT_TYPE_ADD_INSTANCE 1
114 #define RESTARTER_EVENT_TYPE_REMOVE_INSTANCE 2
115 #define RESTARTER_EVENT_TYPE_ENABLE 3
116 #define RESTARTER_EVENT_TYPE_DISABLE 4
117 #define RESTARTER_EVENT_TYPE_ADMIN_DEGRADED 5
118 #define RESTARTER_EVENT_TYPE_ADMIN_REFRESH 6
119 #define RESTARTER_EVENT_TYPE_ADMIN_RESTART 7
120 #define RESTARTER_EVENT_TYPE_ADMIN_MAINT_OFF 8
121 #define RESTARTER_EVENT_TYPE_ADMIN_MAINT_ON 9
122 #define RESTARTER_EVENT_TYPE_ADMIN_MAINT_ON_IMMEDIATE 10
123 #define RESTARTER_EVENT_TYPE_STOP 11
124 #define RESTARTER_EVENT_TYPE_START 12

```



```

125 #define RESTARTER_EVENT_TYPE_DEPENDENCY_CYCLE      13
126 #define RESTARTER_EVENT_TYPE_INVALID_DEPENDENCY   14
127 #define RESTARTER_EVENT_TYPE_ADMIN_DISABLE        15
128 #define RESTARTER_EVENT_TYPE_STOP_RESET           16

130 #define RESTARTER_EVENT_ERROR                      -1

132 #define RESTARTER_EVENT_INSTANCE_DISABLED          0
133 #define RESTARTER_EVENT_INSTANCE_ENABLED          1

135 typedef enum {
136     RESTARTER_STATE_NONE,
137     RESTARTER_STATE_UNINIT,
138     RESTARTER_STATE_MAINT,
139     RESTARTER_STATE_OFFLINE,
140     RESTARTER_STATE_DISABLED,
141     RESTARTER_STATE_ONLINE,
142     RESTARTER_STATE_DEGRADED
143 } restarter_instance_state_t;

145 /*
146  * These values are ordered by severity of required restart, as we use
147  * integer comparisons to determine error flow.
148  */
149 typedef enum {
150     RERR_UNSUPPORTED = -1,
151     RERR_NONE = 0,           /* no error, restart, refresh */
152     RERR_FAULT,             /* fault occurred */
153     RERR_RESTART,           /* transition due to restart */
154     RERR_REFRESH            /* transition due to refresh */
155 } restarter_error_t;

156 /*
157  * restarter_store_contract() and restarter_remove_contract() types
158  */
159 typedef enum {
160     RESTARTER_CONTRACT_PRIMARY,
161     RESTARTER_CONTRACT_TRANSIENT
162 } restarter_contract_type_t;

164 /*
165  * restarter_bind_handle() registers a delegate with svc.startd to
166  * begin consuming events.
167  *
168  * On initial bind, the delgated restarter receives an event for each
169  * instance it is responsible for, as if that instance was new.
170  *
171  * callers must have superuser privileges
172  *
173  * The event handler can return 0 for success, or EAGAIN to request
174  * a retry of event delivery. EAGAIN may be returned 3 times before the
175  * event is discarded.
176  */
177 int restarter_bind_handle(uint32_t, const char *,
178     int (*event_handler)(restarter_event_t *), int,
179     restarter_event_handle_t **);

181 restarter_event_type_t restarter_event_get_type(restarter_event_t *);
182 uint64_t restarter_event_get_seq(restarter_event_t *);
183 void restarter_event_get_time(restarter_event_t *, hrtime_t *);
184 ssize_t restarter_event_get_instance(restarter_event_t *, char *, size_t);
185 restarter_event_handle_t *restarter_event_get_handle(restarter_event_t *);

187 /*
188  * The following functions work only on certain types of events.
189  * They fail with a return of -1 if they're called on an inappropriate event.
190  */

```

```

191 int restarter_event_get_enabled(restarter_event_t *);
192 int restarter_event_get_current_states(restarter_event_t *,
193     restarter_instance_state_t *, restarter_instance_state_t *);

195 /*
196  * State transition reasons
197  */

199 typedef enum {
200     restarter_str_none,
201     restarter_str_administrative_request,
202     restarter_str_bad_repo_state,
203     restarter_str_clear_request,
204     restarter_str_ct_ev_core,
205     restarter_str_ct_ev_exit,
206     restarter_str_ct_ev_hwerr,
207     restarter_str_ct_ev_signal,
208     restarter_str_dependencies_satisfied,
209     restarter_str_dependency_activity,
210     restarter_str_dependency_cycle,
211     restarter_str_disable_request,
212     restarter_str_enable_request,
213     restarter_str_fault_threshold_reached,
214     restarter_str_insert_in_graph,
215     restarter_str_invalid_dependency,
216     restarter_str_invalid_restarter,
217     restarter_str_method_failed,
218     restarter_str_per_configuration,
219     restarter_str_refresh,
220     restarter_str_restart_request,
221     restarter_str_restarting_too_quickly,
222     restarter_str_service_request,
223     restarter_str_startd_restart
224 } restarter_str_t;

226 struct restarter_state_transition_reason {
227     restarter_str_t str_key;
228     const char *str_short;
229     const char *str_long;
230 };

232 /*
233  * Functions for updating the repository.
234  */

236 /*
237  * When setting state to "maintenance", callers of restarter_set_states() can
238  * set aux_state to "service_request" to communicate that another service has
239  * requested maintenance state for the target service.
240  *
241  * Callers should use restarter_inst_validate_aux_fmri() to validate the fmri
242  * of the requested service and pass "service_request" for aux_state when
243  * calling restarter_set_states(). See inetd and startd for examples.
244  */
245 int restarter_set_states(restarter_event_handle_t *, const char *,
246     restarter_instance_state_t, restarter_instance_state_t,
247     restarter_instance_state_t, restarter_instance_state_t, restarter_error_t,
248     restarter_str_t);
249 int restarter_event_publish_retry(evchan_t *, const char *, const char *,
250     const char *, const char *, nvlist_t *, uint32_t);

252 /*
253  * functions for retrieving the state transition reason messages
254  */

256 #define RESTARTER_STRING_VERSION      1

```

```

258 uint32_t restarter_str_version(void);
259 const char *restarter_get_str_short(restarter_str_t);
260 const char *restarter_get_str_long(restarter_str_t);

262 int restarter_store_contract(scf_instance_t *, ctid_t,
263     restarter_contract_type_t);
264 int restarter_remove_contract(scf_instance_t *, ctid_t,
265     restarter_contract_type_t);

267 ssize_t restarter_state_to_string(restarter_instance_state_t, char *, size_t);
268 restarter_instance_state_t restarter_string_to_state(char *);

270 #define RESTARTER_METHOD_CONTEXT_VERSION      8
35 #define RESTARTER_METHOD_CONTEXT_VERSION    7

272 struct method_context {
273     /* Stable */
274     uid_t      uid, euid;
275     gid_t      gid, egid;
276     int        ngroups;          /* -1 means use initgroups(). */
277     gid_t      groups[NGROUPS_MAX];
278     psecflags_t def_secflags;
279     secflagdelta_t secflag_delta;
280 #endif /* ! codereview */
281     priv_set_t *lpriv_set, *priv_set;
282     char        *corefile_pattern; /* Optional. */
283     char        *project;          /* NULL for no change */
284     char        *resource_pool;   /* NULL for project default */
285     char        *working_dir;     /* NULL for :default */
286     char        **env;           /* NULL for no env */
287     size_t      env_sz;          /* size of env array */

289     /* Private */
290     char        *vbuf;
291     ssize_t     vbuf_sz;
292     struct passwd pwd;
293     char        *pwbuff;
294     ssize_t     pwbufsz;
295 };

297 /*
298  * An error structure that contains a message string, and a type
299  * that can be used to determine course of action by the reciever
300  * of the error structure.
301  *
302  * type - usually will be an errno equivalent but could contain
303  * defined error types for exampe SCF_ERROR_XXX
304  * msg - must be at the end of the structure as if the message is
305  * longer than EMSGSIZE we will reallocate the structure to
306  * handle the overflow
307  */
308 typedef struct mc_error {
309     int        destroy;          /* Flag to indicate destruction steps */
310     int        type;            /* Type of error for decision making */
311     int        size;           /* The size of the error message string */
312     char        msg[RESTARTER_ERRMSGSZ];
313 } mc_error_t;

315 int restarter_rm_libs_loadable(void);
316 /* instance, restarter name, method name, command line, structure pointer */
317 mc_error_t *restarter_get_method_context(uint_t, scf_instance_t *,
318     __scf_snapshot_t *, const char *, const char *, struct method_context **);
319 void restarter_mc_error_destroy(mc_error_t *);
320 int restarter_set_method_context(struct method_context *, const char **);
321 void restarter_free_method_context(struct method_context *);

```

```

324 int restarter_is_null_method(const char *);
325 int restarter_is_kill_method(const char *);
326 int restarter_is_kill_proc_method(const char *);

328 /* Validate the inst fmri specified in restarter_actions/auxiliary_fmri */
329 int restarter_inst_validate_reactions_aux_fmri(scf_instance_t *);

331 /* Delete instance's restarter_actions/auxiliary_fmri property */
332 int restarter_inst_reset_reactions_aux_fmri(scf_instance_t *);

334 /* Get boolean value from instance's restarter_actions/auxiliary_tty */
335 int restarter_inst_reactions_from_tty(scf_instance_t *);

337 /* Delete instance's restarter/auxiliary_fmri property */
338 int restarter_inst_reset_aux_fmri(scf_instance_t *);

340 /* Get boolean value from instance's restarter_actions/do_dump */
341 int restarter_inst_dump(scf_instance_t *);

343 /*
344  * Set instance's restarter/auxiliary_fmri, value come from
345  * restarter_actions/auxiliary_fmri
346  */
347 int restarter_inst_set_aux_fmri(scf_instance_t *);

349 #ifdef __cplusplus
350 }
351 #endif

353 #endif /* _LIBRESTART_H */

```

```

*****
10652 Thu Jun 30 22:02:28 2016
new/usr/src/lib/libscf/common/highlevel.c
Code review comments from jeffpc
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */

22 /*
23  * Copyright 2010 Sun Microsystems, Inc. All rights reserved.
24  * Use is subject to license terms.
25 */

27 /*
28  * This file contains high level functions used by multiple utilities.
29 */

31 #include "libscf_impl.h"

33 #include <assert.h>
34 #include <libutil.h>
35 #include <string.h>
36 #include <stdlib.h>
37 #include <sys/systeminfo.h>
38 #include <sys/uadmin.h>
39 #include <sys/utsname.h>
40 #include <sys/secflags.h>
41 #endif /* ! codereview */

43 #ifdef __x86
44 #include <smbios.h>

46 /*
47  * Check whether the platform is on the fastreboot_blacklist.
48  * Return 1 if the platform has been blacklisted, 0 otherwise.
49  */
50 static int
51 scf_is_fb_blacklisted(void)
52 {
53     smbios_hdl_t *shp;
54     smbios_system_t sys;
55     smbios_info_t info;

57     id_t id;

```

```

58     int err;
59     int i;

61     scf_simple_prop_t *prop = NULL;
62     ssize_t numvals;
63     char *platform_name;

65     int blacklisted = 0;

67     /*
68      * If there's no SMBIOS, assume it's blacklisted.
69      */
70     if ((shp = smbios_open(NULL, SMB_VERSION, 0, &err)) == NULL)
71         return (1);

73     /*
74      * If we can't read system info, assume it's blacklisted.
75      */
76     if ((id = smbios_info_system(shp, &sys)) == SMB_ERR ||
77         smbios_info_common(shp, id, &info) == SMB_ERR) {
78         blacklisted = 1;
79         goto fb_out;
80     }

82     /*
83      * If we can't read the "platforms" property from property group
84      * BOOT_CONFIG_PG_FBBLACKLIST, assume no platforms have
85      * been blacklisted.
86      */
87     if ((prop = scf_simple_prop_get(NULL, FMRI_BOOT_CONFIG,
88         BOOT_CONFIG_PG_FBBLACKLIST, "platforms")) == NULL)
89         goto fb_out;

91     numvals = scf_simple_prop_numvalues(prop);

93     for (i = 0; i < numvals; i++) {
94         platform_name = scf_simple_prop_next_astring(prop);
95         if (platform_name == NULL)
96             break;
97         if (strcmp(platform_name, info.smbi_product) == 0) {
98             blacklisted = 1;
99             break;
100         }
101     }

103 fb_out:
104     smbios_close(shp);
105     scf_simple_prop_free(prop);

107     return (blacklisted);
108 }

110 /*
111  * Add or get a property group given an FMRI.
112  * Return SCF_SUCCESS on success, SCF_FAILED on failure.
113  */
114 static int
115 scf_fmri_pg_get_or_add(const char *fmri, const char *pgname,
116     const char *pgtype, uint32_t pgflags, int add)
117 {
118     scf_handle_t *handle = NULL;
119     scf_instance_t *inst = NULL;
120     int rc = SCF_FAILED;
121     int error;

123     if ((handle = scf_handle_create(SCF_VERSION)) == NULL ||

```

```

124     scf_handle_bind(handle) != 0 ||
125     (inst = scf_instance_create(handle)) == NULL ||
126     scf_handle_decode_fmri(handle, fmri, NULL, NULL,
127     inst, NULL, NULL, SCF_DECODE_FMRI_EXACT) != SCF_SUCCESS)
128     goto scferror;

130     if (add) {
131         rc = scf_instance_add_pg(inst, pgname, pgtype, pgflags, NULL);
132         /*
133          * If the property group already exists, return SCF_SUCCESS.
134          */
135         if (rc != SCF_SUCCESS && scf_error() == SCF_ERROR_EXISTS)
136             rc = SCF_SUCCESS;
137     } else {
138         rc = scf_instance_get_pg(inst, pgname, NULL);
139     }

141 scferror:
142     if (rc != SCF_SUCCESS)
143         error = scf_error();

145     scf_instance_destroy(inst);
146     if (handle)
147         (void) scf_handle_unbind(handle);
148     scf_handle_destroy(handle);

150     if (rc != SCF_SUCCESS)
151         (void) scf_set_error(error);

153     return (rc);
154 }
155 #endif /* __x86 */

157 /*
158 * Get config properties from svc:/system/boot-config:default.
159 * It prints errors with uu_warn().
160 */
161 void
162 scf_get_boot_config(uint8_t *boot_config)
163 {
164     uint64_t ret = 0;

166     assert(boot_config);
167     *boot_config = 0;

169     {
170         /*
171          * Property vector for BOOT_CONFIG_PG_PARAMS property group.
172          */
173         scf_propvec_t ua_boot_config[] = {
174             { FASTREBOOT_DEFAULT, NULL, SCF_TYPE_BOOLEAN, NULL,
175               UA_FASTREBOOT_DEFAULT },
176             { FASTREBOOT_ONPANIC, NULL, SCF_TYPE_BOOLEAN, NULL,
177               UA_FASTREBOOT_ONPANIC },
178             { NULL }
179         };
180         scf_propvec_t *prop;

182         for (prop = ua_boot_config; prop->pv_prop != NULL; prop++)
183             prop->pv_ptr = &ret;
184         prop = NULL;
185         if (scf_read_propvec(FMRI_BOOT_CONFIG, BOOT_CONFIG_PG_PARAMS,
186             B_TRUE, ua_boot_config, &prop) != SCF_FAILED) {

188 #ifdef __x86
189     /*

```

```

190         * Unset both flags if the platform has been
191         * blacklisted.
192         */
193         if (scf_is_fb_blacklisted())
194             return;
195 #endif /* __x86 */
196         *boot_config = (uint8_t)ret;
197         return;
198     }
199 #if defined(FASTREBOOT_DEBUG)
200     if (prop != NULL) {
201         (void) uu_warn("Service %s property '%s/%s' "
202             "not found.\n", FMRI_BOOT_CONFIG,
203             BOOT_CONFIG_PG_PARAMS, prop->pv_prop);
204     } else {
205         (void) uu_warn("Unable to read service %s "
206             "property '%s': %s\n", FMRI_BOOT_CONFIG,
207             BOOT_CONFIG_PG_PARAMS, scf_strerror(scf_error()));
208     }
209 #endif /* FASTREBOOT_DEBUG */
210 }

213 /*
214 * Get or set properties in non-persistent "config_ovr" property group
215 * in svc:/system/boot-config:default.
216 * It prints errors with uu_warn().
217 */
218 /*ARGSUSED*/
219 static int
220 scf_getset_boot_config_ovr(int set, uint8_t *boot_config_ovr)
221 {
222     int rc = SCF_SUCCESS;

224     assert(boot_config_ovr);

226 #ifndef __x86
227     return (rc);
228 #else
229     {
230         /*
231          * Property vector for BOOT_CONFIG_PG_OVR property group.
232          */
233         scf_propvec_t ua_boot_config_ovr[] = {
234             { FASTREBOOT_DEFAULT, NULL, SCF_TYPE_BOOLEAN, NULL,
235               UA_FASTREBOOT_DEFAULT },
236             { FASTREBOOT_ONPANIC, NULL, SCF_TYPE_BOOLEAN, NULL,
237               UA_FASTREBOOT_ONPANIC },
238             { NULL }
239         };
240         scf_propvec_t *prop;

242         rc = scf_fmri_pg_get_or_add(FMRI_BOOT_CONFIG,
243             BOOT_CONFIG_PG_OVR, SCF_GROUP_APPLICATION,
244             SCF_PG_FLAG_NONPERSISTENT, set);

246         if (rc != SCF_SUCCESS) {
247 #if defined(FASTREBOOT_DEBUG)
248             if (set)
249                 (void) uu_warn("Unable to add service %s "
250                     "property group '%s'\n",
251                     FMRI_BOOT_CONFIG, BOOT_CONFIG_PG_OVR);
252 #endif /* FASTREBOOT_DEBUG */
253             return (rc);
254         }

```

```

256     for (prop = ua_boot_config_ovr; prop->pv_prop != NULL; prop++)
257         prop->pv_ptr = boot_config_ovr;
258     prop = NULL;

260     if (set)
261         rc = scf_write_propvec(FMRI_BOOT_CONFIG,
262                               BOOT_CONFIG_PG_OVR, ua_boot_config_ovr, &prop);
263     else
264         rc = scf_read_propvec(FMRI_BOOT_CONFIG,
265                              BOOT_CONFIG_PG_OVR, B_FALSE, ua_boot_config_ovr,
266                              &prop);

268 #if defined(FASTREBOOT_DEBUG)
269     if (rc != SCF_SUCCESS) {
270         if (prop != NULL) {
271             (void) uu_warn("Service %s property '%s/%s' "
272                          "not found.\n", FMRI_BOOT_CONFIG,
273                          BOOT_CONFIG_PG_OVR, prop->pv_prop);
274         } else {
275             (void) uu_warn("Unable to %s service %s "
276                          "property '%s': %s\n", set ? "set" : "get",
277                          FMRI_BOOT_CONFIG, BOOT_CONFIG_PG_OVR,
278                          scf_strerror(scf_error()));
279         }
280     }
281 #endif /* FASTREBOOT_DEBUG */

283     if (set)
284         (void) smf_refresh_instance(FMRI_BOOT_CONFIG);

286     return (rc);

288 }
289 #endif /* __x86 */
290 }

292 /*
293  * Get values of properties in non-persistent "config_ovr" property group.
294  */
295 void
296 scf_get_boot_config_ovr(uint8_t *boot_config_ovr)
297 {
298     (void) scf_getset_boot_config_ovr(B_FALSE, boot_config_ovr);
299 }

301 /*
302  * Set value of "config_ovr/fastreboot_default".
303  */
304 int
305 scf_fastreboot_default_set_transient(boolean_t value)
306 {
307     uint8_t boot_config_ovr = 0;

309     if (value == B_TRUE)
310         boot_config_ovr = UA_FASTREBOOT_DEFAULT | UA_FASTREBOOT_ONPANIC;

312     return (scf_getset_boot_config_ovr(B_TRUE, &boot_config_ovr));
313 }

315 /*
316  * Check whether Fast Reboot is the default operating mode.
317  * Return 0 if
318  * 1. the platform is xVM
319  * or
320  * 2. svc:/system/boot-config:default service doesn't exist,
321  * or

```

```

322  * 3. property "config/fastreboot_default" doesn't exist,
323  * or
324  * 4. value of property "config/fastreboot_default" is set to "false"
325  *    and "config_ovr/fastreboot_default" is not set to "true",
326  * or
327  * 5. the platform has been blacklisted.
328  * or
329  * 6. value of property "config_ovr/fastreboot_default" is set to "false".
330  * Return non-zero otherwise.
331  */
332 int
333 scf_is_fastboot_default(void)
334 {
335     uint8_t boot_config = 0, boot_config_ovr;
336     char procbuf[SYS_NMLN];

338     /*
339      * If we are on xVM, do not fast reboot by default.
340      */
341     if (sysinfo(SI_PLATFORM, procbuf, sizeof (procbuf)) == -1 ||
342         strcmp(procbuf, "i86xp") == 0)
343         return (0);

345     /*
346      * Get property values from "config" property group
347      */
348     scf_get_boot_config(&boot_config);

350     /*
351      * Get property values from non-persistent "config_ovr" property group
352      */
353     boot_config_ovr = boot_config;
354     scf_get_boot_config_ovr(&boot_config_ovr);

356     return (boot_config & boot_config_ovr & UA_FASTREBOOT_DEFAULT);
357 }

359 /*
360  * Read the default security-flags from system/process-security and return a
361  * secflagset_t suitable for psecflags(2)
362  *
363  * Unfortunately, this symbol must _exist_ in the native build, for the sake
364  * of the mapfile, even though we don't ever use it, and it will never work.
365  */
366 struct group_desc {
367     secflagset_t *set;
368     char *fmri;
369 };

371 int
372 scf_default_secflags(scf_handle_t *hdl, psecflags_t *flags)
373 {
374 #if !defined(NATIVE_BUILD)
375     scf_property_t *prop;
376     scf_value_t *val;
377     const char *flagname;
378     int flag;
379     struct group_desc *g;
380     struct group_desc groups[] = {
381         {NULL, "svc:/system/process-security/"
382          "properties/default"},
383         {NULL, "svc:/system/process-security/"
384          "properties/lower"},
385         {NULL, "svc:/system/process-security/"
386          "properties/upper"},
387         {NULL, NULL}

```

```
388     };
389
390     groups[0].set = &flags->psf_inherit;
391     groups[1].set = &flags->psf_lower;
392     groups[2].set = &flags->psf_upper;
393
394     /* Ensure sane defaults */
395     psecflags_default(flags);
396
397     for (g = groups; g->set != NULL; g++) {
398         for (flag = 0; (flagname = secflag_to_str(flag)) != NULL;
399             flag++) {
400             char *pfmri;
401             uint8_t flagval = 0;
402
403             if ((val = scf_value_create(hndl)) == NULL)
404                 return (-1);
405
406             if ((prop = scf_property_create(hndl)) == NULL) {
407                 scf_value_destroy(val);
408                 return (-1);
409             }
410
411             if ((pfmri = uu_msprintf("%s/%s", g->fmri,
412                                     flagname)) == NULL)
413                 uu_die("Allocation failure\n");
414
415             if (scf_handle_decode_fmri(hndl, pfmri,
416                                     NULL, NULL, NULL, NULL, prop, NULL) != 0)
417                 goto next;
418
419             if (scf_property_get_value(prop, val) != 0)
420                 goto next;
421
422             (void) scf_value_get_boolean(val, &flagval);
423
424             if (flagval != 0)
425                 secflag_set(g->set, flag);
426             else
427                 secflag_clear(g->set, flag);
428
429 next:
430             uu_free(pfmri);
431             scf_value_destroy(val);
432             scf_property_destroy(prop);
433         }
434     }
435
436     if (!psecflags_validate(flags))
437         return (-1);
438
439     return (0);
440 #else
441     assert(0);
442     abort();
443 #endif /* !NATIVE_BUILD */
444 }
445 #endif /* !codereview */
```

```

*****
      8283 Thu Jun 30 22:02:28 2016
new/usr/src/lib/libscf/common/mapfile-vers
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
_____unchanged_portion_omitted_____

```

```

280 SYMBOL_VERSION SUNWprivate_1.1 {
281     global:
282         gen_filenms_from_fmri;
283         ismember;
284         scf_canonify_fmri;
285         scf_cmp_pattern;
286         _scf_create_errors;
287         scf_decode32;
288         scf_encode32;
289         scf_general_pg_setup;
290         _scf_get_fma_notify_params;
291         _scf_get_svc_notify_params;
292         _scf_handle_decorations;
293         scf_is_compatible_type;
294         _scf_notify_add_pgname;
295         _scf_notify_add_pgtype;
296         _scf_notify_get_params;
297         _scf_notify_wait;
298         scf_parse_file_fmri;
299         scf_parse_fmri;
300         scf_parse_svc_fmri;
301         _scf_pg_is_read_protected;
302         _scf_pg_wait;
303         scf_read_count_property;
304         _scf_read_single_astring_from_pg;
305         _scf_read_tmpl_prop_type_as_string;
306         _scf_repository_switch;
307         _scf_request_backup;
308         _scf_sanitize_locale;
309         _scf_set_annotation;
310         scf_set_count_property;
311         scf_simple_handle_destory;
312         _scf_snapshot_attach;
313         _scf_snapshot_delete;
314         _scf_snapshot_take_attach;
315         _scf_snapshot_take_new;
316         _scf_snapshot_take_new_named;
317         _scf_tmpl_add_error;
318         _scf_tmpl_error_set_prefix;
319         scf_transaction_setup;
320         scf_transaction_restart;
321         scf_walk_fmri;
322         _smf_refresh_instance_i;
323         scf_read_propvec;
324         scf_write_propvec;
325         scf_clean_propvec;
326         scf_instance_delete_prop;
327         scf_get_boot_config;
328         scf_get_boot_config_ovr;
329         scf_is_fastboot_default;
330         scf_fastreboot_default_set_transient;
331         scf_default_secflags;
332 #endif /* !codereview */
333         _check_services;
334         _scf_handle_create_and_bind;
335         _smf_refresh_all_instances;

```

```

336     local:
337         *;
338 };

```

new/usr/src/lib/libscf/inc/libscf.h

1

```
*****
34271 Thu Jun 30 22:02:29 2016
new/usr/src/lib/libscf/inc/libscf.h
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
    unchanged_portion_omitted_

197 typedef struct scf_tmpl_error scf_tmpl_error_t;

199 /*
200  * scf_tmpl_strerror() human readable flag
201  */
202 #define SCF_TMPL_STRERROR_HUMAN 0x1

204 /*
205  * Standard services
206  */
207 #define SCF_SERVICE_CONFIG ((const char *) \
208     "svc:/system/svc/repository:default")
209 #define SCF_INSTANCE_GLOBAL ((const char *) \
210     "svc:/system/svc/global:default")
211 #define SCF_SERVICE_GLOBAL ((const char *) \
212     "svc:/system/svc/global")
213 #define SCF_SERVICE_STARTD ((const char *) \
214     "svc:/system/svc/restarter:default")
215 #define SCF_INSTANCE_EMI ((const char *) \
216     "svc:/system/early-manifest-import:default")
217 #define SCF_INSTANCE_FS_MINIMAL ((const char *) \
218     "svc:/system/filesystem/minimal:default")
219 #define SCF_INSTANCE_MI ((const char *) \
220     "svc:/system/manifest-import:default")

222 /*
223  * Major milestones
224  */
225 #define SCF_MILESTONE_SINGLE_USER \
226     ((const char *) "svc:/milestone/single-user:default")
227 #define SCF_MILESTONE_MULTI_USER \
228     ((const char *) "svc:/milestone/multi-user:default")
229 #define SCF_MILESTONE_MULTI_USER_SERVER \
230     ((const char *) "svc:/milestone/multi-user-server:default")

232 /*
233  * standard scope names
234  */
235 #define SCF_SCOPE_LOCAL ((const char *) "localhost")

237 /*
238  * Property group types
239  */
240 #define SCF_GROUP_APPLICATION ((const char *) "application")
241 #define SCF_GROUP_FRAMEWORK ((const char *) "framework")
242 #define SCF_GROUP_DEPENDENCY ((const char *) "dependency")
243 #define SCF_GROUP_METHOD ((const char *) "method")
244 #define SCF_GROUP_TEMPLATE ((const char *) "template")
245 #define SCF_GROUP_TEMPLATE_PG_PATTERN ((const char *) "template_pg_pattern")
246 #define SCF_GROUP_TEMPLATE_PROP_PATTERN ((const char *) "template_prop_pattern")

248 /*
249  * Dependency types
250  */
251 #define SCF_DEP_REQUIRE_ALL ((const char *) "require_all")
252 #define SCF_DEP_REQUIRE_ANY ((const char *) "require_any")
```

new/usr/src/lib/libscf/inc/libscf.h

2

```
253 #define SCF_DEP_EXCLUDE_ALL ((const char *) "exclude_all")
254 #define SCF_DEP_OPTIONAL_ALL ((const char *) "optional_all")

256 #define SCF_DEP_RESET_ON_ERROR ((const char *) "error")
257 #define SCF_DEP_RESET_ON_RESTART ((const char *) "restart")
258 #define SCF_DEP_RESET_ON_REFRESH ((const char *) "refresh")
259 #define SCF_DEP_RESET_ON_NONE ((const char *) "none")

261 /*
262  * Standard property group names
263  */
264 #define SCF_PG_GENERAL ((const char *) "general")
265 #define SCF_PG_GENERAL_OVR ((const char *) "general_ovr")
266 #define SCF_PG_RESTARTER ((const char *) "restarter")
267 #define SCF_PG_RESTARTER_ACTIONS ((const char *) "restarter_actions")
268 #define SCF_PG_METHOD_CONTEXT ((const char *) "method_context")
269 #define SCF_PG_APP_DEFAULT ((const char *) "application")
270 #define SCF_PG_DEPENDENTS ((const char *) "dependents")
271 #define SCF_PG_OPTIONS ((const char *) "options")
272 #define SCF_PG_OPTIONS_OVR ((const char *) "options_ovr")
273 #define SCF_PG_STARTD ((const char *) "startd")
274 #define SCF_PG_STARTD_PRIVATE ((const char *) "svc-startd-private")
275 #define SCF_PG_DEATHROW ((const char *) "deathrow")
276 #define SCF_PG_MANIFESTFILES ((const char *) "manifestfiles")

278 /*
279  * Template property group names and prefixes
280  */
281 #define SCF_PG_TM_COMMON_NAME ((const char *) "tm_common_name")
282 #define SCF_PG_TM_DESCRIPTION ((const char *) "tm_description")

284 #define SCF_PG_TM_MAN_PREFIX ((const char *) "tm_man_")
285 #define SCF_PG_TM_DOC_PREFIX ((const char *) "tm_doc_")

287 /*
288  * Standard property names
289  */
290 #define SCF_PROPERTY_ACTIVE_POSTFIX ((const char *) "active")
291 #define SCF_PROPERTY_AUX_STATE ((const char *) "auxiliary_state")
292 #define SCF_PROPERTY_AUX_FMRI ((const char *) "auxiliary_fmri")
293 #define SCF_PROPERTY_AUX_TTY ((const char *) "auxiliary_tty")
294 #define SCF_PROPERTY_CONTRACT ((const char *) "contract")
295 #define SCF_PROPERTY_COREFILE_PATTERN ((const char *) "corefile_pattern")
296 #define SCF_PROPERTY_DEGRADED ((const char *) "degraded")
297 #define SCF_PROPERTY_DEGRADE_IMMEDIATE ((const char *) "degrade_immediate")
298 #define SCF_PROPERTY_DODUMP ((const char *) "do_dump")
299 #define SCF_PROPERTY_DURATION ((const char *) "duration")
300 #define SCF_PROPERTY_ENABLED ((const char *) "enabled")
301 #define SCF_PROPERTY_DEATHROW ((const char *) "deathrow")
302 #define SCF_PROPERTY_ENTITY_STABILITY ((const char *) "entity_stability")
303 #define SCF_PROPERTY_ENTITIES ((const char *) "entities")
304 #define SCF_PROPERTY_EXEC ((const char *) "exec")
305 #define SCF_PROPERTY_GROUP ((const char *) "group")
306 #define SCF_PROPERTY_GROUPING ((const char *) "grouping")
307 #define SCF_PROPERTY_IGNORE ((const char *) "ignore_error")
308 #define SCF_PROPERTY_INTERNAL_SEPARATORS ((const char *) "internal_separators")
309 #define SCF_PROPERTY_LIMIT_PRIVILEGES ((const char *) "limit_privileges")
310 #define SCF_PROPERTY_MAINT_OFF ((const char *) "maint_off")
311 #define SCF_PROPERTY_MAINT_ON ((const char *) "maint_on")
312 #define SCF_PROPERTY_MAINT_ON_IMMEDIATE ((const char *) "maint_on_immediate")
313 #define SCF_PROPERTY_MAINT_ON_IMMTEMP ((const char *) "maint_on_immtemp")
314 #define SCF_PROPERTY_MAINT_ON_TEMPORARY ((const char *) "maint_on_temporary")
315 #define SCF_PROPERTY_METHOD_PID ((const char *) "method_pid")
316 #define SCF_PROPERTY_MILESTONE ((const char *) "milestone")
317 #define SCF_PROPERTY_NEED_SESSION ((const char *) "need_session")
318 #define SCF_PROPERTY_NEXT_STATE ((const char *) "next_state")
```



```

319 #define SCF_PROPERTY_PACKAGE ((const char *) "package")
320 #define SCF_PROPERTY_PRIVILEGES ((const char *) "privileges")
321 #define SCF_PROPERTY_PROFILE ((const char *) "profile")
322 #define SCF_PROPERTY_PROJECT ((const char *) "project")
323 #define SCF_PROPERTY_REFRESH ((const char *) "refresh")
324 #define SCF_PROPERTY_RESOURCE_POOL ((const char *) "resource_pool")
325 #define SCF_PROPERTY_ENVIRONMENT ((const char *) "environment")
326 #define SCF_PROPERTY_RESTART ((const char *) "restart")
327 #define SCF_PROPERTY_RESTARTER ((const char *) "restarter")
328 #define SCF_PROPERTY_RESTART_INTERVAL ((const char *) "restart_interval")
329 #define SCF_PROPERTY_RESTART_ON ((const char *) "restart_on")
330 #define SCF_PROPERTY_RESTORE ((const char *) "restore")
331 #define SCF_PROPERTY_SECFLAGS ((const char *) "security_flags")
332 #endif /* ! codereview */
333 #define SCF_PROPERTY_SINGLE_INSTANCE ((const char *) "single_instance")
334 #define SCF_PROPERTY_START_METHOD_TIMESTAMP \
335 ((const char *) "start_method_timestamp")
336 #define SCF_PROPERTY_START_METHOD_WAITSTATUS \
337 ((const char *) "start_method_waitstatus")
338 #define SCF_PROPERTY_START_PID ((const char *) "start_pid")
339 #define SCF_PROPERTY_STATE ((const char *) "state")
340 #define SCF_PROPERTY_STABILITY ((const char *) "stability")
341 #define SCF_PROPERTY_STATE_TIMESTAMP ((const char *) "state_timestamp")
342 #define SCF_PROPERTY_SUPP_GROUPS ((const char *) "supp_groups")
343 #define SCF_PROPERTY_TIMEOUT ((const char *) "timeout_seconds")
344 #define SCF_PROPERTY_TIMEOUT_RETRY ((const char *) "timeout_retry")
345 #define SCF_PROPERTY_TRANSIENT_CONTRACT ((const char *) "transient_contract")
346 #define SCF_PROPERTY_TYPE ((const char *) "type")
347 #define SCF_PROPERTY_USE_PROFILE ((const char *) "use_profile")
348 #define SCF_PROPERTY_USER ((const char *) "user")
349 #define SCF_PROPERTY_UTMPX_PREFIX ((const char *) "utmpx_prefix")
350 #define SCF_PROPERTY_WORKING_DIRECTORY ((const char *) "working_directory")

352 /*
353  * Template property names
354  */
355 #define SCF_PROPERTY_TM_CARDINALITY_MIN ((const char *) "cardinality_min")
356 #define SCF_PROPERTY_TM_CARDINALITY_MAX ((const char *) "cardinality_max")
357 #define SCF_PROPERTY_TM_CHOICES_INCLUDE_VALUES ((const char *) \
358 "choices_include_values")
359 #define SCF_PROPERTY_TM_CHOICES_NAME ((const char *) "choices_name")
360 #define SCF_PROPERTY_TM_CHOICES_RANGE ((const char *) "choices_range")
361 #define SCF_PROPERTY_TM_CONSTRAINT_NAME ((const char *) "constraint_name")
362 #define SCF_PROPERTY_TM_CONSTRAINT_RANGE ((const char *) "constraint_range")
363 #define SCF_PROPERTY_TM_MANPATH ((const char *) "manpath")
364 #define SCF_PROPERTY_TM_NAME ((const char *) "name")
365 #define SCF_PROPERTY_TM_PG_PATTERN ((const char *) "pg_pattern")
366 #define SCF_PROPERTY_TM_REQUIRED ((const char *) "required")
367 #define SCF_PROPERTY_TM_SECTION ((const char *) "section")
368 #define SCF_PROPERTY_TM_TARGET ((const char *) "target")
369 #define SCF_PROPERTY_TM_TITLE ((const char *) "title")
370 #define SCF_PROPERTY_TM_TYPE ((const char *) "type")
371 #define SCF_PROPERTY_TM_URI ((const char *) "uri")
372 #define SCF_PROPERTY_TM_VALUE_PREFIX ((const char *) "value_")
373 #define SCF_PROPERTY_TM_VALUES_NAME ((const char *) "values_name")
374 #define SCF_PROPERTY_TM_VISIBILITY ((const char *) "visibility")
375 #define SCF_PROPERTY_TM_COMMON_NAME_PREFIX ((const char *) "common_name_")
376 #define SCF_PROPERTY_TM_DESCRIPTION_PREFIX ((const char *) "description_")
377 #define SCF_PROPERTY_TM_UNITS_PREFIX ((const char *) "units_")

379 /*
380  * Templates wildcard string
381  */
382 #define SCF_TMPL_WILDCARD ((const char *) "**")

384 /*

```

```

385  * Strings used by restarters for state and next_state properties.
386  * MAX_SCF_STATE_STRING holds the max length of a state string, including the
387  * terminating null.
388  */
389 #define MAX_SCF_STATE_STRING_SZ 14

392 #define SCF_STATE_STRING_NONE ((const char *) "none")
393 #define SCF_STATE_STRING_UNINIT ((const char *) "uninitialized")
394 #define SCF_STATE_STRING_MAINT ((const char *) "maintenance")
395 #define SCF_STATE_STRING_OFFLINE ((const char *) "offline")
396 #define SCF_STATE_STRING_DISABLED ((const char *) "disabled")
397 #define SCF_STATE_STRING_ONLINE ((const char *) "online")
398 #define SCF_STATE_STRING_DEGRADED ((const char *) "degraded")
399 #define SCF_STATE_STRING_LEGACY ((const char *) "legacy_run")

401 #define SCF_STATE_UNINIT 0x00000001
402 #define SCF_STATE_MAINT 0x00000002
403 #define SCF_STATE_OFFLINE 0x00000004
404 #define SCF_STATE_DISABLED 0x00000008
405 #define SCF_STATE_ONLINE 0x00000010
406 #define SCF_STATE_DEGRADED 0x00000020
407 #define SCF_STATE_ALL 0x0000003F

409 /*
410  * software fma svc-transition class
411  */
412 #define SCF_NOTIFY_PARAMS_VERSION 0X0
413 #define SCF_NOTIFY_NAME_FMRI ((const char *) "fmri")
414 #define SCF_NOTIFY_NAME_VERSION ((const char *) "version")
415 #define SCF_NOTIFY_NAME_TSET ((const char *) "tset")
416 #define SCF_NOTIFY_PG_POSTFIX ((const char *) "fmnotify")
417 #define SCF_NOTIFY_PARAMS ((const char *) "notify-params")
418 #define SCF_NOTIFY_PARAMS_INST \
419 ((const char *) "svc:/system/fm/notify-params:default")
420 #define SCF_SVC_TRANSITION_CLASS \
421 ((const char *) "ireport.os.smf.state-transition")
422 #define SCF_NOTIFY_PARAMS_PG_TYPE ((const char *) "notify_params")

424 /*
425  * Useful transition macros
426  */
427 #define SCF_TRANS_SHIFT_INITIAL_STATE(s) ((s) << 16)
428 #define SCF_TRANSITION_ALL \
429 (SCF_TRANS_SHIFT_INITIAL_STATE(SCF_STATE_ALL) | SCF_STATE_ALL)
430 #define SCF_TRANS(f, t) (SCF_TRANS_SHIFT_INITIAL_STATE(f) | (t))
431 #define SCF_TRANS_VALID(t) (!(t) & ~SCF_TRANSITION_ALL)
432 #define SCF_TRANS_INITIAL_STATE(t) ((t) >> 16 & SCF_STATE_ALL)
433 #define SCF_TRANS_FINAL_STATE(t) ((t) & SCF_STATE_ALL)

435 /*
436  * Prefixes for states in state transition notification
437  */
438 #define SCF_STN_PREFIX_FROM ((const char *) "from-")
439 #define SCF_STN_PREFIX_TO ((const char *) "to-")

441 #define SCF_PG_FLAG_NONPERSISTENT 0x1

443 #define SCF_TRACE_LIBRARY 0x1
444 #define SCF_TRACE_DAEMON 0x2

446 #define SMF_IMMEDIATE 0x1
447 #define SMF_TEMPORARY 0x2
448 #define SMF_AT_NEXT_BOOT 0x4

450 scf_error_t scf_error(void);

```

```

451 const char *scf_strerror(scf_error_t);

453 ssize_t scf_limit(uint32_t code);
454 #define SCF_LIMIT_MAX_NAME_LENGTH      -2000U
455 #define SCF_LIMIT_MAX_VALUE_LENGTH     -2001U
456 #define SCF_LIMIT_MAX_PG_TYPE_LENGTH  -2002U
457 #define SCF_LIMIT_MAX_FMRI_LENGTH      -2003U

459 scf_handle_t *scf_handle_create(scf_version_t);

461 int scf_handle_decorate(scf_handle_t *, const char *, scf_value_t *);
462 #define SCF_DECORATE_CLEAR      ((scf_value_t *)0)

464 int scf_handle_bind(scf_handle_t *);
465 int scf_handle_unbind(scf_handle_t *);
466 void scf_handle_destroy(scf_handle_t *);

468 int scf_type_base_type(scf_type_t type, scf_type_t *out);
469 const char *scf_type_to_string(scf_type_t);
470 scf_type_t scf_string_to_type(const char *);

472 /* values */
473 scf_value_t *scf_value_create(scf_handle_t *);
474 scf_handle_t *scf_value_handle(const scf_value_t *);
475 void scf_value_destroy(scf_value_t *);

477 scf_type_t scf_value_base_type(const scf_value_t *);
478 scf_type_t scf_value_type(const scf_value_t *);
479 int scf_value_is_type(const scf_value_t *, scf_type_t);

481 void scf_value_reset(scf_value_t *);

483 int scf_value_get_boolean(const scf_value_t *, uint8_t *);
484 int scf_value_get_count(const scf_value_t *, uint64_t *);
485 int scf_value_get_integer(const scf_value_t *, int64_t *);
486 int scf_value_get_time(const scf_value_t *, int64_t *, int32_t *);
487 ssize_t scf_value_get_astring(const scf_value_t *, char *, size_t);
488 ssize_t scf_value_get_ustring(const scf_value_t *, char *, size_t);
489 ssize_t scf_value_get_opaque(const scf_value_t *, void *, size_t);

491 void scf_value_set_boolean(scf_value_t *, uint8_t);
492 void scf_value_set_count(scf_value_t *, uint64_t);
493 void scf_value_set_integer(scf_value_t *, int64_t);
494 int scf_value_set_time(scf_value_t *, int64_t, int32_t);
495 int scf_value_set_astring(scf_value_t *, const char *);
496 int scf_value_set_ustring(scf_value_t *, const char *);
497 int scf_value_set_opaque(scf_value_t *, const void *, size_t);

499 ssize_t scf_value_get_as_string(const scf_value_t *, char *, size_t);
500 ssize_t scf_value_get_as_string_typed(const scf_value_t *, scf_type_t,
501 char *, size_t);
502 int scf_value_set_from_string(scf_value_t *, scf_type_t, const char *);

504 scf_iter_t *scf_iter_create(scf_handle_t *);
505 scf_handle_t *scf_iter_handle(const scf_iter_t *);
506 void scf_iter_reset(scf_iter_t *);
507 void scf_iter_destroy(scf_iter_t *);

509 int scf_iter_handle_scopes(scf_iter_t *, const scf_handle_t *);
510 int scf_iter_scope_services(scf_iter_t *, const scf_scope_t *);
511 int scf_iter_service_instances(scf_iter_t *, const scf_service_t *);
512 int scf_iter_service_pgs(scf_iter_t *, const scf_service_t *);
513 int scf_iter_instance_pgs(scf_iter_t *, const scf_instance_t *);
514 int scf_iter_instance_pgs_composed(scf_iter_t *, const scf_instance_t *,
515 const scf_snapshot_t *);
516 int scf_iter_service_pgs_typed(scf_iter_t *, const scf_service_t *,

```

```

517 const char *);
518 int scf_iter_instance_pgs_typed(scf_iter_t *, const scf_instance_t *,
519 const char *);
520 int scf_iter_instance_pgs_typed_composed(scf_iter_t *, const scf_instance_t *,
521 const scf_snapshot_t *, const char *);
522 int scf_iter_snaplevel_pgs(scf_iter_t *, const scf_snaplevel_t *);
523 int scf_iter_snaplevel_pgs_typed(scf_iter_t *, const scf_snaplevel_t *,
524 const char *);
525 int scf_iter_instance_snapshots(scf_iter_t *, const scf_instance_t *);
526 int scf_iter_pg_properties(scf_iter_t *, const scf_propertygroup_t *);
527 int scf_iter_property_values(scf_iter_t *, const scf_property_t *);

529 int scf_iter_next_scope(scf_iter_t *, scf_scope_t *);
530 int scf_iter_next_service(scf_iter_t *, scf_service_t *);
531 int scf_iter_next_instance(scf_iter_t *, scf_instance_t *);
532 int scf_iter_next_pg(scf_iter_t *, scf_propertygroup_t *);
533 int scf_iter_next_property(scf_iter_t *, scf_property_t *);
534 int scf_iter_next_snapshot(scf_iter_t *, scf_snapshot_t *);
535 int scf_iter_next_value(scf_iter_t *, scf_value_t *);

537 scf_scope_t *scf_scope_create(scf_handle_t *);
538 scf_handle_t *scf_scope_handle(const scf_scope_t *);

540 /* XXX eventually remove this */
541 #define scf_handle_get_local_scope(h, s) \
542     scf_handle_get_scope((h), SCF_SCOPE_LOCAL, (s))

544 int scf_handle_get_scope(scf_handle_t *, const char *, scf_scope_t *);
545 void scf_scope_destroy(scf_scope_t *);
546 ssize_t scf_scope_get_name(const scf_scope_t *, char *, size_t);

548 ssize_t scf_scope_to_fmri(const scf_scope_t *, char *, size_t);

550 scf_service_t *scf_service_create(scf_handle_t *);
551 scf_handle_t *scf_service_handle(const scf_service_t *);
552 void scf_service_destroy(scf_service_t *);
553 int scf_scope_get_parent(const scf_scope_t *, scf_scope_t *);
554 ssize_t scf_service_get_name(const scf_service_t *, char *, size_t);
555 ssize_t scf_service_to_fmri(const scf_service_t *, char *, size_t);
556 int scf_service_get_parent(const scf_service_t *, scf_scope_t *);
557 int scf_scope_get_service(const scf_scope_t *, const char *, scf_service_t *);
558 int scf_scope_add_service(const scf_scope_t *, const char *, scf_service_t *);
559 int scf_service_delete(scf_service_t *);

561 scf_instance_t *scf_instance_create(scf_handle_t *);
562 scf_handle_t *scf_instance_handle(const scf_instance_t *);
563 void scf_instance_destroy(scf_instance_t *);
564 ssize_t scf_instance_get_name(const scf_instance_t *, char *, size_t);
565 ssize_t scf_instance_to_fmri(const scf_instance_t *, char *, size_t);
566 int scf_service_get_instance(const scf_service_t *, const char *,
567 scf_instance_t *);
568 int scf_service_add_instance(const scf_service_t *, const char *,
569 scf_instance_t *);
570 int scf_instance_delete(scf_instance_t *);

572 scf_snapshot_t *scf_snapshot_create(scf_handle_t *);
573 scf_handle_t *scf_snapshot_handle(const scf_snapshot_t *);
574 void scf_snapshot_destroy(scf_snapshot_t *);
575 ssize_t scf_snapshot_get_name(const scf_snapshot_t *, char *, size_t);
576 int scf_snapshot_get_parent(const scf_snapshot_t *, scf_instance_t *);
577 int scf_instance_get_snapshot(const scf_instance_t *, const char *,
578 scf_snapshot_t *);
579 int scf_snapshot_update(scf_snapshot_t *);

581 scf_snaplevel_t *scf_snaplevel_create(scf_handle_t *);
582 scf_handle_t *scf_snaplevel_handle(const scf_snaplevel_t *);

```

```

583 void scf_snaplevel_destroy(scf_snaplevel_t *);
584 int scf_snaplevel_get_parent(const scf_snaplevel_t *, scf_snapshot_t *);
585 ssize_t scf_snaplevel_get_scope_name(const scf_snaplevel_t *, char *, size_t);
586 ssize_t scf_snaplevel_get_service_name(const scf_snaplevel_t *, char *, size_t);
587 ssize_t scf_snaplevel_get_instance_name(const scf_snaplevel_t *, char *,
588 size_t);
589 int scf_snaplevel_get_pg(const scf_snaplevel_t *, const char *,
590 scf_propertygroup_t *pg);
591 int scf_snapshot_get_base_snaplevel(const scf_snapshot_t *, scf_snaplevel_t *);
592 int scf_snaplevel_get_next_snaplevel(const scf_snaplevel_t *,
593 scf_snaplevel_t *);

595 scf_propertygroup_t *scf_pg_create(scf_handle_t *);
596 scf_handle_t *scf_pg_handle(const scf_propertygroup_t *);
597 void scf_pg_destroy(scf_propertygroup_t *);
598 ssize_t scf_pg_to_fmri(const scf_propertygroup_t *, char *, size_t);
599 ssize_t scf_pg_get_name(const scf_propertygroup_t *, char *, size_t);
600 ssize_t scf_pg_get_type(const scf_propertygroup_t *, char *, size_t);
601 int scf_pg_get_flags(const scf_propertygroup_t *, uint32_t *);
602 int scf_pg_get_parent_service(const scf_propertygroup_t *, scf_service_t *);
603 int scf_pg_get_parent_instance(const scf_propertygroup_t *, scf_instance_t *);
604 int scf_pg_get_parent_snaplevel(const scf_propertygroup_t *, scf_snaplevel_t *);
605 int scf_service_get_pg(const scf_service_t *, const char *,
606 scf_propertygroup_t *);
607 int scf_instance_get_pg(const scf_instance_t *, const char *,
608 scf_propertygroup_t *);
609 int scf_instance_get_pg_composed(const scf_instance_t *, const scf_snapshot_t *,
610 const char *, scf_propertygroup_t *);
611 int scf_service_add_pg(const scf_service_t *, const char *, const char *,
612 uint32_t, scf_propertygroup_t *);
613 int scf_instance_add_pg(const scf_instance_t *, const char *, const char *,
614 uint32_t, scf_propertygroup_t *);
615 int scf_pg_delete(scf_propertygroup_t *);

617 int scf_pg_get_underlying_pg(const scf_propertygroup_t *,
618 scf_propertygroup_t *);
619 int scf_instance_get_parent(const scf_instance_t *, scf_service_t *);

621 int scf_pg_update(scf_propertygroup_t *);

623 scf_property_t *scf_property_create(scf_handle_t *);
624 scf_handle_t *scf_property_handle(const scf_property_t *);
625 void scf_property_destroy(scf_property_t *);
626 int scf_property_is_type(const scf_property_t *, scf_type_t);
627 int scf_property_type(const scf_property_t *, scf_type_t *);
628 ssize_t scf_property_get_name(const scf_property_t *, char *, size_t);
629 int scf_property_get_value(const scf_property_t *, scf_value_t *);
630 ssize_t scf_property_to_fmri(const scf_property_t *, char *, size_t);
631 int scf_pg_get_property(const scf_propertygroup_t *, const char *,
632 scf_property_t *);

634 scf_transaction_t *scf_transaction_create(scf_handle_t *);
635 scf_handle_t *scf_transaction_handle(const scf_transaction_t *);
636 int scf_transaction_start(scf_transaction_t *, scf_propertygroup_t *);
637 void scf_transaction_destroy(scf_transaction_t *);
638 void scf_transaction_destroy_children(scf_transaction_t *);

640 void scf_transaction_reset(scf_transaction_t *);
641 void scf_transaction_reset_all(scf_transaction_t *);

643 int scf_transaction_commit(scf_transaction_t *);

645 scf_transaction_entry_t *scf_entry_create(scf_handle_t *);
646 scf_handle_t *scf_entry_handle(const scf_transaction_entry_t *);
647 void scf_entry_reset(scf_transaction_entry_t *);
648 void scf_entry_destroy(scf_transaction_entry_t *);

```

```

649 void scf_entry_destroy_children(scf_transaction_entry_t *);

651 int scf_transaction_property_change(scf_transaction_t *,
652 scf_transaction_entry_t *, const char *, scf_type_t);
653 int scf_transaction_property_delete(scf_transaction_t *,
654 scf_transaction_entry_t *, const char *);
655 int scf_transaction_property_new(scf_transaction_t *,
656 scf_transaction_entry_t *, const char *, scf_type_t);
657 int scf_transaction_property_change_type(scf_transaction_t *,
658 scf_transaction_entry_t *, const char *, scf_type_t);

660 int scf_entry_add_value(scf_transaction_entry_t *, scf_value_t *);

662 int scf_handle_decode_fmri(scf_handle_t *, const char *, scf_scope_t *,
663 scf_service_t *, scf_instance_t *, scf_propertygroup_t *, scf_property_t *,
664 int);
665 #define SCF_DECODE_FMRI_EXACT 0x00000001
666 #define SCF_DECODE_FMRI_TRUNCATE 0x00000002
667 #define SCF_DECODE_FMRI_REQUIRE_INSTANCE 0x00000004
668 #define SCF_DECODE_FMRI_REQUIRE_NO_INSTANCE 0x00000008

670 ssize_t scf_myname(scf_handle_t *, char *, size_t);

672 /*
673 * Property group template interfaces.
674 */
675 scf_pg_tmpl_t *scf_tmpl_pg_create(scf_handle_t *);
676 void scf_tmpl_pg_destroy(scf_pg_tmpl_t *);
677 void scf_tmpl_pg_reset(scf_pg_tmpl_t *);
678 int scf_tmpl_get_by_pg(scf_propertygroup_t *, scf_pg_tmpl_t *, int);
679 int scf_tmpl_get_by_pg_name(const char *, const char *,
680 const char *, const char *, scf_pg_tmpl_t *, int);
681 int scf_tmpl_iter_pgs(scf_pg_tmpl_t *, const char *, const char *,
682 const char *, int);
683 #define SCF_PG_TMPL_FLAG_REQUIRED 0x1
684 #define SCF_PG_TMPL_FLAG_EXACT 0x2
685 #define SCF_PG_TMPL_FLAG_CURRENT 0x4

687 ssize_t scf_tmpl_pg_name(const scf_pg_tmpl_t *, char **);
688 ssize_t scf_tmpl_pg_common_name(const scf_pg_tmpl_t *, const char *, char **);
689 ssize_t scf_tmpl_pg_description(const scf_pg_tmpl_t *, const char *, char **);
690 ssize_t scf_tmpl_pg_type(const scf_pg_tmpl_t *, char **);

692 ssize_t scf_tmpl_pg_target(const scf_pg_tmpl_t *, char **);
693 #define SCF_TM_TARGET_ALL ((const char *) "all")
694 #define SCF_TM_TARGET_DELEGATE ((const char *) "delegate")
695 #define SCF_TM_TARGET_INSTANCE ((const char *) "instance")
696 #define SCF_TM_TARGET_THIS ((const char *) "this")

698 int scf_tmpl_pg_required(const scf_pg_tmpl_t *, uint8_t *);

700 /*
701 * Property template interfaces.
702 */
703 scf_prop_tmpl_t *scf_tmpl_prop_create(scf_handle_t *);
704 void scf_tmpl_prop_destroy(scf_prop_tmpl_t *);
705 void scf_tmpl_prop_reset(scf_prop_tmpl_t *);
706 int scf_tmpl_get_by_prop(scf_pg_tmpl_t *, const char *,
707 scf_prop_tmpl_t *, int);
708 int scf_tmpl_iter_props(scf_pg_tmpl_t *, scf_prop_tmpl_t *, int);
709 #define SCF_PROP_TMPL_FLAG_REQUIRED 0x1

711 ssize_t scf_tmpl_prop_name(const scf_prop_tmpl_t *, char **);
712 int scf_tmpl_prop_type(const scf_prop_tmpl_t *, scf_type_t *);
713 int scf_tmpl_prop_required(const scf_prop_tmpl_t *, uint8_t *);
714 ssize_t scf_tmpl_prop_common_name(const scf_prop_tmpl_t *, const char *,

```

```

715     char **);
716 ssize_t scf_tmpl_prop_description(const scf_prop_tmpl_t *, const char *,
717     char **);
718 ssize_t scf_tmpl_prop_units(const scf_prop_tmpl_t *, const char *, char **);
719 int scf_tmpl_prop_cardinality(const scf_prop_tmpl_t *prop, uint64_t *,
720     uint64_t *);
721 int scf_tmpl_prop_internal_seps(const scf_prop_tmpl_t *, scf_values_t *);

723 int scf_tmpl_prop_visibility(const scf_prop_tmpl_t *, uint8_t *);
724 #define SCF_TMPL_VISIBILITY_HIDDEN      1
725 #define SCF_TMPL_VISIBILITY_READONLY    2
726 #define SCF_TMPL_VISIBILITY_READWRITE   3

728 const char *scf_tmpl_visibility_to_string(uint8_t);
729 #define SCF_TM_VISIBILITY_HIDDEN        ((const char *)"hidden")
730 #define SCF_TM_VISIBILITY_READONLY      ((const char *)"readonly")
731 #define SCF_TM_VISIBILITY_READWRITE     ((const char *)"readwrite")

733 int scf_tmpl_value_name_constraints(const scf_prop_tmpl_t *prop,
734     scf_values_t *vals);
735 void scf_count_ranges_destroy(scf_count_ranges_t *);
736 void scf_int_ranges_destroy(scf_int_ranges_t *);
737 int scf_tmpl_value_count_range_constraints(const scf_prop_tmpl_t *,
738     scf_count_ranges_t *);
739 int scf_tmpl_value_int_range_constraints(const scf_prop_tmpl_t *,
740     scf_int_ranges_t *);
741 int scf_tmpl_value_count_range_choices(const scf_prop_tmpl_t *,
742     scf_count_ranges_t *);
743 int scf_tmpl_value_int_range_choices(const scf_prop_tmpl_t *,
744     scf_int_ranges_t *);
745 int scf_tmpl_value_name_choices(const scf_prop_tmpl_t *prop,
746     scf_values_t *vals);

748 void scf_values_destroy(scf_values_t *);

750 ssize_t scf_tmpl_value_common_name(const scf_prop_tmpl_t *, const char *,
751     const char *, char **);
752 ssize_t scf_tmpl_value_description(const scf_prop_tmpl_t *, const char *,
753     const char *, char **);

755 int scf_tmpl_value_in_constraint(const scf_prop_tmpl_t *pt, scf_value_t *value,
756     scf_tmpl_errors_t **errs);

758 /*
759  * Template validation interfaces
760  */
761 int scf_tmpl_validate_fmri(scf_handle_t *, const char *,
762     const char *, scf_tmpl_errors_t **, int);
763 #define SCF_TMPL_VALIDATE_FLAG_CURRENT 0x1

765 void scf_tmpl_errors_destroy(scf_tmpl_errors_t **errs);
766 scf_tmpl_error_t *scf_tmpl_next_error(scf_tmpl_errors_t *);
767 void scf_tmpl_reset_errors(scf_tmpl_errors_t **errs);
768 int scf_tmpl_strerror(scf_tmpl_error_t *err, char *s, size_t n, int flag);
769 int scf_tmpl_error_source_fmri(const scf_tmpl_error_t *, char **);
770 int scf_tmpl_error_type(const scf_tmpl_error_t *, scf_tmpl_error_type_t *);
771 int scf_tmpl_error_pg_tmpl(const scf_tmpl_error_t *, char **, char **);
772 int scf_tmpl_error_pg(const scf_tmpl_error_t *, char **, char **);
773 int scf_tmpl_error_prop_tmpl(const scf_tmpl_error_t *, char **, char **);
774 int scf_tmpl_error_prop(const scf_tmpl_error_t *, char **, char **);
775 int scf_tmpl_error_value(const scf_tmpl_error_t *, char **);

777 /*
778  * Simplified calls
779  */
780 int smf_enable_instance(const char *, int);

```

```

781 int smf_disable_instance(const char *, int);
782 int smf_refresh_instance(const char *);
783 int smf_restart_instance(const char *);
784 int smf_maintain_instance(const char *, int);
785 int smf_degrade_instance(const char *, int);
786 int smf_restore_instance(const char *);
787 char *smf_get_state(const char *);

789 int scf_simple_walk_instances(uint_t, void *,
790     int (*inst_callback)(scf_handle_t *, scf_instance_t *, void *));

792 scf_simple_prop_t *scf_simple_prop_get(scf_handle_t *, const char *,
793     const char *, const char *);
794 void scf_simple_prop_free(scf_simple_prop_t *);
795 scf_simple_app_props_t *scf_simple_app_props_get(scf_handle_t *, const char *);
796 void scf_simple_app_props_free(scf_simple_app_props_t *);
797 const scf_simple_prop_t *scf_simple_app_props_next(
798     const scf_simple_app_props_t *, scf_simple_prop_t *);
799 const scf_simple_prop_t *scf_simple_app_props_search(
800     const scf_simple_app_props_t *, const char *, const char *);
801 ssize_t scf_simple_prop_numvalues(const scf_simple_prop_t *);
802 scf_type_t scf_simple_prop_type(const scf_simple_prop_t *);
803 char *scf_simple_prop_name(const scf_simple_prop_t *);
804 char *scf_simple_prop_pname(const scf_simple_prop_t *);
805 uint8_t *scf_simple_prop_next_boolean(const scf_simple_prop_t *);
806 uint64_t *scf_simple_prop_next_count(const scf_simple_prop_t *);
807 int64_t *scf_simple_prop_next_integer(const scf_simple_prop_t *);
808 int64_t *scf_simple_prop_next_time(const scf_simple_prop_t *, int32_t *);
809 char *scf_simple_prop_next_astring(const scf_simple_prop_t *);
810 char *scf_simple_prop_next_ustring(const scf_simple_prop_t *);
811 void *scf_simple_prop_next_opaque(const scf_simple_prop_t *, size_t *);
812 void scf_simple_prop_next_reset(const scf_simple_prop_t *);

814 /*
815  * smf_state_from_string()
816  * return SCF_STATE_* value for the input
817  * -1 on error. String "all" maps to SCF_STATE_ALL macro
818  */
819 int32_t smf_state_from_string(const char *);

821 /*
822  * smf_state_to_string()
823  * return SCF_STATE_STRING* value for the input
824  * NULL on error.
825  */
826 const char *smf_state_to_string(int32_t);

828 /*
829  * Notification interfaces
830  */
831 int smf_notify_set_params(const char *, nvlist_t *);
832 int smf_notify_get_params(nvlist_t **, nvlist_t *);
833 int smf_notify_del_params(const char *, const char *, int32_t);

835 /*
836  * SMF exit status definitions
837  */
838 #define SMF_EXIT_OK                0
839 #define SMF_EXIT_ERR_FATAL          95
840 #define SMF_EXIT_ERR_CONFIG         96
841 #define SMF_EXIT_MON_DEGRADE        97
842 #define SMF_EXIT_MON_OFFLINE        98
843 #define SMF_EXIT_ERR_NOSMF          99
844 #define SMF_EXIT_ERR_PERM           100

846 #ifndef __cplusplus

```

new/usr/src/lib/libscf/inc/libscf.h

11

```
847 }  
848 #endif  
850 #endif /* _LIBSCF_H */
```

new/usr/src/lib/libscf/inc/libscf_priv.h

1

```
*****
20892 Thu Jun 30 22:02:30 2016
new/usr/src/lib/libscf/inc/libscf_priv.h
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
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 (c) 2004, 2010, Oracle and/or its affiliates. All rights reserved.
23 * Copyright 2013, Joyent, Inc. All rights reserved.
24 */

26 #ifndef _LIBSCF_PRIV_H
27 #define _LIBSCF_PRIV_H

30 #include <libscf.h>
31 #include <unistd.h>
32 #if !defined(NATIVE_BUILD)
33 #include <sys/secflags.h>
34 #endif
35 #endif /* ! codereview */

37 #ifdef __cplusplus
38 extern "C" {
39 #endif

41 /*
42  * NOTE
43  *
44  * The contents of this file are private to the implementation of Solaris
45  * and are subject to change at any time without notice.
46  */

48 #define SCF_PG_GENERAL_TYPE          SCF_GROUP_FRAMEWORK
49 #define SCF_PG_GENERAL_FLAGS        0

51 #define SCF_PG_GENERAL_OVR_TYPE      SCF_GROUP_FRAMEWORK
52 #define SCF_PG_GENERAL_OVR_FLAGS    SCF_PG_FLAG_NONPERSISTENT

54 #define SCF_PG_DEATHROW_TYPE         SCF_GROUP_FRAMEWORK
55 #define SCF_PG_DEATHROW_FLAGS       SCF_PG_FLAG_NONPERSISTENT

57 #define SCF_PG_OPTIONS_TYPE          SCF_GROUP_FRAMEWORK
58 #define SCF_PG_OPTIONS_FLAGS        0
```

new/usr/src/lib/libscf/inc/libscf_priv.h

2

```
60 #define SCF_PG_OPTIONS_OVR_TYPE      SCF_GROUP_FRAMEWORK
61 #define SCF_PG_OPTIONS_OVR_FLAGS    SCF_PG_FLAG_NONPERSISTENT

63 #define SCF_PG_RESTARTER_TYPE        SCF_GROUP_FRAMEWORK
64 #define SCF_PG_RESTARTER_FLAGS      SCF_PG_FLAG_NONPERSISTENT

66 #define SCF_PG_RESTARTER_ACTIONS_TYPE SCF_GROUP_FRAMEWORK
67 #define SCF_PG_RESTARTER_ACTIONS_FLAGS SCF_PG_FLAG_NONPERSISTENT

69 #define SCF_PROPERTY_CLEAR            ((const char *)"maint_off")
70 #define SCF_PROPERTY_MAINTENANCE     ((const char *)"maint_on")

72 #define SCF_PROPERTY_LOGFILE         ((const char *)"logfile")
73 #define SCF_PROPERTY_ALT_LOGFILE     ((const char *)"alt_logfile")

75 #define SCF_LEGACY_SERVICE           ((const char *)"smf/legacy_run")

77 #define SCF_LEGACY_PROPERTY_NAME     ((const char *)"name")
78 #define SCF_LEGACY_PROPERTY_INODE   ((const char *)"inode")
79 #define SCF_LEGACY_PROPERTY_SUFFIX   ((const char *)"suffix")

81 #define SCF_FMRI_TYPE_SVC            0x1
82 #define SCF_FMRI_TYPE_FILE          0x2

84 /*
85  * Strings for use in constructing FMRIs
86  */
87 #define SCF_FMRI_SVC_PREFIX           "svc:"
88 #define SCF_FMRI_FILE_PREFIX         "file:"
89 #define SCF_FMRI_SCOPE_PREFIX       "/"
90 #define SCF_FMRI_LOCAL_SCOPE         "localhost"
91 #define SCF_FMRI_SCOPE_SUFFIX        "@localhost"
92 #define SCF_FMRI_SERVICE_PREFIX     "/"
93 #define SCF_FMRI_INSTANCE_PREFIX    ":"
94 #define SCF_FMRI_PROPERTYGRP_PREFIX "properties/"
95 #define SCF_FMRI_PROPERTY_PREFIX    "/"
96 #define SCF_FMRI_LEGACY_PREFIX      "lrc:"

98 /*
99  * slogin Service FMRI
100 */
101 #define SVC_SULOGIN_FMRI ((const char *)"svc:/system/slogin")

103 typedef struct scf_decoration_info {
104     const char *sdi_name;
105     scf_type_t sdi_type;
106     scf_value_t *sdi_value;      /* can be SCF_DECORATE_CLEAR */
107 } scf_decoration_info_t;

109 typedef int (*scf_decoration_func)(const scf_decoration_info_t *, void *);

111 /*
112  * calls a callback function for each decoration on the handle. If the
113  * callback returns 0, the iteration stops and returns 0. If the callback
114  * returns a non-zero value, the iteration continues. After full completion,
115  * *l is returned. On error, -1 is returned.
116  */
117 int _scf_handle_decorations(scf_handle_t *, scf_decoration_func *,
118     scf_value_t *, void *);

120 /*
121  * wait for a change to the propertygroup -- may return early.
122  * For now, only one of these can be outstanding at a time.
123  *
124  * The second argument is how long, in seconds, to wait for a response.
```

```

125 *
126 * Returns SCF_COMPLETE on timeout, -1 on error, and SCF_SUCCESS in every
127 * other case. You must call scf_pg_update() to see if the object has
128 * actually changed.
129 */
130 int _scf_pg_wait(scf_propertygroup_t *, int);

132 /*
133 * set up notifications for changes to a class of property groups (by name
134 * and type)
135 *
136 * Only one thread can be sleeping in _scf_notify_wait() -- others will
137 * fail. Deletions give an fmri in the output path.
138 *
139 * These do not survive unbind()->bind() -- in fact, that is currently the
140 * only way to clear them.
141 */
142 int _scf_notify_add_pgname(scf_handle_t *, const char *);
143 int _scf_notify_add_pgtype(scf_handle_t *, const char *);
144 int _scf_notify_wait(scf_propertygroup_t *, char *, size_t);

146 /*
147 * Internal interfaces for snapshot creation:
148 * _scf_snapshot_take_new(), _scf_snapshot_take_new_named(), and
149 * _scf_snapshot_take_attach() create a set of snaplevels
150 * containing frozen versions of both the instance's property groups and
151 * its parent service's property groups. _scf_snapshot_take_new() and
152 * _scf_snapshot_take_new_named() create a new snapshot to which the
153 * new snaplevels are attached, while _scf_snapshot_take_attach()
154 * attaches the new snaplevels to a pre-existing snapshot.
155 *
156 * _scf_snapshot_take_new_named() records the passed in names into the
157 * snaplevel instead of the instance and service name. This creates
158 * an inconsistency, which should be resolved by using
159 * _scf_snapshot_attach() to attach the new snaplevels to a snapshot
160 * underneath the appropriate instance. The first snapshot can
161 * then be deleted.
162 *
163 * _scf_snapshot_attach(snap1, snap2) points snap2 at the snaplevels
164 * pointed to by snap1. After a call to either
165 * _scf_snapshot_take_attach(snap1, snap2) or
166 * _scf_snapshot_attach(inst, snap), scf_snapshot_update() will be
167 * required for any open references to snap or snap2 to see the new
168 * snaplevels.
169 *
170 * _scf_snapshot_delete() deletes the snapshot object. While
171 * snaplevels, being only loosely connected to snapshots, stay
172 * around until they are no longer referenced, any references *through
173 * this snapshot object* will be invalidated.
174 *
175 * _scf_snapshot_take_new() can fail with at least HANDLE_MISMATCH,
176 * _CONNECTION_BROKEN, _INVALID_ARGUMENT, _NO_RESOURCES, _PERMISSION_DENIED,
177 * _NOT_SET, _EXISTS.
178 *
179 * _scf_snapshot_take_new_named() can fail with at least HANDLE_MISMATCH,
180 * _CONNECTION_BROKEN, _INVALID_ARGUMENT, _NO_RESOURCES, _PERMISSION_DENIED,
181 * _NOT_SET, _EXISTS.
182 *
183 * _scf_snapshot_take_attach() can fail with _CONNECTION_BROKEN, _NOT_SET,
184 * _PERMISSION_DENIED, _NO_RESOURCES, _INVALID_ARGUMENT.
185 *
186 * _scf_snapshot_attach() can fail with HANDLE_MISMATCH, _CONNECTION_BROKEN,
187 * _NOT_SET, _NO_RESOURCES, _PERMISSION_DENIED.
188 */
189 int _scf_snapshot_take_new(scf_instance_t *, const char *, scf_snapshot_t *);
190 int _scf_snapshot_take_new_named(scf_instance_t *,

```

```

191 const char *, const char *, const char *, scf_snapshot_t *);
192 int _scf_snapshot_take_attach(scf_instance_t *, scf_snapshot_t *);
193 int _scf_snapshot_attach(scf_snapshot_t *, scf_snapshot_t *);
194 int _scf_snapshot_delete(scf_snapshot_t *);

196 /*
197 * Destructively portions up the first argument into the different portions
198 * of a svc: fmri, and returns pointers to the applicable portions. Omitted
199 * portions are set to NULL, except for the scope, which is set to the
200 * default local scope if not specified.
201 *
202 * Parsing is attempted in the order of: svc:, file:. The identified type
203 * of the service is returned in the second argument and may take a value
204 * of: SCF_FMRI_TYPE_SVC or SCF_FMRI_TYPE_FILE.
205 *
206 * Note that some of the returned pointers (in particular the scope) may not
207 * point into the passed buffer.
208 */
209 int scf_parse_fmri(char *, int *, const char **, const char **, const char **,
210 const char **, const char **);

212 int scf_parse_svc_fmri(char *, const char **, const char **, const char **,
213 const char **, const char **);

215 int scf_parse_file_fmri(char *fmri, const char **scope, const char **path);

217 ssize_t scf_canonify_fmri(const char *, char *, size_t);

219 int _smf_refresh_instance_i(scf_instance_t *);

221 typedef struct scf_simple_handle {
222     scf_handle_t *h;
223     scf_snapshot_t *snap;
224     scf_instance_t *inst;
225     scf_propertygroup_t *running_pg;
226     scf_propertygroup_t *editing_pg;
227 } scf_simple_handle_t;

229 void scf_simple_handle_destroy(scf_simple_handle_t *);
230 scf_simple_handle_t *scf_general_pg_setup(const char *, const char *);
231 scf_transaction_t *scf_transaction_setup(scf_simple_handle_t *);
232 int scf_transaction_restart(scf_simple_handle_t *, scf_transaction_t *);
233 int scf_read_count_property(scf_simple_handle_t *, char *, uint64_t *);
234 int scf_set_count_property(scf_transaction_t *, char *, uint64_t, boolean_t);

236 /*
237 * Walks all the instances matching a given fmri list. Each fmri in the array
238 * can be one of the following:
239 *
240 * - Full instance name
241 * - Full service name
242 * - Full property group or property name
243 * - Partial service or instance name
244 * - A globbed pattern
245 *
246 * The matching rules for partial fmrirs are a slightly more complex. We allow
247 * for any substring anchored at the end of the instance or service name,
248 * provided it begins with a complete element in the fmri. For example, given
249 * the fmri "svc:/system/filesystem/local:default", any of the following would
250 * be acceptable matches: 'default', 'local', 'local:default',
251 * 'filesystem/local'. The following would not be acceptable:
252 * 'system/filesystem', 'filesystem/loc', 'system/local'. Possible flag values:
253 *
254 * SCF_WALK_MULTIPLE Allow individual arguments to correspond to
255 * multiple instances.
256 */

```

```

257 *   SCF_WALK_LEGACY      Walk legacy services (indicated by a non-NULL
258 *                       property group).
259 *
260 *   SCF_WALK_SERVICE     If the user specifies a service, pass the
261 *                       service to the callback without iterating over
262 *                       its instances.
263 *
264 *   SCF_WALK_PROPERTY     Allow FMRIs which match property groups or
265 *                       individual properties. Incompatible with
266 *                       SCF_WALK_LEGACY.
267 *
268 *   SCF_WALK_NOINSTANCE  Walk only services. Must be used in
269 *                       conjunction with SCF_WALK_SERVICE.
270 *
271 *   SCF_WALK_EXPLICIT    Walk only services if the match is exact
272 *                       else return instances. Must be used in
273 *                       conjunction with SCF_WALK_SERVICE.
274 *
275 *   SCF_WALK_UNIPARTIAL  Can be combined with SCF_WALK_MULTIPLE
276 *                       so that an error is returned if a partial
277 *                       fmri matches multiple instances, unless
278 *                       a wildcard match is also used.
279 *
280 * If no arguments are given, then all instances in the service graph are
281 * walked.
282 *
283 * The second to last parameter is set to UU_EXIT_FATAL if one of the arguments
284 * is an invalid FMRI or matches multiple FMRIs when SCF_WALK_MULTIPLE is not
285 * set.
286 *
287 * The last parameter is a user-supplied error function that is called when
288 * reporting invalid arguments.
289 */

291 #define SCF_WALK_MULTIPLE      0x01
292 #define SCF_WALK_LEGACY      0x02
293 #define SCF_WALK_SERVICE     0x04
294 #define SCF_WALK_PROPERTY     0x08
295 #define SCF_WALK_NOINSTANCE  0x10
296 #define SCF_WALK_EXPLICIT    0x20
297 #define SCF_WALK_UNIPARTIAL  0x40

299 /*
300 * The default locations of the repository dbs
301 */
302 #define REPOSITORY_DB        "/etc/svc/repository.db"
303 #define NONPERSIST_DB       "/etc/svc/volatile/svc_nonpersist.db"
304 #define FAST_REPOSITORY_DB   "/etc/svc/volatile/fast_repository.db"
305 #define REPOSITORY_CHECKPOINT "/etc/svc/volatile/checkpoint_repository.db"

308 typedef struct scf_walkinfo {
309     const char      *fmri;
310     scf_scope_t     *scope;
311     scf_service_t   *svc;
312     scf_instance_t  *inst;
313     scf_propertygroup_t *pg;
314     scf_property_t  *prop;
315     int             count; /* svcprop special */
316 } scf_walkinfo_t;

318 typedef int (*scf_walk_callback)(void *, scf_walkinfo_t *);

320 scf_error_t scf_walk_fmri(scf_handle_t *, int, char **, int,
321     scf_walk_callback, void *, int *, void (*)(const char *, ...));

```

```

323 /*
324 * Requests a backup of the repository with a particular name, which
325 * can be any alphabetic string. Only privileged users can do this.
326 *
327 * Can fail with:
328 *   _NOT_BOUND, _CONNECTION_BROKEN, _PERMISSION_DENIED, _INVALID_ARGUMENT,
329 *   _INTERNAL (path too long, or the backup failed for an odd reason),
330 *   _BACKEND_READONLY (filesystem is still read-only)
331 */
332 int _scf_request_backup(scf_handle_t *, const char *);

334 /*
335 * Repository switch client
336 */
337 int _scf_repository_switch(scf_handle_t *, int);

339 /*
340 * Determines whether a property group requires authorization to read; this
341 * does not in any way reflect whether the caller has that authorization.
342 * To determine that, the caller must attempt to read the value of one of the
343 * group's properties.
344 *
345 * Can fail with:
346 *   _NOT_BOUND, _CONNECTION_BROKEN, _INVALID_ARGUMENT, _INTERNAL,
347 *   _NO_RESOURCES, _CONSTRAINT_VIOLATED, _DELETED.
348 */
349 int _scf_pg_is_read_protected(const scf_propertygroup_t *, boolean_t *);

351 /*
352 * Sets annotation data for SMF audit logging. Once this function has been
353 * set, the next audit record will be preceded by an ADT_smf_annotation
354 * with the information provided in this function. This function is used
355 * to mark operations which comprise multiple primitive operations such as
356 * svccfg import.
357 */
358 int _scf_set_annotation(scf_handle_t *h, const char *operation,
359     const char *file);

361 /*
362 * scf_pattern_t
363 */
364 typedef struct scf_pattern {
365     enum {
366         PATTERN_INVALID, /* Uninitialized state */
367         PATTERN_EXACT,
368         PATTERN_GLOB,
369         PATTERN_PARTIAL
370     } sp_type;
371     char      *sp_arg; /* Original argument */
372     struct scf_match *sp_matches; /* List of matches */
373     int       sp_matchcount; /* # of matches */
374 } scf_pattern_t;

376 int scf_cmp_pattern(char *, scf_pattern_t *);

378 int gen_filenames_from_fmri(const char *, const char *, char *, char *);

380 /*
381 * Interfaces for bulk access to SMF-stored configuration.
382 *
383 * Each scf_propvec_t represents a single property to be read (with
384 * scf_read_propvec) or written (with scf_write_propvec).
385 *
386 * The fields of a scf_propvec_t have the following meanings:
387 *
388 *   pv_prop - the name of the property

```



```

389 *   pv_desc - a description string (optional; to be consumed by the caller)
390 *   pv_type - the type of the property
391 *   pv_ptr - where to store the data read, or a pointer to the data to
392 *           be written
393 *   pv_aux - additional data influencing the interpretation of pv_ptr
394 *
395 * The meaning of pv_ptr and pv_aux depends on the type of property. For:
396 *
397 *   boolean - if pv_aux is 0, pv_ptr is a pointer to a boolean_t
398 *             if pv_aux is non-0, pv_ptr is a pointer to a uint64_t,
399 *             where pv_aux indicates the bit holding the truth value.
400 *   count - pv_ptr is a pointer to a uint64_t; pv_aux is unused
401 *   integer - pv_ptr is a pointer to an int64_t; pv_aux is unused
402 *   time - pv_ptr is a pointer to an scf_time_t; pv_aux is unused
403 *   opaque - pv_ptr is a pointer to an scf_opaque_t; pv_aux is unused
404 *   strings - (scf_read_propvec) pv_ptr is a pointer to a char *
405 *             (scf_write_propvec) pv_ptr is a pointer to an array of char
406 *             (both) pv_aux is unused
407 */
408 typedef struct {
409     void *so_addr;
410     size_t so_size;
411 } scf_opaque_t;
412
413 typedef struct {
414     const char *pv_prop;
415     const char *pv_desc;
416     scf_type_t pv_type;
417     void *pv_ptr;
418     uint64_t pv_aux;
419 } scf_propvec_t;
420
421 void scf_clean_propvec(scf_propvec_t *);
422 int scf_read_propvec(const char *, const char *, boolean_t, scf_propvec_t *,
423     scf_propvec_t **);
424 int scf_write_propvec(const char *, const char *, scf_propvec_t *,
425     scf_propvec_t **);
426
427 scf_tmpl_errors_t *scf_create_errors(const char *, int);
428 int _scf_tmpl_add_error(scf_tmpl_errors_t *errs, scf_tmpl_error_type_t type,
429     const char *pg_name, const char *prop_name,
430     const char *ev1, const char *ev2, const char *actual,
431     const char *tmpl_fmri, const char *tmpl_pg_name, const char *tmpl_pg_type,
432     const char *tmpl_prop_name, const char *tmpl_prop_type);
433 int _scf_tmpl_error_set_prefix(scf_tmpl_errors_t *, const char *);
434
435 /*
436 * Templates definitions
437 */
438
439 /*
440 * For CARDINALITY_VIOLATION and RANGE_VIOLATION, te_ev1 holds
441 * the min value and te_ev2 holds the max value
442 *
443 * For MISSING_PG te_ev1 should hold the expected pg_name and
444 * expected2 holds the expected pg_type.
445 *
446 * For SCF_TERR_PG_PATTERN_CONFLICT and SCF_TERR_GENERAL_REDEFINE te_ev1 is
447 * the FMRI holding the conflicting pg_pattern. te_ev2 is the name of the
448 * conflicting pg_pattern, and actual is the type of the conflicting
449 * pg_pattern.
450 *
451 * SCF_TERR_PROP_PATTERN_CONFLICT te_ev1 is the FMRI holding the
452 * conflicting prop_pattern. te_ev2 is the name of the conflicting
453 * prop_pattern, and actual is the type of the conflicting prop_pattern.
454 */

```

```

455 * For SCF_TERR_INCLUDE_VALUES te_ev1 is the type specified for the
456 * include_values element.
457 *
458 * For all other errors, te_ev1 should hold the expected value and
459 * te_ev2 is ignored
460 *
461 * te_actual holds the current value of the property
462 */
463
464 struct scf_tmpl_error {
465     scf_tmpl_errors_t *te_errs;
466     scf_tmpl_error_type_t te_type;
467     const char *te_pg_name;
468     const char *te_prop_name;
469     const char *te_ev1;
470     const char *te_ev2;
471     const char *te_actual;
472     const char *te_tmpl_fmri;
473     const char *te_tmpl_pg_name;
474     const char *te_tmpl_pg_type;
475     const char *te_tmpl_prop_name;
476     const char *te_tmpl_prop_type;
477 };
478
479 /*
480 * The pg_pattern element has two optional attributes that play a part in
481 * selecting the appropriate prefix for the name of the pg_pattern property
482 * group. The two attributes are name and type. The appropriate prefix
483 * encodes the presence or absence of these attributes.
484 *
485 * SCF_PG_TM_PG_PATTERN_PREFIX neither attribute
486 * SCF_PG_TM_PG_PATTERN_N_PREFIX name only
487 * SCF_PG_TM_PG_PATTERN_T_PREFIX type only
488 * SCF_PG_TM_PG_PATTERN_NT_PREFIX both name and type
489 */
490 #define SCF_PG_TM_PG_PAT_BASE "tm_pgpat"
491 #define SCF_PG_TM_PG_PATTERN_PREFIX ((const char *)SCF_PG_TM_PG_PAT_BASE \
492     " ")
493 #define SCF_PG_TM_PG_PATTERN_N_PREFIX ((const char *)SCF_PG_TM_PG_PAT_BASE \
494     "n ")
495 #define SCF_PG_TM_PG_PATTERN_T_PREFIX ((const char *)SCF_PG_TM_PG_PAT_BASE \
496     "t ")
497 #define SCF_PG_TM_PG_PATTERN_NT_PREFIX ((const char *)SCF_PG_TM_PG_PAT_BASE \
498     "nt ")
499 #define SCF_PG_TM_PROP_PATTERN_PREFIX ((const char *)"tm_proppat ")
500
501 /*
502 * Pad character to use when encoding strings for property names.
503 */
504 #define SCF_ENCODE32_PAD ('-')
505
506 /*
507 * Functions for base 32 encoding/decoding
508 */
509 int scf_decode32(const char *, size_t, char *, size_t, size_t *, char);
510 int scf_encode32(const char *, size_t, char *, size_t, size_t *, char);
511
512 /*
513 * handy functions
514 */
515 /*
516 * _scf_sanitize_locale
517 * Make sure a locale string has only alpha-numeric or '_' characters
518 */
519 void _scf_sanitize_locale(char *);

```

```

521 /*
522 * _scf_read_tmpl_prop_type_as_string()
523 * Handy function to get template property type as a string
524 */
525 char *_scf_read_tmpl_prop_type_as_string(const scf_prop_tmpl_t *);
526 /*
527 * _scf_read_single_astring_from_pg()
528 * Given a property group (pg) and a property name (pn), this function
529 * retrieves an astring value from pg/pn.
530 */
531 char *_scf_read_single_astring_from_pg(scf_propertygroup_t *, const char *);

533 /*
534 * scf_instance_delete_prop()
535 * Given instance, property group, and property, delete the property.
536 */
537 int
538 scf_instance_delete_prop(scf_instance_t *, const char *, const char *);

540 /*
541 * Functions to extract boot config information from FMRI_BOOT_CONFIG
542 */
543 void scf_get_boot_config(uint8_t *);
544 void scf_get_boot_config_ovr(uint8_t *);
545 int scf_is_fastboot_default(void);

547 /*
548 * Set value of "config_ovr/fastreboot_default".
549 */
550 int scf_fastreboot_default_set_transient(booleant_t);

552 /*
553 * scf_is_compatible_type()
554 * Return true if the second type is the same type, or a subtype of the
555 * first.
556 */
557 int scf_is_compatible_type(scf_type_t, scf_type_t);

559 /*
560 * Check an array of services and enable any that don't have the
561 * "application/auto_enable" property set to "false", which is
562 * the interface to turn off this behaviour (see PSARC 2004/739).
563 */
564 void _check_services(char **);

566 /*
567 * _scf_handle_create_and_bind()
568 * convenience function that creates and binds a handle
569 */
570 scf_handle_t *_scf_handle_create_and_bind(scf_version_t);

572 /*
573 * _smf_refresh_all_instances()
574 * refresh all instances of a service
575 * return SCF_SUCCESS or SCF_FAILED on _PERMISSION_DENIED, _BACKEND_ACCESS
576 * or _BACKEND_READONLY.
577 */
578 int _smf_refresh_all_instances(scf_service_t *);

580 /*
581 * _scf_get_fma_notify_params()
582 * Specialized function to get fma notification parameters
583 */
584 int _scf_get_fma_notify_params(const char *, nvlist_t *, int);

586 /*

```

```

587 * _scf_get_svc_notify_params()
588 * Specialized function to get SMF state transition notification parameters
589 */
590 int _scf_get_svc_notify_params(const char *, nvlist_t *, int32_t, int, int);

592 /*
593 * _scf_notify_get_params()
594 * Specialized function to get notification parameters from a pg into an
595 * nvlist_t
596 */
597 int _scf_notify_get_params(scf_propertygroup_t *, nvlist_t *);

599 #if !defined(NATIVE_BUILD)
600 int scf_default_secflags(scf_handle_t *, psecflags_t *);
601 #endif

603 #endif /* !codereview */
604 #define SCF_NOTIFY_PARAMS_SOURCE_NAME ((const char *)"preference_source")

606 #ifdef __cplusplus
607 }
608 #endif

610 #endif /* _LIBSCF_PRIV_H */

```

new/usr/src/lib/libsecdb/auth_attr.txt

1

```
*****
14426 Thu Jun 30 22:02:31 2016
new/usr/src/lib/libsecdb/auth_attr.txt
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
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 (c) 1999, 2010, Oracle and/or its affiliates. All rights reserved.
23 #
24 #
25 #
26 # /etc/security/auth_attr
27 #
28 # authorizations. see auth_attr(4)
29 #
30 solaris.::All Solaris Authorizations::help=AllSolAuthsHeader.html
31 solaris.grant::Grant All Solaris Authorizations::help=PriAdmin.html
32 #
33 solaris.admin.idmap.rules::Manage Identity Mapping Rules::help=IdmapRules.html
34 #
35 solaris.admin.wusb.::Administer Wireless USB::help=WUSBHeader.html
36 solaris.admin.wusb.read::Read Wireless USB Host and Device Information::help=WU
37 solaris.admin.wusb.modify::Add or delete information of Wireless USB Device::he
38 solaris.admin.wusb.host::Manage Wireless USB Host::help=WUSBHost.html
39 #
40 solaris.audit.::Audit System-wide Management::help=AuditHeader.html
41 #
42 solaris.device.::Device Allocation::help=DevAllocHeader.html
43 solaris.device.allocate::Allocate Device::help=DevAllocate.html
44 solaris.device.config::Configure Device Attributes::help=DevConfig.html
45 solaris.device.grant::Delegate Device Administration::help=DevGrant.html
46 solaris.device.revoke::Revoke or Reclaim Device::help=DevRevoke.html
47 solaris.device.cdrw::CD-R/RW Recording Authorizations::help=DevCDRW.html
48 solaris.device.mount.::Device Mount::help=DevMount.html
49 solaris.device.mount.alloptions.fixed::Device Mount Fixed With All Options::hel
50 solaris.device.mount.alloptions.removable::Device Mount Removable With All Opti
51 solaris.device.mount.fixed::Device Mount Fixed::help=DevMount.html
52 solaris.device.mount.removable::Device Mount Removable::help=DevMount.html
53 #
54 solaris.dhcpmgr.::DHCP Service Management::help=DhcpmgrHeader.html
55 solaris.dhcpmgr.write::Modify DHCP Service Configuration::help=DhcpmgrWrite.htm
56 #
57 solaris.file.::File Operations::help=FileHeader.html
58 solaris.file.chown::Change File Owner::help=FileChown.html
```

new/usr/src/lib/libsecdb/auth_attr.txt

2

```
59 solaris.file.owner::Act as File Owner::help=FileOwner.html
60 #
61 solaris.hotplug.::Hotplug::help=HotplugHeader.html
62 solaris.hotplug.modify::Modify Hotplug Connections::help=HotplugModify.html
63 #
64 solaris.jobs.::Job Scheduler::help=JobHeader.html
65 solaris.jobs.admin::Manage All Jobs::help=AuthJobsAdmin.html
66 solaris.jobs.grant::Delegate Cron & At Administration::help=JobsGrant.html
67 solaris.jobs.user::Manage Owned Jobs::help=AuthJobsUser.html
68 #
69 solaris.label.::Label Management::help=LabelHeader.html
70 solaris.label.file.downgrade::Downgrade File Label::help=LabelFileDowngrade.htm
71 solaris.label.file.upgrade::Upgrade File Label::help=LabelFileUpgrade.html
72 solaris.label.print::View Printer Queue at All Labels::help=LabelPrint.html
73 solaris.label.range::Set Label Outside User Accred Range::help=LabelRange.html
74 solaris.label.win.downgrade::Downgrade DragNDrop or CutPaste Info::help=LabelWi
75 solaris.label.win.noview::DragNDrop or CutPaste without viewing contents::help=
76 solaris.label.win.upgrade::Upgrade DragNDrop or CutPaste Info::help=LabelWinUpp
77 #
78 solaris.login.::Login Control::help=LoginHeader.html
79 solaris.login.enable::Enable Logins::help=LoginEnable.html
80 solaris.login.remote::Remote Login::help=LoginRemote.html
81 #
82 solaris.mail.::Mail::help=MailHeader.html
83 solaris.mail.mailq::Mail Queue::help=MailQueue.html
84 #
85 solaris.network.::Network::help=NetworkHeader.html
86 solaris.network.autoconf.read::View Network Auto-Magic Config::help=NetworkAuto
87 solaris.network.autoconf.select::Enable/Disable Network Auto-Magic Config::help
88 solaris.network.autoconf.wlan::Create Network Auto-Magic Config for Known WLANs
89 solaris.network.autoconf.write::Create Network Auto-Magic Config::help=NetworkA
90 solaris.network.ilb.config::Network ILB Configuration::help=NetworkILBconf.html
91 solaris.network.ilb.enable::Network ILB Enable Configuration::help=NetworkILBen
92 solaris.network.interface.config::Network Interface Configuration::help=Network
93 solaris.network.link.security::Link Security::help=LinkSecurity.html
94 solaris.network.wifi.config::Wifi Config::help=WifiConfig.html
95 solaris.network.wifi.wep::Wifi Wep::help=WifiWep.html
96 solaris.network.vrrp::Administer VRRP::help=NetworkVRRP.html
97 #
98 solaris.print.::Printer Management::help=PrintHeader.html
99 solaris.print.admin::Administer Printer::help=PrintAdmin.html
100 solaris.print.cancel::Cancel Print Job::help=PrintCancel.html
101 solaris.print.list::List Jobs in Printer Queue::help=PrintList.html
102 solaris.print.nobanner::Print without Banner::help=PrintNoBanner.html
103 solaris.print.ps::Print Postscript::help=PrintPs.html
104 solaris.print.unlabeled::Print without Label::help=PrintUnlabeled.html
105 #
106 solaris.profmgr.::Rights::help=ProfmgrHeader.html
107 solaris.profmgr.assign::Assign All Rights::help=AuthProfmgrAssign.html
108 solaris.profmgr.delegate::Assign Owned Rights::help=AuthProfmgrDelegate.html
109 solaris.profmgr.write::Manage Rights::help=AuthProfmgrWrite.html
110 solaris.profmgr.read::View Rights::help=AuthProfmgrRead.html
111 solaris.profmgr.execattr.write::Manage Commands::help=AuthProfmgrExecattrWrite.
112 #
113 solaris.role.::Roles::help=RoleHeader.html
114 solaris.role.assign::Assign All Roles::help=AuthRoleAssign.html
115 solaris.role.delegate::Assign Owned Roles::help=AuthRoleDelegate.html
116 solaris.role.write::Manage Roles::help=AuthRoleWrite.html
117 #
118 solaris.smf.::SMF Management::help=SmfHeader.html
119 solaris.smf.modify.::Modify All SMF Service Properties::help=SmfModifyHeader.ht
120 solaris.smf.modify.method::Modify Service Methods::help=SmfModifyMethod.html
121 solaris.smf.modify.dependency::Modify Service Dependencies::help=SmfModifyDepen
122 solaris.smf.modify.application::Modify Application Type Properties::help=SmfMod
123 solaris.smf.modify.framework::Modify Framework Type Properties::help=SmfModifyF
124 solaris.smf.manage.::Manage All SMF Service States::help=SmfManageHeader.html
```

```

125 solaris.smf.manage.allocate::Manage Device Allocation Service::help=SmfAllocate
126 solaris.smf.manage.audit::Manage Audit Service States::help=SmfManageAudit.html
127 solaris.smf.manage.autofs::Manage Automount Service States::help=SmfAutofsState
128 solaris.smf.manage.bind::Manage DNS Service States::help=BindStates.html
129 solaris.smf.manage.coreadm::Manage Coreadm Service States::help=SmfCoreadmState
130 solaris.smf.manage.cron::Manage Cron Service States::help=SmfCronStates.html
131 solaris.smf.manage.discovery.printers.snmp::Manage Network Attached Device Disc
132 solaris.smf.manage.extended-accounting.flow::Manage Flow Extended Accounting Se
133 solaris.smf.manage.extended-accounting.process::Manage Process Extended Account
134 solaris.smf.manage.extended-accounting.flow::Manage Task Extended Accounting Se
135 solaris.smf.manage.hal::Manage HAL Service States::help=SmfHALStates.html
136 solaris.smf.manage.hotplug::Manage Hotplug Service::help=SmfManageHotplug.html
137 solaris.smf.manage.idmap::Manage Identity Mapping Service States::help=SmfIdmap
138 solaris.smf.manage.ilb::Manage Integrated Load Balancer Service States::help=Sm
139 solaris.smf.manage.inetd::Manage inetd and inetd managed services States::help=
140 solaris.smf.manage.ipsec::Manage IPsec Service States::help=SmfIPsecStates.html
141 solaris.smf.manage.labels::Manage label server::help=LabelServer.html
142 solaris.smf.manage.location::Manage Network Location Service States::help=SmfLo
143 solaris.smf.manage.mdns::Manage Multicast DNS Service States::help=SmfMDNSState
144 solaris.smf.manage.name-service-cache::Manage Name Service Cache Daemon Service
145 solaris.smf.manage.nwam::Manage Network Auto-Magic Service States::help=SmfNWAM
146 solaris.smf.manage.power::Manage Power Management Service States::help=SmfPower
147 solaris.smf.manage.smb::Manage SMB Service States::help=SmfSMBStates.html
148 solaris.smf.manage.smbfs::Manage SMB Client States::help=SmfSMBFSStates.html
149 solaris.smf.manage.reparse::Manage Reparse Service States::help=SmfReparseState
150 solaris.smf.manage.rmvolmgr::Manage Rmvolmgr Service States::help=SmfRmvolmgrSt
151 solaris.smf.manage.routing::Manage Routing Service States::help=SmfRoutingState
152 solaris.smf.manage.rpc.bind::Manage RPC Program number mapper::help=SmfRPCBind.
153 solaris.smf.manage.sendmail::Manage Sendmail Service States::help=SmfSendmailSt
154 solaris.smf.manage.smtp-notify::Manage Email Event Notification Agent::
155 solaris.smf.manage.snmp-notify::Manage SNMP Event Notification Agent::
156 solaris.smf.manage.ssh::Manage Secure Shell Service States::help=SmfSshStates.h
157 solaris.smf.manage.stmf::Manage STMF Service States::help=SmfSTMFStates.html
158 solaris.smf.manage.system-log::Manage Syslog Service States::help=SmfSyslogStat
159 solaris.smf.manage.tnctl::Manage Refresh of Trusted Network Parameters::help=TN
160 solaris.smf.manage.tnd::Manage Trusted Network Daemon::help=TNDaemon.html
161 solaris.smf.manage.vrrp::Manage VRRP Service States::help=SmfVRRPStates.html
162 solaris.smf.manage.vscan::Manage VSCAN Service States::help=SmfVscanStates.html
163 solaris.smf.manage.vt::Manage Virtual Console Service States::help=SmfVtStates.
164 solaris.smf.manage.wpa::Manage WPA Service States::help=SmfWpaStates.html
165 solaris.smf.manage.ndmp::Manage NDMP Service States::help=SmfNDMPStates.html
166 solaris.smf.value::Change Values of SMF Service Properties::help=SmfValueHeade
167 solaris.smf.value.audit::Configure the Audit Service::help=SmfValueAudit.html
168 solaris.smf.value.coreadm::Change Values of SMF Coreadm Properties::help=SmfVal
169 solaris.smf.value.discovery.printers.snmp::Manage Network Attached Device Disco
170 solaris.smf.value.extended-accounting.flow::Change Values of Flow Extended Acco
171 solaris.smf.value.extended-accounting.process::Change Values of Process Extende
172 solaris.smf.value.extended-accounting.task::Change Values of Task Extended Acco
173 solaris.smf.value.firewall.config::Change Service Firewall Config::help=SmfValu
174 solaris.smf.value.idmap::Change Values of SMF Identity Mapping Service Properti
175 solaris.smf.value.inetd::Change values of SMF Inetd configuration parameters::h
176 solaris.smf.value.ipsec::Change Values of SMF IPsec Properties::help=SmfValueIP
177 solaris.smf.value.mdns::Change Values of MDNS Service Properties::help=SmfValue
178 solaris.smf.value.nwam::Change Values of SMF Network Auto-Magic Properties::hel
179 solaris.smf.value.process-security::Change Values of Process Security propertie
180 #endif /* ! codereview */
181 solaris.smf.value.smb::Change Values of SMB Service Properties::help=SmfValueSMB
182 solaris.smf.read.smb::Read permission for protected SMF SMB Service Properties:
183 solaris.smf.value.smtp-notify::Change values of Email Event Notification Agent
184 solaris.smf.value.snmp-notify::Change values of SNMP Event Notification Agent p
185 solaris.smf.read.stmf::Read STMF Provider Private Data::help=SmfSTMFRead.html
186 solaris.smf.value.routing::Change Values of SMF Routing Properties::help=SmfVal
187 solaris.smf.value.tnd::Change Trusted Network Daemon Service Property Values::h
188 solaris.smf.value.vscan::Change Values of VSCAN Properties::help=SmfValueVscan.
189 solaris.smf.value.vt::Change Values of Virtual Console Service Properties::help
190 solaris.smf.value.ndmp::Change Values of SMF NDMP Service Properties::help=SmfV

```

```

191 solaris.smf.read.ndmp::Read permission for protected SMF NDMP Service Propertie
192 #
193 solaris.system::Machine Administration::help=SysHeader.html
194 solaris.system.date::Set Date & Time::help=SysDate.html
195 solaris.system.maintenance::Enter Maintenance (single-user) Mode::help=SysMaint
196 solaris.system.shutdown::Shutdown the System::help=SysShutdown.html
197 solaris.system.power::System Power Management::help=SysPowerMgmtHeader.html
198 solaris.system.power.suspend::Suspend the System::help=SysPowerMgmtSuspend.htm
199 solaris.system.power.suspend.disk::Suspend to Disk::help=SysPowerMgmtSuspendtoD
200 solaris.system.power.suspend.ram::Suspend to RAM::help=SysPowerMgmtSuspendtoRAM
201 solaris.system.power.brightness::Control LCD Brightness::help=SysPowerMgmtBrigh
202 solaris.system.power.cpu::Manage CPU related power::help=SysCpuPowerMgmt.html
203 solaris.system.sysevent.read::Retrieve Sysevents::help=SysSyseventRead.html
204 solaris.system.sysevent.write::Publish Sysevents::help=SysSyseventWrite.html
205 #
206 solaris.smf.modify.stmf::Modify STMF Properties::help=SmfSTMFValue.html
207 #
208 solaris.smf.manage.isns::Manage iSNS Service States::help=isnsStates.html
209 solaris.smf.value.isns::Modify iSNS Service Property Values::help=isnsValue.htm
210 solaris.isnsmgr.write::Modify iSNS configuration::help=AuthISNSMgrWrite.html
211 solaris.smf.manage.wusb::Manage Wireless USB Service::help=SmfWusbStates.html
212 solaris.zone::Zone Management::help=ZoneHeader.html
213 solaris.zone.clonefrom::Clone another Zone::help=ZoneCloneFrom.html
214 solaris.zone.login::Zone Login::help=ZoneLogin.html
215 solaris.zone.manage::Zone Deployment::help=ZoneManage.html

```

new/usr/src/lib/libsecdb/help/auths/Makefile

1

```
*****
4783 Thu Jun 30 22:02:32 2016
new/usr/src/lib/libsecdb/help/auths/Makefile
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
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 (c) 1999, 2010, Oracle and/or its affiliates. All rights reserved.
23 #
24 #
25 # lib/libsecdb/help/auths/Makefile
26 #
27 #
28 include ../../../../Makefile.master
29 #
30 HTMLENTS = \
31     AuditHeader.html \
32     DevAllocHeader.html \
33     DevAllocate.html \
34     DevConfig.html \
35     DevCDRW.html \
36     DevGrant.html \
37     DevRevoke.html \
38     HotplugHeader.html \
39     HotplugModify.html \
40     JobHeader.html \
41     AuthJobsAdmin.html \
42     JobsGrant.html \
43     AuthJobsUser.html \
44     LoginEnable.html \
45     LoginHeader.html \
46     LoginRemote.html \
47     MailHeader.html \
48     MailQueue.html \
49     PriAdmin.html \
50     AuthProfmgrAssign.html \
51     AuthProfmgrDelegate.html \
52     AuthProfmgrExecattrWrite.html \
53     AuthProfmgrRead.html \
54     ProfmgrHeader.html \
55     AuthProfmgrWrite.html \
56     AuthRoleAssign.html \
57     AuthRoleDelegate.html \
58     RoleHeader.html \
```

new/usr/src/lib/libsecdb/help/auths/Makefile

2

```
59     AuthRoleWrite.html \
60     SysDate.html \
61     SysHeader.html \
62     SysShutdown.html \
63     AllSolAuthsHeader.html \
64     SysMaintenance.html \
65     DhcpmgrHeader.html \
66     DhcpmgrWrite.html \
67     BindStates.html \
68     SmfAllocate.html \
69     SmfAutofsStates.html \
70     SmfCoreadmStates.html \
71     SmfCronStates.html \
72     SmfExAcctFlowStates.html \
73     SmfExAcctProcessStates.html \
74     SmfExAcctTaskStates.html \
75     SmfExAcctNetStates.html \
76     SmfHeader.html \
77     SmfILBStates.html \
78     SmfInetdStates.html \
79     SmfIPsecStates.html \
80     SmfLocationStates.html \
81     SmfManageAudit.html \
82     SmfManageHeader.html \
83     SmfManageHotplug.html \
84     SmfMDNSStates.html \
85     SmfModifyAppl.html \
86     SmfModifyDepend.html \
87     SmfModifyFramework.html \
88     SmfModifyHeader.html \
89     SmfModifyMethod.html \
90     SmfNscdStates.html \
91     SmfNADDStates.html \
92     SmfNDMPStates.html \
93     SmfNWAMStates.html \
94     SmfPowerStates.html \
95     SmfReparseStates.html \
96     SmfRoutingStates.html \
97     SmfSendmailStates.html \
98     SmfSshStates.html \
99     SmfSyslogStates.html \
100     SmfValueAudit.html \
101     SmfValueCoreadm.html \
102     SmfValueExAcctFlow.html \
103     SmfValueExAcctProcess.html \
104     SmfValueExAcctTask.html \
105     SmfValueExAcctNet.html \
106     SmfValueFirewall.html \
107     SmfVtStates.html \
108     SmfValueHeader.html \
109     SmfValueInetd.html \
110     SmfValueIPsec.html \
111     SmfValueMDNS.html \
112     SmfValueNADD.html \
113     SmfValueNDMP.html \
114     AuthReadNDMP.html \
115     SmfValueNWAM.html \
116     SmfValueProcSec.html \
117 #endif /* !codereview */
118     SmfValueRouting.html \
119     SmfValueSMB.html \
120     AuthReadSMB.html \
121     SmfSMBFSStates.html \
122     SmfSMBStates.html \
123     SmfValueVscan.html \
124     SmfVscanStates.html \
```

```

125 SmfValueVt.html \
126 SmfVRRPStates.html \
127 SmfWpaStates.html \
128 NetworkAutoconfRead.html \
129 NetworkAutoconfSelect.html \
130 NetworkAutoconfWlan.html \
131 NetworkAutoconfWrite.html \
132 NetworkILBconf.html \
133 NetworkILBenable.html \
134 NetworkHeader.html \
135 NetworkVRRP.html \
136 NetworkInterfaceConfig.html \
137 WifiConfig.html \
138 WifiWep.html \
139 LinkSecurity.html \
140 IdmapRules.html \
141 SmfIdmapStates.html \
142 SmfValueIdmap.html \
143 FileChown.html \
144 FileHeader.html \
145 FileOwner.html \
146 LabelFileDowngrade.html \
147 LabelFileUpgrade.html \
148 LabelHeader.html \
149 LabelPrint.html \
150 LabelRange.html \
151 LabelServer.html \
152 LabelWinDowngrade.html \
153 LabelWinNoView.html \
154 LabelWinUpgrade.html \
155 PrintAdmin.html \
156 PrintCancel.html \
157 PrintHeader.html \
158 PrintList.html \
159 PrintNoBanner.html \
160 PrintPs.html \
161 PrintUnlabeled.html \
162 TNDaemon.html \
163 TNctl.html \
164 ValueTND.html \
165 SysPowerMgmtHeader.html \
166 SysPowerMgmtSuspend.html \
167 SysPowerMgmtSuspendtoDisk.html \
168 SysPowerMgmtSuspendtoRAM.html \
169 SysPowerMgmtBrightness.html \
170 SysCpuPowerMgmt.html \
171 SysSyseventRead.html \
172 SysSyseventWrite.html \
173 SmfManageZFSSnap.html \
174 ZoneCloneFrom.html \
175 ZoneHeader.html \
176 ZoneLogin.html \
177 ZoneManage.html

179 HELPDIR=$(ROOT)/usr/lib/help
180 AUTHDIR=$(HELPDIR)/auths
181 LOCALEDIR=$(AUTHDIR)/locale
182 CDIR=$(LOCALEDIR)/C
183 DIRS=$(HELPDIR) $(AUTHDIR) $(LOCALEDIR) $(CDIR)
184 HELPFILES=$(HTMLENTS:%=$(CDIR)/%)

186 MSGDIR= $(LOCALEDIR)
187 MSGDIRS = $(HELPDIR) $(AUTHDIR) $(LOCALEDIR)

189 MSGFILES= $(HTMLENTS)
190 MSGS= $(MSGFILES:%=$(MSGDIR)/%)

```

```

192 FILEMODE= 0444

194 .KEEP_STATE:

196 all: $(HTMLENTS)

198 install: all $(DIRS) $(HELPFILES)

200 _msg: $(MSGDIRS) $(MSGS)

202 $(CDIR)/%: %
203 $(INS.file)

205 $(DIRS):
206 $(INS.dir)

208 $(MSGDIR)/%: %
209 $(INS.file)

211 clean clobber lint:

```

new/usr/src/lib/libsecdb/help/auths/SmfValueProcSec.html

1

855 Thu Jun 30 22:02:33 2016

new/usr/src/lib/libsecdb/help/auths/SmfValueProcSec.html
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.

```
1 <HTML>
2 <!--
3 Copyright 2015, Richard Lowe.
4
5 This file and its contents are supplied under the terms of the
6 Common Development and Distribution License ("CDDL"), version 1.0.
7 You may only use this file in accordance with the terms of version
8 1.0 of the CDDL.

10 A full copy of the text of the CDDL should have accompanied this
11 source. A copy of the CDDL is also available via the Internet at
12 http://www.illumos.org/license/CDDL.
13 -->
14 <!--
15 <META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=iso-8859-1">
16 -->
17 <BODY>
18 When Value Process Security Properties is in the Authorizations Include
19 column, it grants the the authorization to change the default security-flags
20 for processes on this system
21 <P>
22 If Value Process Security Properties is grayed, then you are not entitled to
23 Add or Remove this authorization.
24 <BR>&nbsp;
25 </BODY>
26 </HTML>
27 #endif /* ! codereview */
```

```

*****
205446 Thu Jun 30 22:02:33 2016
new/usr/src/lib/libzonecfg/common/libzonecfg.c
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */

22 /*
23  * Copyright 2014 Gary Mills
24  * Copyright (c) 2003, 2010, Oracle and/or its affiliates. All rights reserved.
25  * Copyright 2015 Nexenta Systems, Inc. All rights reserved.
26 */

28 #include <libsysevent.h>
29 #include <pthread.h>
30 #include <stdlib.h>
31 #include <errno.h>
32 #include <fnmatch.h>
33 #include <strings.h>
34 #include <unistd.h>
35 #include <assert.h>
36 #include <libgen.h>
37 #include <libintl.h>
38 #include <alloca.h>
39 #include <ctype.h>
40 #include <sys/acl.h>
41 #include <sys/stat.h>
42 #include <sys/brand.h>
43 #include <sys/mntio.h>
44 #include <sys/mnttab.h>
45 #include <sys/nvpair.h>
46 #include <sys/types.h>
47 #include <sys/sockio.h>
48 #include <sys/systeminfo.h>
49 #include <ftw.h>
50 #include <pool.h>
51 #include <libscf.h>
52 #include <libproc.h>
53 #include <sys/priocntl.h>
54 #include <libuutil.h>
55 #include <wait.h>
56 #include <bsm/adt.h>
57 #include <auth_attr.h>
58 #include <auth_list.h>

```

```

59 #include <secdb.h>
60 #include <user_attr.h>
61 #include <prof_attr.h>

63 #include <arpa/inet.h>
64 #include <netdb.h>

66 #include <libxml/xmlmemory.h>
67 #include <libxml/parser.h>

69 #include <libdevinfo.h>
70 #include <uuid/uuid.h>
71 #include <dirent.h>
72 #include <libbrand.h>

74 #include <libzonecfg.h>
75 #include "zonecfg_impl.h"

77 #define _PATH_TMPFILE "/zonecfg.XXXXXX"
78 #define ZONE_CB_RETRY_COUNT 10
79 #define ZONE_EVENT_PING_SUBCLASS "ping"
80 #define ZONE_EVENT_PING_PUBLISHER "solaris"

82 /* Hard-code the DTD element/attribute/entity names just once, here. */
83 #define DTD_ELEM_ATTR (const xmlChar *) "attr"
84 #define DTD_ELEM_COMMENT (const xmlChar *) "comment"
85 #define DTD_ELEM_DEVICE (const xmlChar *) "device"
86 #define DTD_ELEM_FS (const xmlChar *) "filesystem"
87 #define DTD_ELEM_FSOPTION (const xmlChar *) "fsoption"
88 #define DTD_ELEM_NET (const xmlChar *) "network"
89 #define DTD_ELEM_RCTL (const xmlChar *) "rctl"
90 #define DTD_ELEM_RCTLVALUE (const xmlChar *) "rctl-value"
91 #define DTD_ELEM_ZONE (const xmlChar *) "zone"
92 #define DTD_ELEM_DATASET (const xmlChar *) "dataset"
93 #define DTD_ELEM_TMPPOOL (const xmlChar *) "tmp_pool"
94 #define DTD_ELEM_PSET (const xmlChar *) "pset"
95 #define DTD_ELEM_MCAP (const xmlChar *) "mcap"
96 #define DTD_ELEM_PACKAGE (const xmlChar *) "package"
97 #define DTD_ELEM_OBSOLETES (const xmlChar *) "obsoletes"
98 #define DTD_ELEM_DEV_PERM (const xmlChar *) "dev-perm"
99 #define DTD_ELEM_ADMIN (const xmlChar *) "admin"
100 #define DTD_ELEM_SECFLAGS (const xmlChar *) "security-flags"
101 #endif /* ! codereview */

103 #define DTD_ATTR_ACTION (const xmlChar *) "action"
104 #define DTD_ATTR_ADDRESS (const xmlChar *) "address"
105 #define DTD_ATTR_ALLOWED_ADDRESS (const xmlChar *) "allowed-address"
106 #define DTD_ATTR_AUTOBOOT (const xmlChar *) "autoboot"
107 #define DTD_ATTR_IPTYPE (const xmlChar *) "ip-type"
108 #define DTD_ATTR_DEFROUTER (const xmlChar *) "defrouter"
109 #define DTD_ATTR_DIR (const xmlChar *) "directory"
110 #define DTD_ATTR_LIMIT (const xmlChar *) "limit"
111 #define DTD_ATTR_LIMITPRIV (const xmlChar *) "limitpriv"
112 #define DTD_ATTR_BOOTARGS (const xmlChar *) "bootargs"
113 #define DTD_ATTR_SCHED (const xmlChar *) "scheduling-class"
114 #define DTD_ATTR_MATCH (const xmlChar *) "match"
115 #define DTD_ATTR_NAME (const xmlChar *) "name"
116 #define DTD_ATTR_PHYSICAL (const xmlChar *) "physical"
117 #define DTD_ATTR_POOL (const xmlChar *) "pool"
118 #define DTD_ATTR_PRIV (const xmlChar *) "priv"
119 #define DTD_ATTR_RAW (const xmlChar *) "raw"
120 #define DTD_ATTR_SPECIAL (const xmlChar *) "special"
121 #define DTD_ATTR_TYPE (const xmlChar *) "type"
122 #define DTD_ATTR_VALUE (const xmlChar *) "value"
123 #define DTD_ATTR_ZONEPATH (const xmlChar *) "zonepath"
124 #define DTD_ATTR_NCPU_MIN (const xmlChar *) "ncpu_min"

```



```

125 #define DTD_ATTR_NCPU_MAX      (const xmlChar *) "ncpu_max"
126 #define DTD_ATTR_IMPORTANCE   (const xmlChar *) "importance"
127 #define DTD_ATTR_PHYSICAP     (const xmlChar *) "physicap"
128 #define DTD_ATTR_VERSION      (const xmlChar *) "version"
129 #define DTD_ATTR_ID           (const xmlChar *) "id"
130 #define DTD_ATTR_UID          (const xmlChar *) "uid"
131 #define DTD_ATTR_GID          (const xmlChar *) "gid"
132 #define DTD_ATTR_MODE         (const xmlChar *) "mode"
133 #define DTD_ATTR_ACL          (const xmlChar *) "acl"
134 #define DTD_ATTR_BRAND        (const xmlChar *) "brand"
135 #define DTD_ATTR_HOSTID       (const xmlChar *) "hostid"
136 #define DTD_ATTR_USER         (const xmlChar *) "user"
137 #define DTD_ATTR_AUTHS        (const xmlChar *) "auths"
138 #define DTD_ATTR_FS_ALLOWED   (const xmlChar *) "fs-allowed"
139 #define DTD_ATTR_DEFAULT      (const xmlChar *) "default"
140 #define DTD_ATTR_LOWER        (const xmlChar *) "lower"
141 #define DTD_ATTR_UPPER        (const xmlChar *) "upper"

143 #endif /* ! codereview */

145 #define DTD_ENTITY_BOOLEAN     "boolean"
146 #define DTD_ENTITY_DEVPATH    "devpath"
147 #define DTD_ENTITY_DRIVER     "driver"
148 #define DTD_ENTITY_DRVMIN     "drv_min"
149 #define DTD_ENTITY_FALSE     "false"
150 #define DTD_ENTITY_INT        "int"
151 #define DTD_ENTITY_STRING     "string"
152 #define DTD_ENTITY_TRUE       "true"
153 #define DTD_ENTITY_UINT       "uint"

155 #define DTD_ENTITY_BOOL_LEN   6      /* "false" */

157 #define ATTACH_FORCED         "SUNWattached.xml"

159 #define TMP_POOL_NAME         "SUNWtmp_%s"
160 #define MAX_TMP_POOL_NAME     (ZONENAME_MAX + 9)
161 #define RCAP_SERVICE          "system/rcap:default"
162 #define POOLD_SERVICE         "system/pools/dynamic:default"

164 /*
165  * rctl alias definitions
166  *
167  * This holds the alias, the full rctl name, the default priv value, action
168  * and lower limit. The functions that handle rctl aliases step through
169  * this table, matching on the alias, and using the full values for setting
170  * the rctl entry as well the limit for validation.
171  */
172 static struct alias {
173     char *shortname;
174     char *realname;
175     char *priv;
176     char *action;
177     uint64_t low_limit;
178 } aliases[] = {
179     {ALIAS_MAXLWPS, "zone.max-lwps", "privileged", "deny", 100},
180     {ALIAS_MAXSHMMEM, "zone.max-shm-memory", "privileged", "deny", 0},
181     {ALIAS_MAXSHMIDS, "zone.max-shm-ids", "privileged", "deny", 0},
182     {ALIAS_MAXMSGIDS, "zone.max-msg-ids", "privileged", "deny", 0},
183     {ALIAS_MAXSEMIDS, "zone.max-sem-ids", "privileged", "deny", 0},
184     {ALIAS_MAXLOCKEDMEM, "zone.max-locked-memory", "privileged", "deny", 0},
185     {ALIAS_MAXSWAP, "zone.max-swap", "privileged", "deny", 0},
186     {ALIAS_SHARES, "zone.cpu-shares", "privileged", "none", 0},
187     {ALIAS_CPUCAP, "zone.cpu-cap", "privileged", "deny", 0},
188     {ALIAS_MAXPROCS, "zone.max-processes", "privileged", "deny", 100},
189     {NULL, NULL, NULL, NULL, 0}
190 };

```

```

192 /*
193  * Structure for applying rctls to a running zone. It allows important
194  * process values to be passed together easily.
195  */
196 typedef struct pr_info_handle {
197     struct ps_prochandle *pr;
198     pid_t pid;
199 } pr_info_handle_t;

201 struct zone_dochandle {
202     char *zone_dh_rootdir;
203     xmlDocPtr zone_dh_doc;
204     xmlNodePtr zone_dh_cur;
205     xmlNodePtr zone_dh_top;
206     boolean_t zone_dh_newzone;
207     boolean_t zone_dh_snapshot;
208     boolean_t zone_dh_sw_inv;
209     zone_userauths_t *zone_dh_userauths;
210     char zone_dh_delete_name[ZONENAME_MAX];
211 };

213 struct znotify {
214     void *zn_private;
215     evchan_t *zn_eventchan;
216     int (*zn_callback)(const char *zonename, zoneid_t zid,
217         const char *newstate, const char *oldstate, hrttime_t when, void *p);
218     pthread_mutex_t zn_mutex;
219     pthread_cond_t zn_cond;
220     pthread_mutex_t zn_bigmutex;
221     volatile enum {ZN_UNLOCKED, ZN_LOCKED, ZN_PING_INFLIGHT,
222         ZN_PING_RECEIVED} zn_state;
223     char zn_subscriber_id[MAX_SUBID_LEN];
224     volatile boolean_t zn_failed;
225     int zn_failure_count;
226 };

228 /* used to track nested zone-lock operations */
229 static int zone_lock_cnt = 0;

231 /* used to communicate lock status to children */
232 #define LOCK_ENV_VAR         "_ZONEADM_LOCK_HELD"
233 static char zoneadm_lock_held[] = LOCK_ENV_VAR="1";
234 static char zoneadm_lock_not_held[] = LOCK_ENV_VAR="0";

236 char *zonecfg_root = "";

238 /*
239  * For functions which return int, which is most of the functions herein,
240  * the return values should be from the Z_foo set defined in <libzonecfg.h>.
241  * In some instances, we take pains mapping some libc errno values to Z_foo
242  * values from this set.
243  */

245 /*
246  * Set the root (/) path for all zonecfg configuration files. This is a
247  * private interface used by Live Upgrade extensions to access zone
248  * configuration inside mounted alternate boot environments.
249  * This interface is also used by zoneadm mount and unmount subcommands.
250  */
251 void
252 zonecfg_set_root(const char *rootpath)
253 {
254     if (*zonecfg_root != '\0')
255         free(zonecfg_root);
256     if (rootpath == NULL || rootpath[0] == '\0' || rootpath[1] == '\0' ||

```

```

257     (zonecfg_root = strdup(rootpath)) == NULL)
258     zonecfg_root = "";
259 }

261 const char *
262 zonecfg_get_root(void)
263 {
264     return (zonecfg_root);
265 }

267 boolean_t
268 zonecfg_in_alt_root(void)
269 {
270     return (*zonecfg_root != '\0');
271 }

273 /*
274  * Callers of the _file_path() functions are expected to have the second
275  * parameter be a (char foo[MAXPATHLEN]).
276  */

278 static boolean_t
279 config_file_path(const char *zonename, char *answer)
280 {
281     return (snprintf(answer, MAXPATHLEN, "%s%s/%s.xml", zonecfg_root,
282                     ZONE_CONFIG_ROOT, zonename) < MAXPATHLEN);
283 }

285 static boolean_t
286 snap_file_path(const char *zonename, char *answer)
287 {
288     return (snprintf(answer, MAXPATHLEN, "%s%s/%s.snapshot.xml",
289                     zonecfg_root, ZONE_SNAPSHOT_ROOT, zonename) < MAXPATHLEN);
290 }

292 /*ARGSUSED*/
293 static void
294 zonecfg_error_func(void *ctx, const char *msg, ...)
295 {
296     /*
297      * This function does nothing by design. Its purpose is to prevent
298      * libxml from dumping unwanted messages to stdout/stderr.
299      */
300 }

302 zone_dochandle_t
303 zonecfg_init_handle(void)
304 {
305     zone_dochandle_t handle = calloc(1, sizeof (struct zone_dochandle));
306     if (handle == NULL) {
307         errno = Z_NOMEM;
308         return (NULL);
309     }

311     /* generic libxml initialization */
312     (void) xmlLineNumbersDefault(1);
313     xmlLoadExtDtdDefaultValue |= XML_DETECT_IDS;
314     xmlDoValidityCheckingDefaultValue = 1;
315     (void) xmlKeepBlanksDefault(0);
316     xmlGetWarningsDefaultValue = 0;
317     xmlSetGenericErrorFunc(NULL, zonecfg_error_func);

319     return (handle);
320 }

322 int

```

```

323 zonecfg_check_handle(zone_dochandle_t handle)
324 {
325     if (handle == NULL || handle->zone_dh_doc == NULL)
326         return (Z_BAD_HANDLE);
327     return (Z_OK);
328 }

330 void
331 zonecfg_fini_handle(zone_dochandle_t handle)
332 {
333     if (zonecfg_check_handle(handle) == Z_OK)
334         xmlFreeDoc(handle->zone_dh_doc);
335     if (handle != NULL)
336         free(handle);
337 }

339 static int
340 zonecfg_destroy_impl(char *filename)
341 {
342     if (unlink(filename) == -1) {
343         if (errno == EACCES)
344             return (Z_ACCES);
345         if (errno == ENOENT)
346             return (Z_NO_ZONE);
347         return (Z_MISC_FS);
348     }
349     return (Z_OK);
350 }

352 int
353 zonecfg_destroy(const char *zonename, boolean_t force)
354 {
355     char path[MAXPATHLEN];
356     struct zoneent ze;
357     int err, state_err;
358     zone_state_t state;

360     if (!config_file_path(zonename, path))
361         return (Z_MISC_FS);

363     state_err = zone_get_state((char *)zonename, &state);
364     err = access(path, W_OK);

366     /*
367      * If there is no file, and no index entry, reliably indicate that no
368      * such zone exists.
369      */
370     if ((state_err == Z_NO_ZONE) && (err == -1) && (errno == ENOENT))
371         return (Z_NO_ZONE);

373     /*
374      * Handle any other filesystem related errors (except if the XML
375      * file is missing, which we treat silently), unless we're forcing,
376      * in which case we plow on.
377      */
378     if (err == -1 && errno != ENOENT) {
379         if (errno == EACCES)
380             return (Z_ACCES);
381         else if (!force)
382             return (Z_MISC_FS);
383     }

385     if (state > ZONE_STATE_INSTALLED)
386         return (Z_BAD_ZONE_STATE);

388     if (!force && state > ZONE_STATE_CONFIGURED)

```

```

389         return (Z_BAD_ZONE_STATE);
391     /*
392     * Index deletion succeeds even if the entry doesn't exist.  So this
393     * will fail only if we've had some more severe problem.
394     */
395     bzero(&ze, sizeof (ze));
396     (void) strncpy(ze.zone_name, zonename, sizeof (ze.zone_name));
397     if ((err = putzoneent(&ze, PZE_REMOVE)) != Z_OK)
398         if (!force)
399             return (err);
401     err = zonecfg_destroy_impl(path);
403     /*
404     * Treat failure to find the XML file silently, since, well, it's
405     * gone, and with the index file cleaned up, we're done.
406     */
407     if (err == Z_OK || err == Z_NO_ZONE)
408         return (Z_OK);
409     return (err);
410 }
412 int
413 zonecfg_destroy_snapshot(const char *zonename)
414 {
415     char path[MAXPATHLEN];
417     if (!snap_file_path(zonename, path))
418         return (Z_MISC_FS);
419     return (zonecfg_destroy_impl(path));
420 }
422 static int
423 getroot(zone_dochandle_t handle, xmlNodePtr *root)
424 {
425     if (zonecfg_check_handle(handle) == Z_BAD_HANDLE)
426         return (Z_BAD_HANDLE);
428     *root = xmlDocGetRootElement(handle->zone_dh_doc);
430     if (*root == NULL)
431         return (Z_EMPTY_DOCUMENT);
433     if (xmlStrcmp((*root)->name, DTD_ELEM_ZONE))
434         return (Z_WRONG_DOC_TYPE);
436     return (Z_OK);
437 }
439 static int
440 operation_prep(zone_dochandle_t handle)
441 {
442     xmlNodePtr root;
443     int err;
445     if ((err = getroot(handle, &root)) != 0)
446         return (err);
448     handle->zone_dh_cur = root;
449     handle->zone_dh_top = root;
450     return (Z_OK);
451 }
453 static int
454 fetchprop(xmlNodePtr cur, const xmlChar *propname, char *dst, size_t dstsize)

```

```

455 {
456     xmlChar *property;
457     size_t srcsize;
459     if ((property = xmlGetProp(cur, propname)) == NULL)
460         return (Z_BAD_PROPERTY);
461     srcsize = strlen(property);
462     xmlFree(property);
463     if (srcsize >= dstsize)
464         return (Z_TOO_BIG);
465     return (Z_OK);
466 }
468 static int
469 fetch_alloc_prop(xmlNodePtr cur, const xmlChar *propname, char **dst)
470 {
471     xmlChar *property;
473     if ((property = xmlGetProp(cur, propname)) == NULL)
474         return (Z_BAD_PROPERTY);
475     if ((*dst = strdup((char *)property)) == NULL) {
476         xmlFree(property);
477         return (Z_NOMEM);
478     }
479     xmlFree(property);
480     return (Z_OK);
481 }
483 static int
484 getrootattr(zone_dochandle_t handle, const xmlChar *propname,
485             char *propval, size_t propsize)
486 {
487     xmlNodePtr root;
488     int err;
490     if ((err = getroot(handle, &root)) != 0)
491         return (err);
493     return (fetchprop(root, propname, propval, propsize));
494 }
496 static int
497 get_alloc_rootattr(zone_dochandle_t handle, const xmlChar *propname,
498                   char **propval)
499 {
500     xmlNodePtr root;
501     int err;
503     if ((err = getroot(handle, &root)) != 0)
504         return (err);
506     return (fetch_alloc_prop(root, propname, propval));
507 }
509 static int
510 setrootattr(zone_dochandle_t handle, const xmlChar *propname,
511             const char *propval)
512 {
513     int err;
514     xmlNodePtr root;
516     if ((err = getroot(handle, &root)) != Z_OK)
517         return (err);
519     /*
520     * If we get a null propval remove the property (ignore return since it

```

```

521     * may not be set to begin with).
522     */
523     if (propval == NULL) {
524         (void) xmlUnsetProp(root, propname);
525     } else {
526         if (xmlSetProp(root, propname, (const xmlChar *) propval)
527             == NULL)
528             return (Z_INVALID);
529     }
530     return (Z_OK);
531 }

532 static void
533 addcomment(zone_dochandle_t handle, const char *comment)
534 {
535     xmlNodePtr node;
536     node = xmlNewComment((xmlChar *) comment);
537
538     if (node != NULL)
539         (void) xmlAddPrevSibling(handle->zone_dh_top, node);
540 }

541 static void
542 stripcomments(zone_dochandle_t handle)
543 {
544     xmlDocPtr top;
545     xmlNodePtr child, next;
546
547     top = handle->zone_dh_doc;
548     for (child = top->xmlChildrenNode; child != NULL; child = next) {
549         next = child->next;
550         if (child->name == NULL)
551             continue;
552         if (xmlStrcmp(child->name, DTD_ELEM_COMMENT) == 0) {
553             next = child->next;
554             xmlUnlinkNode(child);
555             xmlFreeNode(child);
556         }
557     }
558 }

559 static void
560 strip_sw_inv(zone_dochandle_t handle)
561 {
562     xmlNodePtr root, child, next;
563
564     root = xmlDocGetRootElement(handle->zone_dh_doc);
565     for (child = root->xmlChildrenNode; child != NULL; child = next) {
566         next = child->next;
567         if (child->name == NULL)
568             continue;
569         if (xmlStrcmp(child->name, DTD_ELEM_PACKAGE) == 0) {
570             next = child->next;
571             xmlUnlinkNode(child);
572             xmlFreeNode(child);
573         }
574     }
575 }

576 static int
577 zonecfg_get_handle_impl(const char *zonename, const char *filename,
578     zone_dochandle_t handle)
579 {
580     xmlValidCtxtPtr cvp;
581     struct stat statbuf;
582     int valid;

```

```

583     if (zonename == NULL)
584         return (Z_NO_ZONE);
585
586     if ((handle->zone_dh_doc = xmlParseFile(filename)) == NULL) {
587         /* distinguish file not found vs. found but not parsed */
588         if (stat(filename, &statbuf) == 0)
589             return (Z_INVALID_DOCUMENT);
590         return (Z_NO_ZONE);
591     }
592     if ((cvp = xmlNewValidCtxt()) == NULL)
593         return (Z_NOMEM);
594     cvp->error = zonecfg_error_func;
595     cvp->warning = zonecfg_error_func;
596     valid = xmlValidateDocument(cvp, handle->zone_dh_doc);
597     if (valid == 0)
598         return (Z_INVALID_DOCUMENT);
599
600     /* delete any comments such as inherited Sun copyright / ident str */
601     stripcomments(handle);
602     return (Z_OK);
603 }

604 int
605 zonecfg_get_handle(const char *zonename, zone_dochandle_t handle)
606 {
607     char path[MAXPATHLEN];
608
609     if (!config_file_path(zonename, path))
610         return (Z_MISC_FS);
611     handle->zone_dh_newzone = B_FALSE;
612
613     return (zonecfg_get_handle_impl(zonename, path, handle));
614 }

615 int
616 zonecfg_get_attach_handle(const char *path, const char *fname,
617     const char *zonename, boolean_t preserve_sw, zone_dochandle_t handle)
618 {
619     char migpath[MAXPATHLEN];
620     int err;
621     struct stat buf;
622
623     if (snprintf(migpath, sizeof(migpath), "%s/root", path) >=
624         sizeof(migpath))
625         return (Z_NOMEM);
626
627     if (stat(migpath, &buf) == -1 || !S_ISDIR(buf.st_mode))
628         return (Z_NO_ZONE);
629
630     if (snprintf(migpath, sizeof(migpath), "%s/%s", path, fname) >=
631         sizeof(migpath))
632         return (Z_NOMEM);
633
634     if ((err = zonecfg_get_handle_impl(zonename, migpath, handle)) != Z_OK)
635         return (err);
636
637     if (!preserve_sw)
638         strip_sw_inv(handle);
639
640     handle->zone_dh_newzone = B_TRUE;
641     if ((err = setrootattr(handle, DTD_ATTR_ZONEPATH, path)) != Z_OK)
642         return (err);
643
644     return (setrootattr(handle, DTD_ATTR_NAME, zonename));

```

```

653 }
655 int
656 zonecfg_get_snapshot_handle(const char *zonename, zone_dochandle_t handle)
657 {
658     char path[MAXPATHLEN];
660     if (!snap_file_path(zonename, path))
661         return (Z_MISC_FS);
662     handle->zone_dh_newzone = B_FALSE;
663     return (zonecfg_get_handle_impl(zonename, path, handle));
664 }
666 int
667 zonecfg_get_template_handle(const char *template, const char *zonename,
668     zone_dochandle_t handle)
669 {
670     char path[MAXPATHLEN];
671     int err;
673     if (!config_file_path(template, path))
674         return (Z_MISC_FS);
676     if ((err = zonecfg_get_handle_impl(template, path, handle)) != Z_OK)
677         return (err);
678     handle->zone_dh_newzone = B_TRUE;
679     return (setrootattr(handle, DTD_ATTR_NAME, zonename));
680 }
682 int
683 zonecfg_get_xml_handle(const char *path, zone_dochandle_t handle)
684 {
685     struct stat buf;
686     int err;
688     if (stat(path, &buf) == -1)
689         return (Z_MISC_FS);
691     if ((err = zonecfg_get_handle_impl("xml", path, handle)) != Z_OK)
692         return (err);
693     handle->zone_dh_newzone = B_TRUE;
694     return (Z_OK);
695 }
697 /*
698 * Initialize two handles from the manifest read on fd. The rem_handle
699 * is initialized from the input file, including the sw inventory. The
700 * local_handle is initialized with the same zone configuration but with
701 * no sw inventory.
702 */
703 int
704 zonecfg_attach_manifest(int fd, zone_dochandle_t local_handle,
705     zone_dochandle_t rem_handle)
706 {
707     xmlValidCtxtPtr cvp;
708     int valid;
710     /* load the manifest into the handle for the remote system */
711     if ((rem_handle->zone_dh_doc = xmlReadFd(fd, NULL, NULL, 0)) == NULL) {
712         return (Z_INVALID_DOCUMENT);
713     }
714     if ((cvp = xmlNewValidCtxt()) == NULL)
715         return (Z_NOMEM);
716     cvp->error = zonecfg_error_func;
717     cvp->warning = zonecfg_error_func;
718     valid = xmlValidateDocument(cvp, rem_handle->zone_dh_doc);

```

```

719     xmlFreeValidCtxt(cvp);
720     if (valid == 0)
721         return (Z_INVALID_DOCUMENT);
723     /* delete any comments such as inherited Sun copyright / ident str */
724     stripcomments(rem_handle);
726     rem_handle->zone_dh_newzone = B_TRUE;
727     rem_handle->zone_dh_sw_inv = B_TRUE;
729     /*
730     * Now use the remote system handle to generate a local system handle
731     * with an identical zones configuration but no sw inventory.
732     */
733     if ((local_handle->zone_dh_doc = xmlCopyDoc(rem_handle->zone_dh_doc,
734         1)) == NULL) {
735         return (Z_INVALID_DOCUMENT);
736     }
738     /*
739     * We need to re-run xmlValidateDocument on local_handle to properly
740     * update the in-core representation of the configuration.
741     */
742     if ((cvp = xmlNewValidCtxt()) == NULL)
743         return (Z_NOMEM);
744     cvp->error = zonecfg_error_func;
745     cvp->warning = zonecfg_error_func;
746     valid = xmlValidateDocument(cvp, local_handle->zone_dh_doc);
747     xmlFreeValidCtxt(cvp);
748     if (valid == 0)
749         return (Z_INVALID_DOCUMENT);
751     strip_sw_inv(local_handle);
753     local_handle->zone_dh_newzone = B_TRUE;
754     local_handle->zone_dh_sw_inv = B_FALSE;
756     return (Z_OK);
757 }
759 static boolean_t
760 is_renaming(zone_dochandle_t handle)
761 {
762     if (handle->zone_dh_newzone)
763         return (B_FALSE);
764     if (strlen(handle->zone_dh_delete_name) > 0)
765         return (B_TRUE);
766     return (B_FALSE);
767 }
769 static boolean_t
770 is_new(zone_dochandle_t handle)
771 {
772     return (handle->zone_dh_newzone || handle->zone_dh_snapshot);
773 }
775 static boolean_t
776 is_snapshot(zone_dochandle_t handle)
777 {
778     return (handle->zone_dh_snapshot);
779 }
781 /*
782 * It would be great to be able to use libc's ctype(3c) macros, but we
783 * can't, as they are locale sensitive, and it would break our limited thread
784 * safety if this routine had to change the app locale on the fly.

```

```

785 */
786 int
787 zonecfg_validate_zonename(const char *zone)
788 {
789     int i;
790
791     if (strcmp(zone, GLOBAL_ZONENAME) == 0)
792         return (Z_BOGUS_ZONE_NAME);
793
794     if (strlen(zone) >= ZONENAME_MAX)
795         return (Z_BOGUS_ZONE_NAME);
796
797     if (!((zone[0] >= 'a' && zone[0] <= 'z') ||
798          (zone[0] >= 'A' && zone[0] <= 'Z') ||
799          (zone[0] >= '0' && zone[0] <= '9'))))
800         return (Z_BOGUS_ZONE_NAME);
801
802     for (i = 1; zone[i] != '\0'; i++) {
803         if (!((zone[i] >= 'a' && zone[i] <= 'z') ||
804              (zone[i] >= 'A' && zone[i] <= 'Z') ||
805              (zone[i] >= '0' && zone[i] <= '9') ||
806              (zone[i] == '-') || (zone[i] == '_') || (zone[i] == '.')))
807             return (Z_BOGUS_ZONE_NAME);
808     }
809
810     return (Z_OK);
811 }
812
813 /*
814  * Changing the zone name requires us to track both the old and new
815  * name of the zone until commit time.
816  */
817 int
818 zonecfg_get_name(zone_dochandle_t handle, char *name, size_t namesize)
819 {
820     return (getrootattr(handle, DTD_ATTR_NAME, name, namesize));
821 }
822
823 static int
824 insert_admins(zone_dochandle_t handle, char *zonename)
825 {
826     int err;
827     struct zone_admintab admintab;
828
829     if ((err = zonecfg_setadminent(handle)) != Z_OK) {
830         return (err);
831     }
832     while (zonecfg_getadminent(handle, &admintab) == Z_OK) {
833         err = zonecfg_insert_userauths(handle,
834                                       admintab.zone_admin_user,
835                                       zonename);
836         if (err != Z_OK) {
837             (void) zonecfg_endadminent(handle);
838             return (err);
839         }
840     }
841     (void) zonecfg_endadminent(handle);
842     return (Z_OK);
843 }
844
845 int
846 zonecfg_set_name(zone_dochandle_t handle, char *name)
847 {
848     zone_state_t state;
849     char curname[ZONENAME_MAX], old_delname[ZONENAME_MAX];
850     int err;

```

```

851     if ((err = getrootattr(handle, DTD_ATTR_NAME, curname,
852                          sizeof (curname))) != Z_OK)
853         return (err);
854
855     if (strcmp(name, curname) == 0)
856         return (Z_OK);
857
858     /*
859      * Switching zone names to one beginning with SUNW is not permitted.
860      */
861     if (strncmp(name, "SUNW", 4) == 0)
862         return (Z_BOGUS_ZONE_NAME);
863
864     if ((err = zonecfg_validate_zonename(name)) != Z_OK)
865         return (err);
866
867     /*
868      * Setting the name back to the original name (effectively a revert of
869      * the name) is fine. But if we carry on, we'll falsely identify the
870      * name as "in use," so special case here.
871      */
872     if (strcmp(name, handle->zone_dh_delete_name) == 0) {
873         err = setrootattr(handle, DTD_ATTR_NAME, name);
874         handle->zone_dh_delete_name[0] = '\0';
875         return (err);
876     }
877
878     /* Check to see if new name chosen is already in use */
879     if (zone_get_state(name, &state) != Z_NO_ZONE)
880         return (Z_NAME_IN_USE);
881
882     /*
883      * If this isn't already "new" or in a renaming transition, then
884      * we're initiating a rename here; so stash the "delete name"
885      * (i.e. the name of the zone we'll be removing) for the rename.
886      */
887     (void) strncpy(old_delname, handle->zone_dh_delete_name,
888                  sizeof (old_delname));
889     if (!is_new(handle) && !is_renaming(handle)) {
890         /*
891          * Name change is allowed only when the zone we're altering
892          * is not ready or running.
893          */
894         err = zone_get_state(curname, &state);
895         if (err == Z_OK) {
896             if (state > ZONE_STATE_INSTALLED)
897                 return (Z_BAD_ZONE_STATE);
898             } else if (err != Z_NO_ZONE) {
899                 return (err);
900             }
901
902         (void) strncpy(handle->zone_dh_delete_name, curname,
903                     sizeof (handle->zone_dh_delete_name));
904         assert(is_renaming(handle));
905     } else if (is_renaming(handle)) {
906         err = zone_get_state(handle->zone_dh_delete_name, &state);
907         if (err == Z_OK) {
908             if (state > ZONE_STATE_INSTALLED)
909                 return (Z_BAD_ZONE_STATE);
910             } else if (err != Z_NO_ZONE) {
911                 return (err);
912             }
913     }
914
915     if ((err = setrootattr(handle, DTD_ATTR_NAME, name)) != Z_OK) {
916         /*

```

```

917     * Restore the deletename to whatever it was at the
918     * top of the routine, since we've had a failure.
919     */
920     (void) strcpy(handle->zone_dh_delete_name, old_delname,
921                 sizeof (handle->zone_dh_delete_name));
922     return (err);
923 }
924
925 /*
926  * Record the old admins from the old zonename
927  * so that they can be deleted when the operation is committed.
928  */
929 if ((err = insert_admins(handle, curname)) != Z_OK)
930     return (err);
931 else
932     return (Z_OK);
933 }
934
935 int
936 zonecfg_get_zonepath(zone_dochandle_t handle, char *path, size_t pathsize)
937 {
938     size_t len;
939
940     if ((len = strcpy(path, zonecfg_root, pathsize)) >= pathsize)
941         return (Z_TOO_BIG);
942     return (getrootattr(handle, DTD_ATTR_ZONEPATH, path + len,
943                        pathsize - len));
944 }
945
946 int
947 zonecfg_set_zonepath(zone_dochandle_t handle, char *zonepath)
948 {
949     size_t len;
950     char *modpath, *copy_mp, *curr_mp;    /* modified path ptrs */
951     char last_copied;
952     int ret;
953
954     /*
955      * Collapse multiple contiguous slashes and remove trailing slash.
956      */
957     modpath = strdup(zonepath);
958     if (modpath == NULL)
959         return (Z_NOMEM);
960     last_copied = '\0';
961     for (copy_mp = curr_mp = modpath; *curr_mp != '\0'; curr_mp++) {
962         if (*curr_mp != '/' || last_copied != '/') {
963             last_copied = *copy_mp = *curr_mp;
964             copy_mp++;
965         }
966     }
967     if (last_copied == '/')
968         copy_mp--;
969     *copy_mp = '\0';
970
971     /*
972      * The user deals in absolute paths in the running global zone, but the
973      * internal configuration files deal with boot environment relative
974      * paths. Strip out the alternate root when specified.
975      */
976     len = strlen(zonecfg_root);
977     if (strncmp(modpath, zonecfg_root, len) != 0 || modpath[len] != '/') {
978         free(modpath);
979         return (Z_BAD_PROPERTY);
980     }
981     curr_mp = modpath + len;
982     ret = setrootattr(handle, DTD_ATTR_ZONEPATH, curr_mp);

```

```

983     free(modpath);
984     return (ret);
985 }
986
987 static int
988 i_zonecfg_get_brand(zone_dochandle_t handle, char *brand, size_t brandsize,
989                   boolean_t default_query)
990 {
991     int ret, sz;
992
993     ret = getrootattr(handle, DTD_ATTR_BRAND, brand, brandsize);
994
995     /*
996      * If the lookup failed, or succeeded in finding a non-null brand
997      * string then return.
998      */
999     if (ret != Z_OK || brand[0] != '\0')
1000         return (ret);
1001
1002     if (!default_query) {
1003         /* If the zone has no brand, it is the default brand. */
1004         return (zonecfg_default_brand(brand, brandsize));
1005     }
1006
1007     /* if SUNWdefault didn't specify a brand, fallback to "native" */
1008     sz = strcpy(brand, NATIVE_BRAND_NAME, brandsize);
1009     if (sz >= brandsize)
1010         return (Z_TOO_BIG);
1011     return (Z_OK);
1012 }
1013
1014 int
1015 zonecfg_get_brand(zone_dochandle_t handle, char *brand, size_t brandsize)
1016 {
1017     return (i_zonecfg_get_brand(handle, brand, brandsize, B_FALSE));
1018 }
1019
1020 int
1021 zonecfg_set_brand(zone_dochandle_t handle, char *brand)
1022 {
1023     return (setrootattr(handle, DTD_ATTR_BRAND, brand));
1024 }
1025
1026 int
1027 zonecfg_get_autoboot(zone_dochandle_t handle, boolean_t *autoboot)
1028 {
1029     char autobootstr[DTD_ENTITY_BOOL_LEN];
1030     int ret;
1031
1032     if ((ret = getrootattr(handle, DTD_ATTR_AUTOBOOT, autobootstr,
1033                          sizeof (autobootstr))) != Z_OK)
1034         return (ret);
1035
1036     if (strcmp(autobootstr, DTD_ENTITY_TRUE) == 0)
1037         *autoboot = B_TRUE;
1038     else if (strcmp(autobootstr, DTD_ENTITY_FALSE) == 0)
1039         *autoboot = B_FALSE;
1040     else
1041         ret = Z_BAD_PROPERTY;
1042     return (ret);
1043 }
1044
1045 int
1046 zonecfg_set_autoboot(zone_dochandle_t handle, boolean_t autoboot)
1047 {
1048     return (setrootattr(handle, DTD_ATTR_AUTOBOOT,

```

```

1049         autoboot ? DTD_ENTITY_TRUE : DTD_ENTITY_FALSE));
1050     }

1052 int
1053 zonecfg_get_pool(zone_dochandle_t handle, char *pool, size_t poolsize)
1054 {
1055     return (getrootattr(handle, DTD_ATTR_POOL, pool, poolsize));
1056 }

1058 int
1059 zonecfg_set_pool(zone_dochandle_t handle, char *pool)
1060 {
1061     return (setrootattr(handle, DTD_ATTR_POOL, pool));
1062 }

1064 int
1065 zonecfg_get_limitpriv(zone_dochandle_t handle, char **limitpriv)
1066 {
1067     return (get_alloc_rootattr(handle, DTD_ATTR_LIMITPRIV, limitpriv));
1068 }

1070 int
1071 zonecfg_set_limitpriv(zone_dochandle_t handle, char *limitpriv)
1072 {
1073     return (setrootattr(handle, DTD_ATTR_LIMITPRIV, limitpriv));
1074 }

1076 int
1077 zonecfg_get_bootargs(zone_dochandle_t handle, char *bargs, size_t bargssize)
1078 {
1079     return (getrootattr(handle, DTD_ATTR_BOOTARGS, bargs, bargssize));
1080 }

1082 int
1083 zonecfg_set_bootargs(zone_dochandle_t handle, char *bargs)
1084 {
1085     return (setrootattr(handle, DTD_ATTR_BOOTARGS, bargs));
1086 }

1088 int
1089 zonecfg_get_sched_class(zone_dochandle_t handle, char *sched, size_t schedsize)
1090 {
1091     return (getrootattr(handle, DTD_ATTR_SCHED, sched, schedsize));
1092 }

1094 int
1095 zonecfg_set_sched(zone_dochandle_t handle, char *sched)
1096 {
1097     return (setrootattr(handle, DTD_ATTR_SCHED, sched));
1098 }

1100 /*
1101  * /etc/zones/index caches a vital piece of information which is also
1102  * in the <zonename>.xml file: the path to the zone. This is for performance,
1103  * since we need to walk all zonepath's in order to be able to detect conflicts
1104  * (see crosscheck_zonepaths() in the zoneadm command).
1105  *
1106  * An additional complexity is that when doing a rename, we'd like the entire
1107  * index update operation (rename, and potential state changes) to be atomic.
1108  * In general, the operation of this function should succeed or fail as
1109  * a unit.
1110  */
1111 int
1112 zonecfg_refresh_index_file(zone_dochandle_t handle)
1113 {
1114     char name[ZONENAME_MAX], zonepath[MAXPATHLEN];

```

```

1115     struct zoneent ze;
1116     int err;
1117     int opcode;
1118     char *zn;

1120     bzero(&ze, sizeof (ze));
1121     ze.zone_state = -1; /* Preserve existing state in index */

1123     if ((err = zonecfg_get_name(handle, name, sizeof (name))) != Z_OK)
1124         return (err);
1125     (void) strncpy(ze.zone_name, name, sizeof (ze.zone_name));

1127     if ((err = zonecfg_get_zonepath(handle, zonepath,
1128     sizeof (zonepath))) != Z_OK)
1129         return (err);
1130     (void) strncpy(ze.zone_path, zonepath + strlen(zonecfg_root),
1131     sizeof (ze.zone_path));

1133     if (is_renaming(handle)) {
1134         opcode = PZE_MODIFY;
1135         (void) strncpy(ze.zone_name, handle->zone_dh_delete_name,
1136         sizeof (ze.zone_name));
1137         (void) strncpy(ze.zone_newname, name, sizeof (ze.zone_newname));
1138     } else if (is_new(handle)) {
1139         FILE *cookie;
1140         /*
1141          * Be tolerant of the zone already existing in the index file,
1142          * since we might be forcibly overwriting an existing
1143          * configuration with a new one (for example 'create -F'
1144          * in zonecfg).
1145          */
1146         opcode = PZE_ADD;
1147         cookie = setzoneent();
1148         while ((zn = getzoneent(cookie)) != NULL) {
1149             if (strcmp(zn, name) == 0) {
1150                 opcode = PZE_MODIFY;
1151                 free(zn);
1152                 break;
1153             }
1154             free(zn);
1155         }
1156         endzoneent(cookie);
1157         ze.zone_state = ZONE_STATE_CONFIGURED;
1158     } else {
1159         opcode = PZE_MODIFY;
1160     }

1162     if ((err = putzoneent(&ze, opcode)) != Z_OK)
1163         return (err);

1165     return (Z_OK);
1166 }

1168 /*
1169  * The goal of this routine is to cause the index file update and the
1170  * document save to happen as an atomic operation. We do the document
1171  * first, saving a backup copy using a hard link; if that succeeds, we go
1172  * on to the index. If that fails, we roll the document back into place.
1173  *
1174  * Strategy:
1175  *
1176  * New zone 'foo' configuration:
1177  * Create tmpfile (zonecfg.xxxxxx)
1178  * Write XML to tmpfile
1179  * Rename tmpfile to xmlfile (zonecfg.xxxxxx -> foo.xml)
1180  * Add entry to index file

```



```

1181 *      If it fails, delete foo.xml, leaving nothing behind.
1182 *
1183 * Save existing zone 'foo':
1184 *   Make backup of foo.xml -> .backup
1185 *   Create tmpfile (zonecfg.xxxxxx)
1186 *   Write XML to tmpfile
1187 *   Rename tmpfile to xmlfile (zonecfg.xxxxxx -> foo.xml)
1188 *   Modify index file as needed
1189 *   If it fails, recover from .backup -> foo.xml
1190 *
1191 * Rename 'foo' to 'bar':
1192 *   Create tmpfile (zonecfg.xxxxxx)
1193 *   Write XML to tmpfile
1194 *   Rename tmpfile to xmlfile (zonecfg.xxxxxx -> bar.xml)
1195 *   Add entry for 'bar' to index file, Remove entry for 'foo' (refresh)
1196 *   If it fails, delete bar.xml; foo.xml is left behind.
1197 */
1198 static int
1199 zonecfg_save_impl(zone_dochandle_t handle, char *filename)
1200 {
1201     char tmpfile[MAXPATHLEN];
1202     char bakdir[MAXPATHLEN], bakbase[MAXPATHLEN], bakfile[MAXPATHLEN];
1203     int tmpfd, err, valid;
1204     xmlValidCtxt cvp = { NULL };
1205     boolean_t backup;
1206
1207     (void) strcpy(tmpfile, filename, sizeof (tmpfile));
1208     (void) dirname(tmpfile);
1209     (void) strcat(tmpfile, _PATH_TMPFILE, sizeof (tmpfile));
1210
1211     tmpfd = mkstemp(tmpfile);
1212     if (tmpfd == -1) {
1213         (void) unlink(tmpfile);
1214         return (Z_TEMP_FILE);
1215     }
1216     (void) close(tmpfd);
1217
1218     cvp.error = zonecfg_error_func;
1219     cvp.warning = zonecfg_error_func;
1220
1221     /*
1222     * We do a final validation of the document. Since the library has
1223     * malfunctioned if it fails to validate, we follow-up with an
1224     * assert() that the doc is valid.
1225     */
1226     valid = xmlValidateDocument(&cvp, handle->zone_dh_doc);
1227     assert(valid != 0);
1228
1229     if (xmlSaveFormatFile(tmpfile, handle->zone_dh_doc, 1) <= 0)
1230         goto err;
1231
1232     (void) chmod(tmpfile, 0644);
1233
1234     /*
1235     * In the event we are doing a standard save, hard link a copy of the
1236     * original file in .backup.<pid>.filename so we can restore it if
1237     * something goes wrong.
1238     */
1239     if (!is_new(handle) && !is_renaming(handle)) {
1240         backup = B_TRUE;
1241
1242         (void) strcpy(bakdir, filename, sizeof (bakdir));
1243         (void) strcpy(bakbase, filename, sizeof (bakbase));
1244         (void) sprintf(bakfile, sizeof (bakfile), "%s/.backup.%d.%s",
1245             dirname(bakdir), getpid(), basename(bakbase));

```

```

1247         if (link(filename, bakfile) == -1) {
1248             err = errno;
1249             (void) unlink(tmpfile);
1250             if (errno == EACCES)
1251                 return (Z_ACCES);
1252             return (Z_MISC_FS);
1253         }
1254     }
1255
1256     /*
1257     * Move the new document over top of the old.
1258     * i.e.: zonecfg.XXXXXX -> myzone.xml
1259     */
1260     if (rename(tmpfile, filename) == -1) {
1261         err = errno;
1262         (void) unlink(tmpfile);
1263         if (backup)
1264             (void) unlink(bakfile);
1265         if (err == EACCES)
1266             return (Z_ACCES);
1267         return (Z_MISC_FS);
1268     }
1269
1270     /*
1271     * If this is a snapshot, we're done-- don't add an index entry.
1272     */
1273     if (is_snapshot(handle))
1274         return (Z_OK);
1275
1276     /* now update the index file to reflect whatever we just did */
1277     if ((err = zonecfg_refresh_index_file(handle)) != Z_OK) {
1278         if (backup) {
1279             /*
1280             * Try to restore from our backup.
1281             */
1282             (void) unlink(filename);
1283             (void) rename(bakfile, filename);
1284         } else {
1285             /*
1286             * Either the zone is new, in which case we can delete
1287             * new.xml, or we're doing a rename, so ditto.
1288             */
1289             assert(is_new(handle) || is_renaming(handle));
1290             (void) unlink(filename);
1291         }
1292         return (Z_UPDATING_INDEX);
1293     }
1294
1295     if (backup)
1296         (void) unlink(bakfile);
1297
1298     return (Z_OK);
1299
1300 err:
1301     (void) unlink(tmpfile);
1302     return (Z_SAVING_FILE);
1303 }
1304
1305 int
1306 zonecfg_save(zone_dochandle_t handle)
1307 {
1308     char zname[ZONENAME_MAX], path[MAXPATHLEN];
1309     char delpath[MAXPATHLEN];
1310     int err = Z_SAVING_FILE;
1311
1312     if (zonecfg_check_handle(handle) != Z_OK)

```

```

1313         return (Z_BAD_HANDLE);
1314
1315     /*
1316     * We don't support saving snapshots or a tree containing a w
1317     * inventory at this time.
1318     */
1319     if (handle->zone_dh_snapshot || handle->zone_dh_sw_inv)
1320         return (Z_INVALID);
1321
1322     if ((err = zonecfg_get_name(handle, zname, sizeof (zname))) != Z_OK)
1323         return (err);
1324
1325     if (!config_file_path(zname, path))
1326         return (Z_MISC_FS);
1327
1328     addcomment(handle, "\n    DO NOT EDIT THIS "
1329                "FILE. Use zonecfg(1M) instead.\n");
1330
1331     /*
1332     * Update user_attr first so that it will be older
1333     * than the config file.
1334     */
1335     (void) zonecfg_authorize_users(handle, zname);
1336     err = zonecfg_save_impl(handle, path);
1337
1338     stripcomments(handle);
1339
1340     if (err != Z_OK)
1341         return (err);
1342
1343     handle->zone_dh_newzone = B_FALSE;
1344
1345     if (is_renaming(handle)) {
1346         if (config_file_path(handle->zone_dh_delete_name, delpath))
1347             (void) unlink(delpath);
1348         handle->zone_dh_delete_name[0] = '\0';
1349     }
1350
1351     return (Z_OK);
1352 }
1353
1354 int
1355 zonecfg_verify_save(zone_dochandle_t handle, char *filename)
1356 {
1357     int valid;
1358
1359     xmlValidCtxt cvp = { NULL };
1360
1361     if (zonecfg_check_handle(handle) != Z_OK)
1362         return (Z_BAD_HANDLE);
1363
1364     cvp.error = zonecfg_error_func;
1365     cvp.warning = zonecfg_error_func;
1366
1367     /*
1368     * We do a final validation of the document. Since the library has
1369     * malfunctioned if it fails to validate, we follow-up with an
1370     * assert() that the doc is valid.
1371     */
1372     valid = xmlValidateDocument(&cvp, handle->zone_dh_doc);
1373     assert(valid != 0);
1374
1375     if (xmlSaveFormatFile(filename, handle->zone_dh_doc, 1) <= 0)
1376         return (Z_SAVING_FILE);
1377
1378     return (Z_OK);

```

```

1379 }
1380
1381 int
1382 zonecfg_detach_save(zone_dochandle_t handle, uint_t flags)
1383 {
1384     char zname[ZONENAME_MAX];
1385     char path[MAXPATHLEN];
1386     char migpath[MAXPATHLEN];
1387     xmlValidCtxt cvp = { NULL };
1388     int err = Z_SAVING_FILE;
1389     int valid;
1390
1391     if (zonecfg_check_handle(handle) != Z_OK)
1392         return (Z_BAD_HANDLE);
1393
1394     if (flags & ZONE_DRY_RUN) {
1395         (void) strcpy(migpath, "-", sizeof (migpath));
1396     } else {
1397         if ((err = zonecfg_get_name(handle, zname, sizeof (zname)))
1398             != Z_OK)
1399             return (err);
1400
1401         if ((err = zone_get_zonepath(zname, path, sizeof (path)))
1402             != Z_OK)
1403             return (err);
1404
1405         if (snprintf(migpath, sizeof (migpath), "%s/%s", path,
1406                    ZONE_DETACHED) >= sizeof (migpath))
1407             return (Z_NOMEM);
1408     }
1409
1410     if ((err = operation_prep(handle)) != Z_OK)
1411         return (err);
1412
1413     addcomment(handle, "\n    DO NOT EDIT THIS FILE. "
1414                "Use zonecfg(1M) and zoneadm(1M) attach.\n");
1415
1416     cvp.error = zonecfg_error_func;
1417     cvp.warning = zonecfg_error_func;
1418
1419     /*
1420     * We do a final validation of the document. Since the library has
1421     * malfunctioned if it fails to validate, we follow-up with an
1422     * assert() that the doc is valid.
1423     */
1424     valid = xmlValidateDocument(&cvp, handle->zone_dh_doc);
1425     assert(valid != 0);
1426
1427     if (xmlSaveFormatFile(migpath, handle->zone_dh_doc, 1) <= 0)
1428         return (Z_SAVING_FILE);
1429
1430     if (!(flags & ZONE_DRY_RUN))
1431         (void) chmod(migpath, 0644);
1432
1433     stripcomments(handle);
1434
1435     handle->zone_dh_newzone = B_FALSE;
1436
1437     return (Z_OK);
1438 }
1439
1440 boolean_t
1441 zonecfg_detached(const char *path)
1442 {
1443     char          migpath[MAXPATHLEN];
1444     struct stat   buf;

```

```

1446     if (snprintf(migpath, sizeof (migpath), "%s/%s", path, ZONE_DETACHED) >=
1447         sizeof (migpath))
1448         return (B_FALSE);
1450     if (stat(migpath, &buf) != -1)
1451         return (B_TRUE);
1453     return (B_FALSE);
1454 }
1456 void
1457 zonecfg_rm_detached(zone_dochandle_t handle, boolean_t forced)
1458 {
1459     char zname[ZONENAME_MAX];
1460     char path[MAXPATHLEN];
1461     char detached[MAXPATHLEN];
1462     char attached[MAXPATHLEN];
1464     if (zonecfg_check_handle(handle) != Z_OK)
1465         return;
1467     if (zonecfg_get_name(handle, zname, sizeof (zname)) != Z_OK)
1468         return;
1470     if (zone_get_zonepath(zname, path, sizeof (path)) != Z_OK)
1471         return;
1473     (void) snprintf(detached, sizeof (detached), "%s/%s", path,
1474         ZONE_DETACHED);
1475     (void) snprintf(attached, sizeof (attached), "%s/%s", path,
1476         ATTACH_FORCED);
1478     if (forced) {
1479         (void) rename(detached, attached);
1480     } else {
1481         (void) unlink(attached);
1482         (void) unlink(detached);
1483     }
1484 }
1486 /*
1487  * Special case: if access(2) fails with ENOENT, then try again using
1488  * ZONE_CONFIG_ROOT instead of config_file_path(zonename). This is how we
1489  * work around the case of a config file which has not been created yet:
1490  * the user will need access to the directory so use that as a heuristic.
1491  */
1493 int
1494 zonecfg_access(const char *zonename, int amode)
1495 {
1496     char path[MAXPATHLEN];
1498     if (!config_file_path(zonename, path))
1499         return (Z_INVALID);
1500     if (access(path, amode) == 0)
1501         return (Z_OK);
1502     if (errno == ENOENT) {
1503         if (snprintf(path, sizeof (path), "%s%s", zonecfg_root,
1504             ZONE_CONFIG_ROOT) >= sizeof (path))
1505             return (Z_INVALID);
1506         if (access(path, amode) == 0)
1507             return (Z_OK);
1508     }
1509     if (errno == EACCES)
1510         return (Z_ACCES);

```

```

1511     if (errno == EINVAL)
1512         return (Z_INVALID);
1513     return (Z_MISC_FS);
1514 }
1516 int
1517 zonecfg_create_snapshot(const char *zonename)
1518 {
1519     zone_dochandle_t handle;
1520     char path[MAXPATHLEN], zonepath[MAXPATHLEN], rpath[MAXPATHLEN];
1521     int error = Z_OK, res;
1523     if ((handle = zonecfg_init_handle()) == NULL) {
1524         return (Z_NOMEM);
1525     }
1527     handle->zone_dh_newzone = B_TRUE;
1528     handle->zone_dh_snapshot = B_TRUE;
1530     if ((error = zonecfg_get_handle(zonename, handle)) != Z_OK)
1531         goto out;
1532     if ((error = operation_prep(handle)) != Z_OK)
1533         goto out;
1534     error = zonecfg_get_zonepath(handle, zonepath, sizeof (zonepath));
1535     if (error != Z_OK)
1536         goto out;
1537     if ((res = resolvepath(zonepath, rpath, sizeof (rpath))) == -1) {
1538         error = Z_RESOLVED_PATH;
1539         goto out;
1540     }
1541     /*
1542      * If the resolved path is not the same as the original path, then
1543      * save the resolved path in the snapshot, thus preventing any
1544      * potential problems down the line when zoneadmd goes to unmount
1545      * file systems and depends on initial string matches with resolved
1546      * paths.
1547      */
1548     rpath[res] = '\0';
1549     if (strcmp(zonepath, rpath) != 0) {
1550         if ((error = zonecfg_set_zonepath(handle, rpath)) != Z_OK)
1551             goto out;
1552     }
1553     if (snprintf(path, sizeof (path), "%s%s", zonecfg_root,
1554         ZONE_SNAPSHOT_ROOT) >= sizeof (path)) {
1555         error = Z_MISC_FS;
1556         goto out;
1557     }
1558     if ((mkdir(path, S_IRWXU) == -1) && (errno != EEXIST)) {
1559         error = Z_MISC_FS;
1560         goto out;
1561     }
1563     if (!snap_file_path(zonename, path)) {
1564         error = Z_MISC_FS;
1565         goto out;
1566     }
1568     addcomment(handle, "\n DO NOT EDIT THIS FILE. "
1569         "It is a snapshot of running zone state.\n");
1571     error = zonecfg_save_impl(handle, path);
1573     stripcomments(handle);
1575 out:
1576     zonecfg_fini_handle(handle);

```

```

1577     return (error);
1578 }

1580 int
1581 zonecfg_get_iptype(zone_dochandle_t handle, zone_iptype_t *iptypep)
1582 {
1583     char property[10]; /* 10 is big enough for "shared"/"exclusive" */
1584     int err;

1586     err = getrootattr(handle, DTD_ATTR_IPTYPE, property, sizeof (property));
1587     if (err == Z_BAD_PROPERTY) {
1588         /* Return default value */
1589         *iptypep = ZS_SHARED;
1590         return (Z_OK);
1591     } else if (err != Z_OK) {
1592         return (err);
1593     }

1595     if (strlen(property) == 0 ||
1596         strcmp(property, "shared") == 0)
1597         *iptypep = ZS_SHARED;
1598     else if (strcmp(property, "exclusive") == 0)
1599         *iptypep = ZS_EXCLUSIVE;
1600     else
1601         return (Z_INVALID);

1603     return (Z_OK);
1604 }

1606 int
1607 zonecfg_set_iptype(zone_dochandle_t handle, zone_iptype_t iptype)
1608 {
1609     xmlNodePtr cur;

1611     if (handle == NULL)
1612         return (Z_INVALID);

1614     cur = xmlDocGetRootElement(handle->zone_dh_doc);
1615     if (cur == NULL) {
1616         return (Z_EMPTY_DOCUMENT);
1617     }

1619     if (xmlStrcmp(cur->name, DTD_ELEM_ZONE) != 0) {
1620         return (Z_WRONG_DOC_TYPE);
1621     }
1622     switch (iptype) {
1623     case ZS_SHARED:
1624         /*
1625          * Since "shared" is the default, we don't write it to the
1626          * configuration file, so that it's easier to migrate those
1627          * zones elsewhere, eg., to systems which are not IP-Instances
1628          * aware.
1629          * xmlUnsetProp only fails when the attribute doesn't exist,
1630          * which we don't care.
1631          */
1632         (void) xmlUnsetProp(cur, DTD_ATTR_IPTYPE);
1633         break;
1634     case ZS_EXCLUSIVE:
1635         if (xmlSetProp(cur, DTD_ATTR_IPTYPE,
1636             (const xmlChar *) "exclusive") == NULL)
1637             return (Z_INVALID);
1638         break;
1639     }
1640     return (Z_OK);
1641 }

```

```

1643 static int
1644 newprop(xmlNodePtr node, const xmlChar *attrname, char *src)
1645 {
1646     xmlAttrPtr newattr;

1648     newattr = xmlNewProp(node, attrname, (xmlChar *)src);
1649     if (newattr == NULL) {
1650         xmlUnlinkNode(node);
1651         xmlFreeNode(node);
1652         return (Z_BAD_PROPERTY);
1653     }
1654     return (Z_OK);
1655 }

1657 static int
1658 zonecfg_add_filesystem_core(zone_dochandle_t handle, struct zone_fstab *tabptr)
1659 {
1660     xmlNodePtr newnode, cur = handle->zone_dh_cur, options_node;
1661     zone_fsopt_t *ptr;
1662     int err;

1664     newnode = xmlNewTextChild(cur, NULL, DTD_ELEM_FS, NULL);
1665     if ((err = newprop(newnode, DTD_ATTR_SPECIAL,
1666         tabptr->zone_fs_special)) != Z_OK)
1667         return (err);
1668     if (tabptr->zone_fs_raw[0] != '\0' &&
1669         (err = newprop(newnode, DTD_ATTR_RAW, tabptr->zone_fs_raw)) != Z_OK)
1670         return (err);
1671     if ((err = newprop(newnode, DTD_ATTR_DIR, tabptr->zone_fs_dir)) != Z_OK)
1672         return (err);
1673     if ((err = newprop(newnode, DTD_ATTR_TYPE,
1674         tabptr->zone_fs_type)) != Z_OK)
1675         return (err);
1676     if (tabptr->zone_fs_options != NULL) {
1677         for (ptr = tabptr->zone_fs_options; ptr != NULL;
1678             ptr = ptr->zone_fsopt_next) {
1679             options_node = xmlNewTextChild(newnode, NULL,
1680                 DTD_ELEM_FSOPTION, NULL);
1681             if ((err = newprop(options_node, DTD_ATTR_NAME,
1682                 ptr->zone_fsopt_opt)) != Z_OK)
1683                 return (err);
1684         }
1685     }
1686     return (Z_OK);
1687 }

1689 int
1690 zonecfg_add_filesystem(zone_dochandle_t handle, struct zone_fstab *tabptr)
1691 {
1692     int err;

1694     if (tabptr == NULL)
1695         return (Z_INVALID);

1697     if ((err = operation_prep(handle)) != Z_OK)
1698         return (err);

1700     if ((err = zonecfg_add_filesystem_core(handle, tabptr)) != Z_OK)
1701         return (err);

1703     return (Z_OK);
1704 }

1706 int
1707 zonecfg_add_fs_option(struct zone_fstab *tabptr, char *option)
1708 {

```

```

1709     zone_fsopt_t *last, *old, *new;

1711     last = tabptr->zone_fs_options;
1712     for (old = last; old != NULL; old = old->zone_fsopt_next)
1713         last = old; /* walk to the end of the list */
1714     new = (zone_fsopt_t *)malloc(sizeof (zone_fsopt_t));
1715     if (new == NULL)
1716         return (Z_NOMEM);
1717     (void) strncpy(new->zone_fsopt_opt, option,
1718                 sizeof (new->zone_fsopt_opt));
1719     new->zone_fsopt_next = NULL;
1720     if (last == NULL)
1721         tabptr->zone_fs_options = new;
1722     else
1723         last->zone_fsopt_next = new;
1724     return (Z_OK);
1725 }

1727 int
1728 zonecfg_remove_fs_option(struct zone_fstab *tabptr, char *option)
1729 {
1730     zone_fsopt_t *last, *this, *next;

1732     last = tabptr->zone_fs_options;
1733     for (this = last; this != NULL; this = this->zone_fsopt_next) {
1734         if (strcmp(this->zone_fsopt_opt, option) == 0) {
1735             next = this->zone_fsopt_next;
1736             if (this == tabptr->zone_fs_options)
1737                 tabptr->zone_fs_options = next;
1738             else
1739                 last->zone_fsopt_next = next;
1740             free(this);
1741             return (Z_OK);
1742         } else
1743             last = this;
1744     }
1745     return (Z_NO_PROPERTY_ID);
1746 }

1748 void
1749 zonecfg_free_fs_option_list(zone_fsopt_t *list)
1750 {
1751     zone_fsopt_t *this, *next;

1753     for (this = list; this != NULL; this = next) {
1754         next = this->zone_fsopt_next;
1755         free(this);
1756     }
1757 }

1759 void
1760 zonecfg_free_rctl_value_list(struct zone_rctlvaltab *valtab)
1761 {
1762     if (valtab == NULL)
1763         return;
1764     zonecfg_free_rctl_value_list(valtab->zone_rctlval_next);
1765     free(valtab);
1766 }

1768 static boolean_t
1769 match_prop(xmlNodePtr cur, const xmlChar *attr, char *user_prop)
1770 {
1771     xmlChar *gotten_prop;
1772     int prop_result;

1774     gotten_prop = xmlGetProp(cur, attr);

```

```

1775     if (gotten_prop == NULL) /* shouldn't happen */
1776         return (B_FALSE);
1777     prop_result = xmlStrcmp(gotten_prop, (const xmlChar *) user_prop);
1778     xmlFree(gotten_prop);
1779     return ((prop_result == 0)); /* empty strings will match */
1780 }

1782 static int
1783 zonecfg_delete_filesystem_core(zone_dochandle_t handle,
1784                               struct zone_fstab *tabptr)
1785 {
1786     xmlNodePtr cur = handle->zone_dh_cur;
1787     boolean_t dir_match, spec_match, raw_match, type_match;

1789     for (cur = cur->xmlChildrenNode; cur != NULL; cur = cur->next) {
1790         if (xmlStrcmp(cur->name, DTD_ELEM_FS))
1791             continue;
1792         dir_match = match_prop(cur, DTD_ATTR_DIR, tabptr->zone_fs_dir);
1793         spec_match = match_prop(cur, DTD_ATTR_SPECIAL,
1794                                 tabptr->zone_fs_special);
1795         raw_match = match_prop(cur, DTD_ATTR_RAW,
1796                               tabptr->zone_fs_raw);
1797         type_match = match_prop(cur, DTD_ATTR_TYPE,
1798                                tabptr->zone_fs_type);
1799         if (dir_match && spec_match && raw_match && type_match) {
1800             xmlUnlinkNode(cur);
1801             xmlFreeNode(cur);
1802             return (Z_OK);
1803         }
1804     }
1805     return (Z_NO_RESOURCE_ID);
1806 }

1808 int
1809 zonecfg_delete_filesystem(zone_dochandle_t handle, struct zone_fstab *tabptr)
1810 {
1811     int err;

1813     if (tabptr == NULL)
1814         return (Z_INVALID);

1816     if ((err = operation_prep(handle)) != Z_OK)
1817         return (err);

1819     if ((err = zonecfg_delete_filesystem_core(handle, tabptr)) != Z_OK)
1820         return (err);

1822     return (Z_OK);
1823 }

1825 int
1826 zonecfg_modify_filesystem(
1827     zone_dochandle_t handle,
1828     struct zone_fstab *oldtabptr,
1829     struct zone_fstab *newtabptr)
1830 {
1831     int err;

1833     if (oldtabptr == NULL || newtabptr == NULL)
1834         return (Z_INVALID);

1836     if ((err = operation_prep(handle)) != Z_OK)
1837         return (err);

1839     if ((err = zonecfg_delete_filesystem_core(handle, oldtabptr)) != Z_OK)
1840         return (err);

```

```

1842     if ((err = zonecfg_add_filesystem_core(handle, newtabptr)) != Z_OK)
1843         return (err);
1844
1845     return (Z_OK);
1846 }
1847
1848 int
1849 zonecfg_lookup_filesystem(
1850     zone_dochandle_t handle,
1851     struct zone_fstab *tabptr)
1852 {
1853     xmlNodePtr cur, options, firstmatch;
1854     int err;
1855     char dirname[MAXPATHLEN], special[MAXPATHLEN], raw[MAXPATHLEN];
1856     char type[FSTYPSTR];
1857     char options_str[MAX_MNTOPT_STR];
1858
1859     if (tabptr == NULL)
1860         return (Z_INVALID);
1861
1862     if ((err = operation_prep(handle)) != Z_OK)
1863         return (err);
1864
1865     /*
1866      * Walk the list of children looking for matches on any properties
1867      * specified in the fstab parameter.  If more than one resource
1868      * matches, we return Z_INSUFFICIENT_SPEC; if none match, we return
1869      * Z_NO_RESOURCE_ID.
1870      */
1871     cur = handle->zone_dh_cur;
1872     firstmatch = NULL;
1873     for (cur = cur->xmlChildrenNode; cur != NULL; cur = cur->next) {
1874         if (xmlStrcmp(cur->name, DTD_ELEM_FS))
1875             continue;
1876         if (strlen(tabptr->zone_fs_dir) > 0) {
1877             if ((fetchprop(cur, DTD_ATTR_DIR, dirname,
1878                 sizeof(dirname)) == Z_OK) &&
1879                 (strcmp(tabptr->zone_fs_dir, dirname) == 0)) {
1880                 if (firstmatch == NULL)
1881                     firstmatch = cur;
1882                 else
1883                     return (Z_INSUFFICIENT_SPEC);
1884             }
1885         }
1886         if (strlen(tabptr->zone_fs_special) > 0) {
1887             if ((fetchprop(cur, DTD_ATTR_SPECIAL, special,
1888                 sizeof(special)) == Z_OK) {
1889                 if (strcmp(tabptr->zone_fs_special,
1890                     special) == 0) {
1891                     if (firstmatch == NULL)
1892                         firstmatch = cur;
1893                     else if (firstmatch != cur)
1894                         return (Z_INSUFFICIENT_SPEC);
1895                 } else {
1896                     /*
1897                      * If another property matched but this
1898                      * one doesn't then reset firstmatch.
1899                      */
1900                     if (firstmatch == cur)
1901                         firstmatch = NULL;
1902                 }
1903             }
1904         }
1905         if (strlen(tabptr->zone_fs_raw) > 0) {
1906             if ((fetchprop(cur, DTD_ATTR_RAW, raw,

```

```

1907                 sizeof(raw)) == Z_OK) {
1908                     if (strcmp(tabptr->zone_fs_raw, raw) == 0) {
1909                         if (firstmatch == NULL)
1910                             firstmatch = cur;
1911                         else if (firstmatch != cur)
1912                             return (Z_INSUFFICIENT_SPEC);
1913                     } else {
1914                         /*
1915                          * If another property matched but this
1916                          * one doesn't then reset firstmatch.
1917                          */
1918                         if (firstmatch == cur)
1919                             firstmatch = NULL;
1920                     }
1921                 }
1922             }
1923         if (strlen(tabptr->zone_fs_type) > 0) {
1924             if ((fetchprop(cur, DTD_ATTR_TYPE, type,
1925                 sizeof(type)) == Z_OK) {
1926                 if (strcmp(tabptr->zone_fs_type, type) == 0) {
1927                     if (firstmatch == NULL)
1928                         firstmatch = cur;
1929                     else if (firstmatch != cur)
1930                         return (Z_INSUFFICIENT_SPEC);
1931                 } else {
1932                     /*
1933                      * If another property matched but this
1934                      * one doesn't then reset firstmatch.
1935                      */
1936                     if (firstmatch == cur)
1937                         firstmatch = NULL;
1938                 }
1939             }
1940         }
1941     }
1942
1943     if (firstmatch == NULL)
1944         return (Z_NO_RESOURCE_ID);
1945
1946     cur = firstmatch;
1947
1948     if ((err = fetchprop(cur, DTD_ATTR_DIR, tabptr->zone_fs_dir,
1949         sizeof(tabptr->zone_fs_dir))) != Z_OK)
1950         return (err);
1951
1952     if ((err = fetchprop(cur, DTD_ATTR_SPECIAL, tabptr->zone_fs_special,
1953         sizeof(tabptr->zone_fs_special))) != Z_OK)
1954         return (err);
1955
1956     if ((err = fetchprop(cur, DTD_ATTR_RAW, tabptr->zone_fs_raw,
1957         sizeof(tabptr->zone_fs_raw))) != Z_OK)
1958         return (err);
1959
1960     if ((err = fetchprop(cur, DTD_ATTR_TYPE, tabptr->zone_fs_type,
1961         sizeof(tabptr->zone_fs_type))) != Z_OK)
1962         return (err);
1963
1964     /* options are optional */
1965     tabptr->zone_fs_options = NULL;
1966     for (options = cur->xmlChildrenNode; options != NULL;
1967         options = options->next) {
1968         if ((fetchprop(options, DTD_ATTR_NAME, options_str,
1969             sizeof(options_str)) != Z_OK)
1970             break;
1971         if (zonecfg_add_fs_option(tabptr, options_str) != Z_OK)
1972             break;

```

```

1973     }
1974     return (Z_OK);
1975 }

1977 /*
1978  * Compare two IP addresses in string form. Allow for the possibility that
1979  * one might have "<prefix-length>" at the end: allow a match on just the
1980  * IP address (or host name) part.
1981  */

1983 boolean_t
1984 zonecfg_same_net_address(char *a1, char *a2)
1985 {
1986     char *slashp, *slashp1, *slashp2;
1987     int result;

1989     if (strcmp(a1, a2) == 0)
1990         return (B_TRUE);

1992     /*
1993      * If neither has a slash or both do, they need to match to be
1994      * considered the same, but they did not match above, so fail.
1995      */
1996     slashp1 = strchr(a1, '/');
1997     slashp2 = strchr(a2, '/');
1998     if ((slashp1 == NULL && slashp2 == NULL) ||
1999         (slashp1 != NULL && slashp2 != NULL))
2000         return (B_FALSE);

2002     /*
2003      * Only one had a slash: pick that one, zero out the slash, compare
2004      * the "address only" strings, restore the slash, and return the
2005      * result of the comparison.
2006      */
2007     slashp = (slashp1 == NULL) ? slashp2 : slashp1;
2008     *slashp = '\0';
2009     result = strcmp(a1, a2);
2010     *slashp = '/';
2011     return ((result == 0));
2012 }

2014 int
2015 zonecfg_valid_net_address(char *address, struct lifreq *lifr)
2016 {
2017     struct sockaddr_in *sin4;
2018     struct sockaddr_in6 *sin6;
2019     struct addrinfo hints, *result;
2020     char *slashp = strchr(address, '/');

2022     bzero(lifr, sizeof (struct lifreq));
2023     sin4 = (struct sockaddr_in *)&lifr->lifr_addr;
2024     sin6 = (struct sockaddr_in6 *)&lifr->lifr_addr;
2025     if (slashp != NULL)
2026         *slashp = '\0';
2027     if (inet_pton(AF_INET, address, &sin4->sin_addr) == 1) {
2028         sin4->sin_family = AF_INET;
2029     } else if (inet_pton(AF_INET6, address, &sin6->sin6_addr) == 1) {
2030         if (slashp == NULL)
2031             return (Z_IPV6_ADDR_PREFIX_LEN);
2032         sin6->sin6_family = AF_INET6;
2033     } else {
2034         /* "address" may be a host name */
2035         (void) memset(&hints, 0, sizeof (hints));
2036         hints.ai_family = PF_INET;
2037         if (getaddrinfo(address, NULL, &hints, &result) != 0)
2038             return (Z_BOGUS_ADDRESS);

```

```

2039         sin4->sin_family = result->ai_family;

2041         (void) memcpy(&sin4->sin_addr,
2042             /* LINTED E_BAD_PTR_CAST_ALIGN */
2043             &((struct sockaddr_in *)result->ai_addr)->sin_addr,
2044             sizeof (struct in_addr));

2046         freeaddrinfo(result);
2047     }
2048     return (Z_OK);
2049 }

2051 boolean_t
2052 zonecfg_ifname_exists(sa_family_t af, char *ifname)
2053 {
2054     struct lifreq lifr;
2055     int so;
2056     int save_errno;

2058     (void) memset(&lifr, 0, sizeof (lifr));
2059     (void) strncpy(lifr.lifr_name, ifname, sizeof (lifr.lifr_name));
2060     lifr.lifr_addr.ss_family = af;
2061     if ((so = socket(af, SOCK_DGRAM, 0)) < 0) {
2062         /* Odd - can't tell if the ifname exists */
2063         return (B_FALSE);
2064     }
2065     if (ioctl(so, SIOCGLIFFLAGS, (caddr_t)&lifr) < 0) {
2066         save_errno = errno;
2067         (void) close(so);
2068         errno = save_errno;
2069         return (B_FALSE);
2070     }
2071     (void) close(so);
2072     return (B_TRUE);
2073 }

2075 /*
2076  * Determines whether there is a net resource with the physical interface, IP
2077  * address, and default router specified by 'tabptr' in the zone configuration
2078  * to which 'handle' refers. 'tabptr' must have an interface, an address, a
2079  * default router, or a combination of the three. This function returns Z_OK
2080  * if there is exactly one net resource matching the query specified by
2081  * 'tabptr'. The function returns Z_INSUFFICIENT_SPEC if there are multiple
2082  * matches or 'tabptr' does not specify a physical interface, address, or
2083  * default router. The function returns Z_NO_RESOURCE_ID if there are no matches.
2084  *
2085  * Errors might also be returned if the entry that exactly matches the
2086  * query lacks critical network resource information.
2087  *
2088  * If there is a single match, then the matching entry's physical interface, IP
2089  * address, and default router information are stored in 'tabptr'.
2090  */
2091 int
2092 zonecfg_lookup_nwif(zone_dochandle_t handle, struct zone_nwif *tabptr)
2093 {
2094     xmlNodePtr cur;
2095     xmlNodePtr firstmatch;
2096     int err;
2097     char address[INET6_ADDRSTRLEN];
2098     char physical[LIFNAMSIZ];
2099     size_t addrspec; /* nonzero if tabptr has IP addr */
2100     size_t physspec; /* nonzero if tabptr has interface */
2101     size_t defrouterspec; /* nonzero if tabptr has def. router */
2102     size_t allowed_addrspec;
2103     zone_ipctype_t ipctype;

```

```

2105     if (tabptr == NULL)
2106         return (Z_INVALID);

2108     /*
2109     * Determine the fields that will be searched. There must be at least
2110     * one.
2111     *
2112     * zone_nwif_address, zone_nwif_physical, and zone_nwif_defrouter are
2113     * arrays, so no NULL checks are necessary.
2114     */
2115     addrspec = strlen(tabptr->zone_nwif_address);
2116     physspec = strlen(tabptr->zone_nwif_physical);
2117     defrouterspec = strlen(tabptr->zone_nwif_defrouter);
2118     allowed_addrspec = strlen(tabptr->zone_nwif_allowed_address);
2119     if (addrspec != 0 && allowed_addrspec != 0)
2120         return (Z_INVALID); /* can't specify both */
2121     if (addrspec == 0 && physspec == 0 && defrouterspec == 0 &&
2122         allowed_addrspec == 0)
2123         return (Z_INSUFFICIENT_SPEC);

2125     if ((err = operation_prep(handle)) != Z_OK)
2126         return (err);

2128     if ((err = zonecfg_get_iptype(handle, &iptype)) != Z_OK)
2129         return (err);
2130     /*
2131     * Iterate over the configuration's elements and look for net elements
2132     * that match the query.
2133     */
2134     firstmatch = NULL;
2135     cur = handle->zone_dh_cur;
2136     for (cur = cur->xmlChildrenNode; cur != NULL; cur = cur->next) {
2137         /* Skip non-net elements */
2138         if (xmlStrcmp(cur->name, DTD_ELEM_NET))
2139             continue;

2141         /*
2142         * If any relevant fields don't match the query, then skip
2143         * the current net element.
2144         */
2145         if (physspec != 0 && (fetchprop(cur, DTD_ATTR_PHYSICAL,
2146             physical, sizeof(physical)) != Z_OK ||
2147             strcmp(tabptr->zone_nwif_physical, physical) != 0))
2148             continue;
2149         if (iptype == Z_SHARED && addrspec != 0 &&
2150             (fetchprop(cur, DTD_ATTR_ADDRESS, address,
2151                 sizeof(address)) != Z_OK ||
2152             !zonecfg_same_net_address(tabptr->zone_nwif_address,
2153                 address)))
2154             continue;
2155         if (iptype == Z_EXCLUSIVE && allowed_addrspec != 0 &&
2156             (fetchprop(cur, DTD_ATTR_ALLOWED_ADDRESS, address,
2157                 sizeof(address)) != Z_OK ||
2158             !zonecfg_same_net_address(tabptr->zone_nwif_allowed_address,
2159                 address)))
2160             continue;
2161         if (defrouterspec != 0 && (fetchprop(cur, DTD_ATTR_DEFROUTER,
2162             address, sizeof(address)) != Z_OK ||
2163             !zonecfg_same_net_address(tabptr->zone_nwif_defrouter,
2164                 address)))
2165             continue;

2167         /*
2168         * The current net element matches the query. Select it if
2169         * it's the first match; otherwise, abort the search.
2170         */

```

```

2171         if (firstmatch == NULL)
2172             firstmatch = cur;
2173         else
2174             return (Z_INSUFFICIENT_SPEC);
2175     }
2176     if (firstmatch == NULL)
2177         return (Z_NO_RESOURCE_ID);

2179     cur = firstmatch;

2181     if ((err = fetchprop(cur, DTD_ATTR_PHYSICAL, tabptr->zone_nwif_physical,
2182         sizeof(tabptr->zone_nwif_physical))) != Z_OK)
2183         return (err);

2185     if (iptype == Z_SHARED &&
2186         (err = fetchprop(cur, DTD_ATTR_ADDRESS, tabptr->zone_nwif_address,
2187             sizeof(tabptr->zone_nwif_address))) != Z_OK)
2188         return (err);

2190     if (iptype == Z_EXCLUSIVE &&
2191         (err = fetchprop(cur, DTD_ATTR_ALLOWED_ADDRESS,
2192             tabptr->zone_nwif_allowed_address,
2193             sizeof(tabptr->zone_nwif_allowed_address))) != Z_OK)
2194         return (err);

2196     if ((err = fetchprop(cur, DTD_ATTR_DEFROUTER,
2197         tabptr->zone_nwif_defrouter,
2198         sizeof(tabptr->zone_nwif_defrouter))) != Z_OK)
2199         return (err);

2201     return (Z_OK);
2202 }

2204 static int
2205 zonecfg_add_nwif_core(zone_dochandle_t handle, struct zone_nwif *tabptr)
2206 {
2207     xmlNodePtr newnode, cur = handle->zone_dh_cur;
2208     int err;

2210     newnode = xmlNewTextChild(cur, NULL, DTD_ELEM_NET, NULL);
2211     if (strlen(tabptr->zone_nwif_address) > 0 &&
2212         (err = newprop(newnode, DTD_ATTR_ADDRESS,
2213             tabptr->zone_nwif_address)) != Z_OK)
2214         return (err);
2215     if (strlen(tabptr->zone_nwif_allowed_address) > 0 &&
2216         (err = newprop(newnode, DTD_ATTR_ALLOWED_ADDRESS,
2217             tabptr->zone_nwif_allowed_address)) != Z_OK)
2218         return (err);
2219     if ((err = newprop(newnode, DTD_ATTR_PHYSICAL,
2220         tabptr->zone_nwif_physical)) != Z_OK)
2221         return (err);
2222     /*
2223     * Do not add this property when it is not set, for backwards
2224     * compatibility and because it is optional.
2225     */
2226     if ((strlen(tabptr->zone_nwif_defrouter) > 0) &&
2227         ((err = newprop(newnode, DTD_ATTR_DEFROUTER,
2228             tabptr->zone_nwif_defrouter)) != Z_OK))
2229         return (err);
2230     return (Z_OK);
2231 }

2233 int
2234 zonecfg_add_nwif(zone_dochandle_t handle, struct zone_nwif *tabptr)
2235 {
2236     int err;

```



```

2238     if (tabptr == NULL)
2239         return (Z_INVALID);

2241     if ((err = operation_prep(handle)) != Z_OK)
2242         return (err);

2244     if ((err = zonecfg_add_nwif_core(handle, tabptr)) != Z_OK)
2245         return (err);

2247     return (Z_OK);
2248 }

2250 static int
2251 zonecfg_delete_nwif_core(zone_dochandle_t handle, struct zone_nwifab *tabptr)
2252 {
2253     xmlNodePtr cur = handle->zone_dh_cur;
2254     boolean_t addr_match, phys_match, allowed_addr_match;

2256     for (cur = cur->xmlChildrenNode; cur != NULL; cur = cur->next) {
2257         if (xmlStrcmp(cur->name, DTD_ELEM_NET))
2258             continue;

2260         addr_match = match_prop(cur, DTD_ATTR_ADDRESS,
2261             tabptr->zone_nwif_address);
2262         allowed_addr_match = match_prop(cur, DTD_ATTR_ALLOWED_ADDRESS,
2263             tabptr->zone_nwif_allowed_address);
2264         phys_match = match_prop(cur, DTD_ATTR_PHYSICAL,
2265             tabptr->zone_nwif_physical);

2267         if (addr_match && allowed_addr_match && phys_match) {
2268             xmlUnlinkNode(cur);
2269             xmlFreeNode(cur);
2270             return (Z_OK);
2271         }
2272     }
2273     return (Z_NO_RESOURCE_ID);
2274 }

2276 int
2277 zonecfg_delete_nwif(zone_dochandle_t handle, struct zone_nwifab *tabptr)
2278 {
2279     int err;

2281     if (tabptr == NULL)
2282         return (Z_INVALID);

2284     if ((err = operation_prep(handle)) != Z_OK)
2285         return (err);

2287     if ((err = zonecfg_delete_nwif_core(handle, tabptr)) != Z_OK)
2288         return (err);

2290     return (Z_OK);
2291 }

2293 int
2294 zonecfg_modify_nwif(
2295     zone_dochandle_t handle,
2296     struct zone_nwifab *oldtabptr,
2297     struct zone_nwifab *newtabptr)
2298 {
2299     int err;

2301     if (oldtabptr == NULL || newtabptr == NULL)
2302         return (Z_INVALID);

```

```

2304     if ((err = operation_prep(handle)) != Z_OK)
2305         return (err);

2307     if ((err = zonecfg_delete_nwif_core(handle, oldtabptr)) != Z_OK)
2308         return (err);

2310     if ((err = zonecfg_add_nwif_core(handle, newtabptr)) != Z_OK)
2311         return (err);

2313     return (Z_OK);
2314 }

2316 /*
2317  * Must be a comma-separated list of alpha-numeric file system names.
2318  */
2319 static int
2320 zonecfg_valid_fs_allowed(const char *fsallowedp)
2321 {
2322     char tmp[ZONE_FS_ALLOWED_MAX];
2323     char *cp = tmp;
2324     char *p;

2326     if (strlen(fsallowedp) > ZONE_FS_ALLOWED_MAX)
2327         return (Z_TOO_BIG);

2329     (void) strncpy(tmp, fsallowedp, sizeof (tmp));

2331     while (*cp != '\0') {
2332         p = cp;
2333         while (*p != '\0' && *p != ',') {
2334             if (!isalnum(*p) && *p != '-')
2335                 return (Z_INVALID_PROPERTY);
2336             p++;
2337         }

2339         if (*p == ',') {
2340             if (p == cp)
2341                 return (Z_INVALID_PROPERTY);

2343             p++;

2345             if (*p == '\0')
2346                 return (Z_INVALID_PROPERTY);
2347         }

2349         cp = p;
2350     }

2352     return (Z_OK);
2353 }

2355 int
2356 zonecfg_get_fs_allowed(zone_dochandle_t handle, char *bufp, size_t buflen)
2357 {
2358     int err;

2360     if ((err = getrootattr(handle, DTD_ATTR_FS_ALLOWED,
2361         bufp, buflen)) != Z_OK)
2362         return (err);
2363     if (bufp[0] == '\0')
2364         return (Z_BAD_PROPERTY);
2365     return (zonecfg_valid_fs_allowed(bufp));
2366 }

2368 int

```

```

2369 zonecfg_set_fs_allowed(zone_dochandle_t handle, const char *bufp)
2370 {
2371     int err;

2373     if (bufp == NULL || (err = zonecfg_valid_fs_allowed(bufp)) == Z_OK)
2374         return (setrootattr(handle, DTD_ATTR_FS_ALLOWED, bufp));
2375     return (err);
2376 }

2378 /*
2379  * Determines if the specified string is a valid hostid string. This function
2380  * returns Z_OK if the string is a valid hostid string. It returns Z_INVALID if
2381  * 'hostidp' is NULL, Z_TOO_BIG if 'hostidp' refers to a string buffer
2382  * containing a hex string with more than 8 digits, and Z_INVALID_PROPERTY if
2383  * the string has an invalid format.
2384  */
2385 static int
2386 zonecfg_valid_hostid(const char *hostidp)
2387 {
2388     char *currentp;
2389     u_longlong_t hostidval;
2390     size_t len;

2392     if (hostidp == NULL)
2393         return (Z_INVALID);

2395     /* Empty strings and strings with whitespace are invalid. */
2396     if (*hostidp == '\0')
2397         return (Z_INVALID_PROPERTY);
2398     for (currentp = (char *)hostidp; *currentp != '\0'; ++currentp) {
2399         if (isspace(*currentp))
2400             return (Z_INVALID_PROPERTY);
2401     }
2402     len = (size_t)(currentp - hostidp);

2404     /*
2405      * The caller might pass a hostid that is larger than the maximum
2406      * unsigned 32-bit integral value. Check for this! Also, make sure
2407      * that the whole string is converted (this helps us find illegal
2408      * characters) and that the whole string fits within a buffer of size
2409      * HW_HOSTID_LEN.
2410      */
2411     currentp = (char *)hostidp;
2412     if (strncmp(hostidp, "0x", 2) == 0 || strncmp(hostidp, "0X", 2) == 0)
2413         currentp += 2;
2414     hostidval = strtoull(currentp, &currentp, 16);
2415     if ((size_t)(currentp - hostidp) >= HW_HOSTID_LEN)
2416         return (Z_TOO_BIG);
2417     if (hostidval > UINT_MAX || hostidval == HW_INVALID_HOSTID ||
2418         currentp != hostidp + len)
2419         return (Z_INVALID_PROPERTY);
2420     return (Z_OK);
2421 }

2423 /*
2424  * Gets the zone hostid string stored in the specified zone configuration
2425  * document. This function returns Z_OK on success. Z_BAD_PROPERTY is returned
2426  * if the config file doesn't specify a hostid or if the hostid is blank.
2427  * Note that buflen should be at least HW_HOSTID_LEN.
2428  */
2429 int
2430 zonecfg_get_hostid(zone_dochandle_t handle, char *bufp, size_t buflen)
2431 {
2432     int err;

```

```

2435     if ((err = getrootattr(handle, DTD_ATTR_HOSTID, bufp, buflen)) != Z_OK)
2436         return (err);
2437     if (bufp[0] == '\0')
2438         return (Z_BAD_PROPERTY);
2439     return (zonecfg_valid_hostid(bufp));
2440 }

2442 /*
2443  * Sets the hostid string in the specified zone config document to the given
2444  * string value. If 'hostidp' is NULL, then the config document's hostid
2445  * attribute is cleared. Non-NULL hostids are validated. This function returns
2446  * Z_OK on success. Any other return value indicates failure.
2447  */
2448 int
2449 zonecfg_set_hostid(zone_dochandle_t handle, const char *hostidp)
2450 {
2451     int err;

2453     /*
2454      * A NULL hostid string is interpreted as a request to clear the
2455      * hostid.
2456      */
2457     if (hostidp == NULL || (err = zonecfg_valid_hostid(hostidp)) == Z_OK)
2458         return (setrootattr(handle, DTD_ATTR_HOSTID, hostidp));
2459     return (err);
2460 }

2462 int
2463 zonecfg_lookup_dev(zone_dochandle_t handle, struct zone_devtab *tabptr)
2464 {
2465     xmlNodePtr cur, firstmatch;
2466     int err;
2467     char match[MAXPATHLEN];

2469     if (tabptr == NULL)
2470         return (Z_INVALID);

2472     if ((err = operation_prep(handle)) != Z_OK)
2473         return (err);

2475     cur = handle->zone_dh_cur;
2476     firstmatch = NULL;
2477     for (cur = cur->xmlChildrenNode; cur != NULL; cur = cur->next) {
2478         if (xmlStrcmp(cur->name, DTD_ELEM_DEVICE))
2479             continue;
2480         if (strlen(tabptr->zone_dev_match) == 0)
2481             continue;

2483         if ((fetchprop(cur, DTD_ATTR_MATCH, match,
2484             sizeof(match)) == Z_OK) {
2485             if (strcmp(tabptr->zone_dev_match,
2486                 match) == 0) {
2487                 if (firstmatch == NULL)
2488                     firstmatch = cur;
2489                 else if (firstmatch != cur)
2490                     return (Z_INSUFFICIENT_SPEC);
2491             } else {
2492                 /*
2493                  * If another property matched but this
2494                  * one doesn't then reset firstmatch.
2495                  */
2496                 if (firstmatch == cur)
2497                     firstmatch = NULL;
2498             }
2499         }
2500     }

```

```

2501     if (firstmatch == NULL)
2502         return (Z_NO_RESOURCE_ID);
2504     cur = firstmatch;
2506     if ((err = fetchprop(cur, DTD_ATTR_MATCH, tabptr->zone_dev_match,
2507         sizeof (tabptr->zone_dev_match))) != Z_OK)
2508         return (err);
2510     return (Z_OK);
2511 }
2513 static int
2514 zonecfg_add_dev_core(zone_dochandle_t handle, struct zone_devtab *tabptr)
2515 {
2516     xmlNodePtr newnode, cur = handle->zone_dh_cur;
2517     int err;
2519     newnode = xmlNewTextChild(cur, NULL, DTD_ELEM_DEVICE, NULL);
2521     if ((err = newprop(newnode, DTD_ATTR_MATCH,
2522         tabptr->zone_dev_match)) != Z_OK)
2523         return (err);
2525     return (Z_OK);
2526 }
2528 int
2529 zonecfg_add_dev(zone_dochandle_t handle, struct zone_devtab *tabptr)
2530 {
2531     int err;
2533     if (tabptr == NULL)
2534         return (Z_INVALID);
2536     if ((err = operation_prep(handle)) != Z_OK)
2537         return (err);
2539     if ((err = zonecfg_add_dev_core(handle, tabptr)) != Z_OK)
2540         return (err);
2542     return (Z_OK);
2543 }
2545 static int
2546 zonecfg_delete_dev_core(zone_dochandle_t handle, struct zone_devtab *tabptr)
2547 {
2548     xmlNodePtr cur = handle->zone_dh_cur;
2549     int match_match;
2551     for (cur = cur->xmlChildrenNode; cur != NULL; cur = cur->next) {
2552         if (xmlStrcmp(cur->name, DTD_ELEM_DEVICE))
2553             continue;
2555         match_match = match_prop(cur, DTD_ATTR_MATCH,
2556             tabptr->zone_dev_match);
2558         if (match_match) {
2559             xmlUnlinkNode(cur);
2560             xmlFreeNode(cur);
2561             return (Z_OK);
2562         }
2563     }
2564     return (Z_NO_RESOURCE_ID);
2565 }

```

```

2567 int
2568 zonecfg_delete_dev(zone_dochandle_t handle, struct zone_devtab *tabptr)
2569 {
2570     int err;
2572     if (tabptr == NULL)
2573         return (Z_INVALID);
2575     if ((err = operation_prep(handle)) != Z_OK)
2576         return (err);
2578     if ((err = zonecfg_delete_dev_core(handle, tabptr)) != Z_OK)
2579         return (err);
2581     return (Z_OK);
2582 }
2584 int
2585 zonecfg_modify_dev(
2586     zone_dochandle_t handle,
2587     struct zone_devtab *oldtabptr,
2588     struct zone_devtab *newtabptr)
2589 {
2590     int err;
2592     if (oldtabptr == NULL || newtabptr == NULL)
2593         return (Z_INVALID);
2595     if ((err = operation_prep(handle)) != Z_OK)
2596         return (err);
2598     if ((err = zonecfg_delete_dev_core(handle, oldtabptr)) != Z_OK)
2599         return (err);
2601     if ((err = zonecfg_add_dev_core(handle, newtabptr)) != Z_OK)
2602         return (err);
2604     return (Z_OK);
2605 }
2607 static int
2608 zonecfg_add_auth_core(zone_dochandle_t handle, struct zone_admintab *tabptr,
2609     char *zonename)
2610 {
2611     xmlNodePtr newnode, cur = handle->zone_dh_cur;
2612     int err;
2614     newnode = xmlNewTextChild(cur, NULL, DTD_ELEM_ADMIN, NULL);
2615     err = newprop(newnode, DTD_ATTR_USER, tabptr->zone_admin_user);
2616     if (err != Z_OK)
2617         return (err);
2618     err = newprop(newnode, DTD_ATTR_AUTHS, tabptr->zone_admin_auths);
2619     if (err != Z_OK)
2620         return (err);
2621     if ((err = zonecfg_remove_userauths(
2622         handle, tabptr->zone_admin_user, zonename, B_FALSE)) != Z_OK)
2623         return (err);
2624     return (Z_OK);
2625 }
2627 int
2628 zonecfg_add_admin(zone_dochandle_t handle, struct zone_admintab *tabptr,
2629     char *zonename)
2630 {
2631     int err;

```

```

2633     if (tabptr == NULL)
2634         return (Z_INVALID);

2636     if ((err = operation_prep(handle)) != Z_OK)
2637         return (err);

2639     if ((err = zonecfg_add_auth_core(handle, tabptr,
2640         zonename)) != Z_OK)
2641         return (err);

2643     return (Z_OK);
2644 }

2646 #endif /* ! codereview */
2647 static int
2648 zonecfg_delete_auth_core(zone_dochandle_t handle, struct zone_admintab *tabptr,
2649     char *zonename)
2650 {
2651     xmlNodePtr cur = handle->zone_dh_cur;
2652     boolean_t auth_match;
2653     int err;

2655     for (cur = cur->xmlChildrenNode; cur != NULL; cur = cur->next) {
2656         if (xmlStrcmp(cur->name, DTD_ELEM_ADMIN))
2657             continue;
2658         auth_match = match_prop(cur, DTD_ATTR_USER,
2659             tabptr->zone_admin_user);
2660         if (auth_match) {
2661             if ((err = zonecfg_insert_userauths(
2662                 handle, tabptr->zone_admin_user,
2663                 zonename)) != Z_OK)
2664                 return (err);
2665             xmlUnlinkNode(cur);
2666             xmlFreeNode(cur);
2667             return (Z_OK);
2668         }
2669     }
2670     return (Z_NO_RESOURCE_ID);
2671 }

2673 int
2674 zonecfg_delete_admin(zone_dochandle_t handle, struct zone_admintab *tabptr,
2675     char *zonename)
2676 {
2677     int err;

2679     if (tabptr == NULL)
2680         return (Z_INVALID);

2682     if ((err = operation_prep(handle)) != Z_OK)
2683         return (err);

2685     if ((err = zonecfg_delete_auth_core(handle, tabptr, zonename)) != Z_OK)
2686         return (err);

2688     return (Z_OK);
2689 }

2691 int
2692 zonecfg_modify_admin(zone_dochandle_t handle, struct zone_admintab *oldtabptr,
2693     struct zone_admintab *newtabptr, char *zonename)
2694 {
2695     int err;

2697     if (oldtabptr == NULL || newtabptr == NULL)
2698         return (Z_INVALID);

```

```

2700     if ((err = operation_prep(handle)) != Z_OK)
2701         return (err);

2703     if ((err = zonecfg_delete_auth_core(handle, oldtabptr, zonename))
2704         != Z_OK)
2705         return (err);

2707     if ((err = zonecfg_add_auth_core(handle, newtabptr,
2708         zonename)) != Z_OK)
2709         return (err);

2711     return (Z_OK);
2712 }

2714 int
2715 zonecfg_lookup_admin(zone_dochandle_t handle, struct zone_admintab *tabptr)
2716 {
2717     xmlNodePtr cur, firstmatch;
2718     int err;
2719     char user[MAXUSERNAME];

2721     if (tabptr == NULL)
2722         return (Z_INVALID);

2724     if ((err = operation_prep(handle)) != Z_OK)
2725         return (err);

2727     cur = handle->zone_dh_cur;
2728     firstmatch = NULL;
2729     for (cur = cur->xmlChildrenNode; cur != NULL; cur = cur->next) {
2730         if (xmlStrcmp(cur->name, DTD_ELEM_ADMIN))
2731             continue;
2732         if (strlen(tabptr->zone_admin_user) > 0) {
2733             if ((fetchprop(cur, DTD_ATTR_USER, user,
2734                 sizeof (user)) == Z_OK) &&
2735                 (strcmp(tabptr->zone_admin_user, user) == 0)) {
2736                 if (firstmatch == NULL)
2737                     firstmatch = cur;
2738                 else
2739                     return (Z_INSUFFICIENT_SPEC);
2740             }
2741         }
2742     }
2743     if (firstmatch == NULL)
2744         return (Z_NO_RESOURCE_ID);

2746     cur = firstmatch;

2748     if ((err = fetchprop(cur, DTD_ATTR_USER, tabptr->zone_admin_user,
2749         sizeof (tabptr->zone_admin_user))) != Z_OK)
2750         return (err);

2752     if ((err = fetchprop(cur, DTD_ATTR_AUTHS, tabptr->zone_admin_auths,
2753         sizeof (tabptr->zone_admin_auths))) != Z_OK)
2754         return (err);

2756     return (Z_OK);
2757 }

2759 static int
2760 zonecfg_add_secflags_core(zone_dochandle_t handle,
2761     struct zone_secflagstab *tabptr)
2762 {
2763     xmlNodePtr newnode, cur = handle->zone_dh_cur;
2764     int err;

```

```

2766     newnode = xmlNewTextChild(cur, NULL, DTD_ELEM_SECFLAGS, NULL);
2767     err = newprop(newnode, DTD_ATTR_DEFAULT, tabptr->zone_secflags_default);
2768     if (err != Z_OK)
2769         return (err);
2770     err = newprop(newnode, DTD_ATTR_LOWER, tabptr->zone_secflags_lower);
2771     if (err != Z_OK)
2772         return (err);
2773     err = newprop(newnode, DTD_ATTR_UPPER, tabptr->zone_secflags_upper);
2774     if (err != Z_OK)
2775         return (err);
2777     return (Z_OK);
2778 }

2780 int
2781 zonecfg_add_secflags(zone_dochandle_t handle, struct zone_secflagstab *tabptr)
2782 {
2783     int err;

2786     if (tabptr == NULL)
2787         return (Z_INVALID);

2789     if ((err = operation_prep(handle)) != Z_OK)
2790         return (err);

2792     if ((err = zonecfg_add_secflags_core(handle, tabptr)) != Z_OK)
2793         return (err);

2795     return (Z_OK);
2796 }

2798 static int
2799 zonecfg_delete_secflags_core(zone_dochandle_t handle,
2800     struct zone_secflagstab *tabptr)
2801 {
2802     xmlNodePtr cur = handle->zone_dh_cur;
2803     boolean_t def_match, low_match, up_match;

2805     for (cur = cur->xmlChildrenNode; cur != NULL; cur = cur->next) {
2806         if (xmlStrcmp(cur->name, DTD_ELEM_SECFLAGS) != 0)
2807             continue;

2809         def_match = match_prop(cur, DTD_ATTR_DEFAULT,
2810             tabptr->zone_secflags_default);
2811         low_match = match_prop(cur, DTD_ATTR_LOWER,
2812             tabptr->zone_secflags_lower);
2813         up_match = match_prop(cur, DTD_ATTR_UPPER,
2814             tabptr->zone_secflags_upper);

2816         if (def_match && low_match && up_match) {
2817             xmlUnlinkNode(cur);
2818             xmlFreeNode(cur);
2819             return (Z_OK);
2820         }

2822     }
2823     return (Z_NO_RESOURCE_ID);
2824 }

2826 int
2827 zonecfg_delete_secflags(zone_dochandle_t handle,
2828     struct zone_secflagstab *tabptr)
2829 {
2830     int err;

```

```

2832     if (tabptr == NULL)
2833         return (Z_INVALID);

2835     if ((err = operation_prep(handle)) != Z_OK)
2836         return (err);

2838     if ((err = zonecfg_delete_secflags_core(handle, tabptr)) != Z_OK)
2839         return (err);

2841     return (Z_OK);
2842 }

2844 int
2845 zonecfg_modify_secflags(zone_dochandle_t handle,
2846     struct zone_secflagstab *oldtabptr,
2847     struct zone_secflagstab *newtabptr)
2848 {
2849     int err;

2851     if (oldtabptr == NULL || newtabptr == NULL)
2852         return (Z_INVALID);

2854     if ((err = operation_prep(handle)) != Z_OK)
2855         return (err);

2857     if ((err = zonecfg_delete_secflags_core(handle, oldtabptr))
2858         != Z_OK)
2859         return (err);

2861     if ((err = zonecfg_add_secflags_core(handle, newtabptr)) != Z_OK)
2862         return (err);

2864     return (Z_OK);
2865 }

2867 int
2868 zonecfg_lookup_secflags(zone_dochandle_t handle,
2869     struct zone_secflagstab *tabptr)
2870 {
2871     xmlNodePtr cur;
2872     int err;

2874     if (tabptr == NULL)
2875         return (Z_INVALID);

2877     if ((err = operation_prep(handle)) != Z_OK)
2878         return (err);

2880     cur = handle->zone_dh_cur;

2882     for (cur = cur->xmlChildrenNode; cur != NULL; cur = cur->next) {
2883         if (xmlStrcmp(cur->name, DTD_ELEM_SECFLAGS) != 0)
2884             continue;

2886         if ((err = fetchprop(cur, DTD_ATTR_DEFAULT,
2887             tabptr->zone_secflags_default,
2888             sizeof(tabptr->zone_secflags_default))) != Z_OK) {
2889             handle->zone_dh_cur = handle->zone_dh_top;
2890             return (err);
2891         }

2893         if ((err = fetchprop(cur, DTD_ATTR_LOWER,
2894             tabptr->zone_secflags_lower,
2895             sizeof(tabptr->zone_secflags_lower))) != Z_OK) {
2896             handle->zone_dh_cur = handle->zone_dh_top;

```

```

2897         return (err);
2898     }

2900     if ((err = fetchprop(cur, DTD_ATTR_UPPER,
2901         tabptr->zone_secflags_upper,
2902         sizeof (tabptr->zone_secflags_upper))) != Z_OK) {
2903         handle->zone_dh_cur = handle->zone_dh_top;
2904         return (err);
2905     }

2907     return (Z_OK);
2908 }

2910     return (Z_NO_ENTRY);
2911 }
2912 #endif /* ! codereview */

2914 /* Lock to serialize all devwalks */
2915 static pthread_mutex_t zonecfg_devwalk_lock = PTHREAD_MUTEX_INITIALIZER;
2916 /*
2917  * Global variables used to pass data from zonecfg_dev_manifest to the nftw
2918  * call-back (zonecfg_devwalk_cb).  g_devwalk_data is really the void*
2919  * parameter and g_devwalk_cb is really the *cb parameter from
2920  * zonecfg_dev_manifest.
2921  */
2922 typedef struct __g_devwalk_data *g_devwalk_data_t;
2923 static g_devwalk_data_t g_devwalk_data;
2924 static int (*g_devwalk_cb)(const char *, uid_t, gid_t, mode_t, const char *,
2925     void *);
2926 static size_t g_devwalk_skip_prefix;

2928 /*
2929  * zonecfg_dev_manifest call-back function used during detach to generate the
2930  * dev info in the manifest.
2931  */
2932 static int
2933 get_detach_dev_entry(const char *name, uid_t uid, gid_t gid, mode_t mode,
2934     const char *acl, void *hdl)
2935 {
2936     zone_dochandle_t handle = (zone_dochandle_t)hdl;
2937     xmlNodePtr newnode;
2938     xmlNodePtr cur;
2939     int err;
2940     char buf[128];

2942     if ((err = operation_prep(handle)) != Z_OK)
2943         return (err);

2945     cur = handle->zone_dh_cur;
2946     newnode = xmlNewTextChild(cur, NULL, DTD_ELEM_DEV_PERM, NULL);
2947     if ((err = newprop(newnode, DTD_ATTR_NAME, (char *)name)) != Z_OK)
2948         return (err);
2949     (void) snprintf(buf, sizeof (buf), "%lu", uid);
2950     if ((err = newprop(newnode, DTD_ATTR_UID, buf)) != Z_OK)
2951         return (err);
2952     (void) snprintf(buf, sizeof (buf), "%lu", gid);
2953     if ((err = newprop(newnode, DTD_ATTR_GID, buf)) != Z_OK)
2954         return (err);
2955     (void) snprintf(buf, sizeof (buf), "%o", mode);
2956     if ((err = newprop(newnode, DTD_ATTR_MODE, buf)) != Z_OK)
2957         return (err);
2958     if ((err = newprop(newnode, DTD_ATTR_ACL, (char *)acl)) != Z_OK)
2959         return (err);
2960     return (Z_OK);
2961 }

```

```

2963 /*
2964  * This is the nftw call-back function used by zonecfg_dev_manifest.  It is
2965  * responsible for calling the actual call-back.
2966  */
2967 /* ARGSUSED2 */
2968 static int
2969 zonecfg_devwalk_cb(const char *path, const struct stat *st, int f,
2970     struct FTW *ftw)
2971 {
2972     acl_t *acl;
2973     char *acl_txt = NULL;

2975     /* skip all but character and block devices */
2976     if (!S_ISBLK(st->st_mode) && !S_ISCHR(st->st_mode))
2977         return (0);

2979     if ((acl_get(path, ACL_NO_TRIVIAL, &acl) == 0) &&
2980         acl != NULL) {
2981         acl_txt = acl_totext(acl, ACL_NORESOLVE);
2982         acl_free(acl);
2983     }

2985     if (strlen(path) <= g_devwalk_skip_prefix)
2986         return (0);

2988     g_devwalk_cb(path + g_devwalk_skip_prefix, st->st_uid, st->st_gid,
2989         st->st_mode & S_IAMB, acl_txt != NULL ? acl_txt : "",
2990         g_devwalk_data);
2991     free(acl_txt);
2992     return (0);
2993 }

2995 /*
2996  * Walk the dev tree for the zone specified by hdl and call the
2997  * get_detach_dev_entry call-back function for each entry in the tree.  The
2998  * call-back will be passed the name, uid, gid, mode, acl string and the
2999  * handle input parameter for each dev entry.
3000  */
3001 /* Data is passed to get_detach_dev_entry through the global variables
3002  * g_devwalk_data, *g_devwalk_cb, and g_devwalk_skip_prefix.  The
3003  * zonecfg_devwalk_cb function will actually call get_detach_dev_entry.
3004  */
3005 int
3006 zonecfg_dev_manifest(zone_dochandle_t hdl)
3007 {
3008     char path[MAXPATHLEN];
3009     int ret;

3011     if ((ret = zonecfg_get_zonepath(hdl, path, sizeof (path))) != Z_OK)
3012         return (ret);

3014     if (strlen(path, "/dev", sizeof (path)) >= sizeof (path))
3015         return (Z_TOO_BIG);

3017     /*
3018      * We have to serialize all devwalks in the same process
3019      * (which should be fine), since nftw() is so badly designed.
3020      */
3021     (void) pthread_mutex_lock(&zonecfg_devwalk_lock);

3023     g_devwalk_skip_prefix = strlen(path) + 1;
3024     g_devwalk_data = (g_devwalk_data_t)hdl;
3025     g_devwalk_cb = get_detach_dev_entry;
3026     (void) nftw(path, zonecfg_devwalk_cb, 0, FTW_PHYS);

3028     (void) pthread_mutex_unlock(&zonecfg_devwalk_lock);

```

```

3029     return (Z_OK);
3030 }

3032 /*
3033  * Update the owner, group, mode and acl on the specified dev (inpath) for
3034  * the zone (hdl). This function can be used to fix up the dev tree after
3035  * attaching a migrated zone.
3036  */
3037 int
3038 zonecfg_devperms_apply(zone_dochandle_t hdl, const char *inpath, uid_t owner,
3039     gid_t group, mode_t mode, const char *acltxt)
3040 {
3041     int ret;
3042     char path[MAXPATHLEN];
3043     struct stat st;
3044     acl_t *aclp;

3046     if ((ret = zonecfg_get_zonepath(hdl, path, sizeof (path))) != Z_OK)
3047         return (ret);

3049     if (strlcat(path, "/dev/", sizeof (path)) >= sizeof (path))
3050         return (Z_TOO_BIG);
3051     if (strlcat(path, inpath, sizeof (path)) >= sizeof (path))
3052         return (Z_TOO_BIG);

3054     if (stat(path, &st) == -1)
3055         return (Z_INVALID);

3057     /* make sure we're only touching device nodes */
3058     if (!S_ISCHR(st.st_mode) && !S_ISBLK(st.st_mode))
3059         return (Z_INVALID);

3061     if (chown(path, owner, group) == -1)
3062         return (Z_SYSTEM);

3064     if (chmod(path, mode) == -1)
3065         return (Z_SYSTEM);

3067     if ((acltxt == NULL) || (strcmp(acltxt, "") == 0))
3068         return (Z_OK);

3070     if (acl_fromtext(acltxt, &aclp) != 0) {
3071         errno = EINVAL;
3072         return (Z_SYSTEM);
3073     }

3075     errno = 0;
3076     if (acl_set(path, aclp) == -1) {
3077         free(aclp);
3078         return (Z_SYSTEM);
3079     }

3081     free(aclp);
3082     return (Z_OK);
3083 }

3085 /*
3086  * This function finds everything mounted under a zone's rootpath.
3087  * This returns the number of mounts under rootpath, or -1 on error.
3088  * callback is called once per mount found with the first argument
3089  * pointing to a mnttab structure containing the mount's information.
3090  *
3091  * If the callback function returns non-zero zonecfg_find_mounts
3092  * aborts with an error.
3093  */
3094 int

```

```

3095 zonecfg_find_mounts(char *rootpath, int (*callback)(const struct mnttab *,
3096     void *), void *priv)
3097 {
3098     void *, void *priv) {
3099     FILE *mnttab;
3100     struct mnttab m;
3101     size_t l;
3102     int zfs1;
3103     int rv = 0;
3104     char zfs_path[MAXPATHLEN];

3105     assert(rootpath != NULL);

3107     if ((zfs1 = snprintf(zfs_path, sizeof (zfs_path), "%s/.zfs/", rootpath))
3108         >= sizeof (zfs_path))
3109         return (-1);

3111     l = strlen(rootpath);

3113     mnttab = fopen("/etc/mnttab", "r");

3115     if (mnttab == NULL)
3116         return (-1);

3118     if (ioctl(fileno(mnttab), MNTIOC_SHOWHIDDEN, NULL) < 0) {
3119         rv = -1;
3120         goto out;
3121     }

3123     while (!getmntent(mnttab, &m)) {
3124         if ((strcmp(rootpath, m.mnt_mountp, l) == 0) &&
3125             (m.mnt_mountp[l] == '/') &&
3126             (strcmp(zfs_path, m.mnt_mountp, zfs1) != 0)) {
3127             rv++;
3128             if (callback == NULL)
3129                 continue;
3130             if (callback(&m, priv)) {
3131                 rv = -1;
3132                 goto out;
3133             }
3134         }
3135     }
3136 }

3138 out:
3139     (void) fclose(mnttab);
3140     return (rv);
3141 }

unchanged_portion_omitted

7088 int
7089 zonecfg_getsecflagsent(zone_dochandle_t handle,
7090     struct zone_secflagstab *tabptr)
7091 {
7092     int err;
7093     xmlNodePtr cur;

7095     if (handle == NULL)
7096         return (Z_INVALID);

7098     if ((err = zonecfg_setent(handle)) != Z_OK)
7099         return (err);

7102     if ((cur = handle->zone_dh_cur) == NULL)
7103         return (Z_NO_ENTRY);

```

```

7105     for (; cur != NULL; cur = cur->next) {
7106         if (xmlStrcmp(cur->name, DTD_ELEM_SECFLAGS) == 0)
7107             break;
7108     }
7110     if (cur == NULL) {
7111         handle->zone_dh_cur = handle->zone_dh_top;
7112         return (Z_NO_ENTRY);
7113     }
7115     if ((err = fetchprop(cur, DTD_ATTR_DEFAULT,
7116         tabptr->zone_secflags_default,
7117         sizeof (tabptr->zone_secflags_default))) != Z_OK) {
7118         handle->zone_dh_cur = handle->zone_dh_top;
7119         return (err);
7120     }
7122     if ((err = fetchprop(cur, DTD_ATTR_LOWER,
7123         tabptr->zone_secflags_lower,
7124         sizeof (tabptr->zone_secflags_lower))) != Z_OK) {
7125         handle->zone_dh_cur = handle->zone_dh_top;
7126         return (err);
7127     }
7129     if ((err = fetchprop(cur, DTD_ATTR_UPPER,
7130         tabptr->zone_secflags_upper,
7131         sizeof (tabptr->zone_secflags_upper))) != Z_OK) {
7132         handle->zone_dh_cur = handle->zone_dh_top;
7133         return (err);
7134     }
7136     handle->zone_dh_cur = cur->next;
7138     (void) zonecfg_endent(handle);
7140     return (err);
7141 }
7143 #endif /* ! codereview */
7144 static int
7145 getmcapent_core(zone_dochandle_t handle, struct zone_mcaptab *tabptr)
7146 {
7147     xmlNodePtr cur;
7148     int err;
7150     if (handle == NULL)
7151         return (Z_INVALID);
7153     if ((cur = handle->zone_dh_cur) == NULL)
7154         return (Z_NO_ENTRY);
7156     for (; cur != NULL; cur = cur->next)
7157         if (xmlStrcmp(cur->name, DTD_ELEM_MCAP) == 0)
7158             break;
7159     if (cur == NULL) {
7160         handle->zone_dh_cur = handle->zone_dh_top;
7161         return (Z_NO_ENTRY);
7162     }
7164     if ((err = fetchprop(cur, DTD_ATTR_PHYSCAP, tabptr->zone_physmem_cap,
7165         sizeof (tabptr->zone_physmem_cap))) != Z_OK) {
7166         handle->zone_dh_cur = handle->zone_dh_top;
7167         return (err);
7168     }

```

```

7170         handle->zone_dh_cur = cur->next;
7171         return (Z_OK);
7172     }
7174     int
7175     zonecfg_getmcapent(zone_dochandle_t handle, struct zone_mcaptab *tabptr)
7176     {
7177         int err;
7179         if ((err = zonecfg_setent(handle)) != Z_OK)
7180             return (err);
7182         err = getmcapent_core(handle, tabptr);
7184         (void) zonecfg_endent(handle);
7186         return (err);
7187     }
7189     /*
7190     * Get the full tree of pkg metadata in a set of nested AVL trees.
7191     * pkgs_avl is an AVL tree of pkgs.
7192     *
7193     * The zone xml data contains DTD_ELEM_PACKAGE elements.
7194     */
7195     int
7196     zonecfg_getpkgdata(zone_dochandle_t handle, uu_avl_pool_t *pkg_pool,
7197         uu_avl_t *pkgs_avl)
7198     {
7199         xmlNodePtr cur;
7200         int res;
7201         zone_pkg_entry_t *pkg;
7202         char name[MAXNAMELEN];
7203         char version[ZONE_PKG_VERSMAX];
7205         if (handle == NULL)
7206             return (Z_INVALID);
7208         if ((res = zonecfg_setent(handle)) != Z_OK)
7209             return (res);
7211         if ((cur = handle->zone_dh_cur) == NULL) {
7212             res = Z_NO_ENTRY;
7213             goto done;
7214         }
7216         for (; cur != NULL; cur = cur->next) {
7217             if (xmlStrcmp(cur->name, DTD_ELEM_PACKAGE) == 0) {
7218                 uu_avl_index_t where;
7220                 if ((res = fetchprop(cur, DTD_ATTR_NAME, name,
7221                     sizeof (name))) != Z_OK)
7222                     goto done;
7224                 if ((res = fetchprop(cur, DTD_ATTR_VERSION, version,
7225                     sizeof (version))) != Z_OK)
7226                     goto done;
7228                 if ((pkg = (zone_pkg_entry_t *)
7229                     malloc(sizeof (zone_pkg_entry_t))) == NULL) {
7230                     res = Z_NOMEM;
7231                     goto done;
7232                 }
7234                 if ((pkg->zpe_name = strdup(name)) == NULL) {
7235                     free(pkg);

```



```

7236         res = Z_NOMEM;
7237         goto done;
7238     }
7240     if ((pkg->zpe_vers = strdup(version)) == NULL) {
7241         free(pkg->zpe_name);
7242         free(pkg);
7243         res = Z_NOMEM;
7244         goto done;
7245     }
7247     uu_avl_node_init(pkg, &pkg->zpe_entry, pkg_pool);
7248     if (uu_avl_find(pkgs_avl, pkg, NULL, &where) != NULL) {
7249         free(pkg->zpe_name);
7250         free(pkg->zpe_vers);
7251         free(pkg);
7252     } else {
7253         uu_avl_insert(pkgs_avl, pkg, where);
7254     }
7255 }
7256
7258 done:
7259     (void) zonecfg_endent(handle);
7260     return (res);
7261 }
7263 int
7264 zonecfg_setdevperment(zone_dochandle_t handle)
7265 {
7266     return (zonecfg_setent(handle));
7267 }
7269 int
7270 zonecfg_getdevperment(zone_dochandle_t handle, struct zone_devpermtab *tabptr)
7271 {
7272     xmlNodePtr cur;
7273     int err;
7274     char buf[128];
7276     tabptr->zone_devperm_acl = NULL;
7278     if (handle == NULL)
7279         return (Z_INVALID);
7281     if ((cur = handle->zone_dh_cur) == NULL)
7282         return (Z_NO_ENTRY);
7284     for (; cur != NULL; cur = cur->next)
7285         if (!xmlStrcmp(cur->name, DTD_ELEM_DEV_PERM))
7286             break;
7287     if (cur == NULL) {
7288         handle->zone_dh_cur = handle->zone_dh_top;
7289         return (Z_NO_ENTRY);
7290     }
7292     if ((err = fetchprop(cur, DTD_ATTR_NAME, tabptr->zone_devperm_name,
7293         sizeof (tabptr->zone_devperm_name))) != Z_OK) {
7294         handle->zone_dh_cur = handle->zone_dh_top;
7295         return (err);
7296     }
7298     if ((err = fetchprop(cur, DTD_ATTR_UID, buf, sizeof (buf))) != Z_OK) {
7299         handle->zone_dh_cur = handle->zone_dh_top;
7300         return (err);
7301     }

```

```

7302     tabptr->zone_devperm_uid = (uid_t)atol(buf);
7304     if ((err = fetchprop(cur, DTD_ATTR_GID, buf, sizeof (buf))) != Z_OK) {
7305         handle->zone_dh_cur = handle->zone_dh_top;
7306         return (err);
7307     }
7308     tabptr->zone_devperm_gid = (gid_t)atol(buf);
7310     if ((err = fetchprop(cur, DTD_ATTR_MODE, buf, sizeof (buf))) != Z_OK) {
7311         handle->zone_dh_cur = handle->zone_dh_top;
7312         return (err);
7313     }
7314     tabptr->zone_devperm_mode = (mode_t)strtol(buf, (char **)NULL, 8);
7316     if ((err = fetch_alloc_prop(cur, DTD_ATTR_ACL,
7317         &(tabptr->zone_devperm_acl))) != Z_OK) {
7318         handle->zone_dh_cur = handle->zone_dh_top;
7319         return (err);
7320     }
7322     handle->zone_dh_cur = cur->next;
7323     return (Z_OK);
7324 }
7326 int
7327 zonecfg_enddevperment(zone_dochandle_t handle)
7328 {
7329     return (zonecfg_endent(handle));
7330 }
7332 /* PRINTFLIKE1 */
7333 static void
7334 zerror(const char *zone_name, const char *fmt, ...)
7335 {
7336     va_list alist;
7338     va_start(alist, fmt);
7339     (void) fprintf(stderr, "zone '%s': ", zone_name);
7340     (void) vfprintf(stderr, fmt, alist);
7341     (void) fprintf(stderr, "\n");
7342     va_end(alist);
7343 }
7345 static void
7346 zperror(const char *str)
7347 {
7348     (void) fprintf(stderr, "%s: %s\n", str, strerror(errno));
7349 }
7351 /*
7352  * The following three routines implement a simple locking mechanism to
7353  * ensure that only one instance of zoneadm at a time is able to manipulate
7354  * a given zone. The lock is built on top of an fcntl(2) lock of
7355  * [/var/run/zones/<zonename>.zoneadm.lock. If a zoneadm instance
7356  * can grab that lock, it is allowed to manipulate the zone.
7357  *
7358  * Since zoneadm may call external applications which in turn invoke
7359  * zoneadm again, we introduce the notion of "lock inheritance". Any
7360  * instance of zoneadm that has another instance in its ancestry is assumed
7361  * to be acting on behalf of the original zoneadm, and is thus allowed to
7362  * manipulate its zone.
7363  *
7364  * This inheritance is implemented via the _ZONEADM_LOCK_HELD environment
7365  * variable. When zoneadm is granted a lock on its zone, this environment
7366  * variable is set to 1. When it releases the lock, the variable is set to
7367  * 0. Since a child process inherits its parent's environment, checking

```

```

7368 * the state of this variable indicates whether or not any ancestor owns
7369 * the lock.
7370 */
7371 void
7372 zonecfg_init_lock_file(const char *zone_name, char **lock_env)
7373 {
7374     *lock_env = getenv(LOCK_ENV_VAR);
7375     if (*lock_env == NULL) {
7376         if (putenv(zoneadm_lock_not_held) != 0) {
7377             zerror(zone_name, gettext("could not set env: %s"),
7378                 strerror(errno));
7379             exit(1);
7380         }
7381     } else {
7382         if (atoi(*lock_env) == 1)
7383             zone_lock_cnt = 1;
7384     }
7385 }

7387 void
7388 zonecfg_release_lock_file(const char *zone_name, int lockfd)
7389 {
7390     /*
7391     * If we are cleaning up from a failed attempt to lock the zone for
7392     * the first time, we might have a zone_lock_cnt of 0. In that
7393     * error case, we don't want to do anything but close the lock
7394     * file.
7395     */
7396     assert(zone_lock_cnt >= 0);
7397     if (zone_lock_cnt > 0) {
7398         assert(getenv(LOCK_ENV_VAR) != NULL);
7399         assert(atoi(getenv(LOCK_ENV_VAR)) == 1);
7400         if (--zone_lock_cnt > 0) {
7401             assert(lockfd == -1);
7402             return;
7403         }
7404         if (putenv(zoneadm_lock_not_held) != 0) {
7405             zerror(zone_name, gettext("could not set env: %s"),
7406                 strerror(errno));
7407             exit(1);
7408         }
7409     }
7410     assert(lockfd >= 0);
7411     (void) close(lockfd);
7412 }

7414 int
7415 zonecfg_grab_lock_file(const char *zone_name, int *lockfd)
7416 {
7417     char pathbuf[PATH_MAX];
7418     struct flock flock;

7420     /*
7421     * If we already have the lock, we can skip this expensive song
7422     * and dance.
7423     */
7424     assert(zone_lock_cnt >= 0);
7425     assert(getenv(LOCK_ENV_VAR) != NULL);
7426     if (zone_lock_cnt > 0) {
7427         assert(atoi(getenv(LOCK_ENV_VAR)) == 1);
7428         zone_lock_cnt++;
7429         *lockfd = -1;
7430         return (Z_OK);
7431     }
7432     assert(getenv(LOCK_ENV_VAR) != NULL);
7433     assert(atoi(getenv(LOCK_ENV_VAR)) == 0);

```

```

7435     if (snprintf(pathbuf, sizeof (pathbuf), "%s%s", zonecfg_get_root(),
7436                 ZONES_TMPDIR) >= sizeof (pathbuf)) {
7437         zerror(zone_name, gettext("alternate root path is too long"));
7438         return (-1);
7439     }
7440     if (mkdir(pathbuf, S_IRWXU) < 0 && errno != EEXIST) {
7441         zerror(zone_name, gettext("could not mkdir %s: %s"), pathbuf,
7442             strerror(errno));
7443         return (-1);
7444     }
7445     (void) chmod(pathbuf, S_IRWXU);

7447     /*
7448     * One of these lock files is created for each zone (when needed).
7449     * The lock files are not cleaned up (except on system reboot),
7450     * but since there is only one per zone, there is no resource
7451     * starvation issue.
7452     */
7453     if (snprintf(pathbuf, sizeof (pathbuf), "%s%s/%s.zoneadm.lock",
7454                 zonecfg_get_root(), ZONES_TMPDIR, zone_name) >= sizeof (pathbuf)) {
7455         zerror(zone_name, gettext("alternate root path is too long"));
7456         return (-1);
7457     }
7458     if ((*lockfd = open(pathbuf, O_RDWR|O_CREAT, S_IRUSR|S_IWUSR)) < 0) {
7459         zerror(zone_name, gettext("could not open %s: %s"), pathbuf,
7460             strerror(errno));
7461         return (-1);
7462     }
7463     /*
7464     * Lock the file to synchronize with other zoneadms
7465     */
7466     flock.l_type = F_WRLCK;
7467     flock.l_whence = SEEK_SET;
7468     flock.l_start = (off_t)0;
7469     flock.l_len = (off_t)0;
7470     if ((fcntl(*lockfd, F_SETLK, &flock) < 0) ||
7471         (putenv(zoneadm_lock_held) != 0)) {
7472         zerror(zone_name, gettext("unable to lock %s: %s"), pathbuf,
7473             strerror(errno));
7474         zonecfg_release_lock_file(zone_name, *lockfd);
7475         return (-1);
7476     }
7477     zone_lock_cnt = 1;
7478     return (Z_OK);
7479 }

7481 boolean_t
7482 zonecfg_lock_file_held(int *lockfd)
7483 {
7484     if (*lockfd >= 0 || zone_lock_cnt > 0)
7485         return (B_TRUE);
7486     return (B_FALSE);
7487 }

7489 static boolean_t
7490 get_doorname(const char *zone_name, char *buffer)
7491 {
7492     return (snprintf(buffer, PATH_MAX, "%s" ZONE_DOOR_PATH,
7493                     zonecfg_get_root(), zone_name) < PATH_MAX);
7494 }

7496 /*
7497 * system daemons are not audited. For the global zone, this occurs
7498 * "naturally" since init is started with the default audit
7499 * characteristics. Since zoneadmd is a system daemon and it starts

```

```

7500 * init for a zone, it is necessary to clear out the audit
7501 * characteristics inherited from whomever started zoneadmd. This is
7502 * indicated by the audit id, which is set from the ruid parameter of
7503 * adt_set_user(), below.
7504 */

7506 static void
7507 prepare_audit_context(const char *zone_name)
7508 {
7509     adt_session_data_t    *ah;
7510     char                  *failure = gettext("audit failure: %s");

7512     if (adt_start_session(&ah, NULL, 0)) {
7513         zerror(zone_name, failure, strerror(errno));
7514         return;
7515     }
7516     if (adt_set_user(ah, ADT_NO_AUDIT, ADT_NO_AUDIT,
7517                    ADT_NO_AUDIT, ADT_NO_AUDIT, NULL, ADT_NEW)) {
7518         zerror(zone_name, failure, strerror(errno));
7519         (void) adt_end_session(ah);
7520         return;
7521     }
7522     if (adt_set_proc(ah))
7523         zerror(zone_name, failure, strerror(errno));

7525     (void) adt_end_session(ah);
7526 }

7528 static int
7529 start_zoneadmd(const char *zone_name, boolean_t lock)
7530 {
7531     char doorpath[PATH_MAX];
7532     pid_t child_pid;
7533     int error = -1;
7534     int doorfd, lockfd;
7535     struct door_info info;

7537     if (!get_doorname(zone_name, doorpath))
7538         return (-1);

7540     if (lock)
7541         if (zonecfg_grab_lock_file(zone_name, &lockfd) != Z_OK)
7542             return (-1);

7544     /*
7545      * Now that we have the lock, re-confirm that the daemon is
7546      * *not* up and working fine. If it is still down, we have a green
7547      * light to start it.
7548      */
7549     if ((doorfd = open(doorpath, O_RDONLY)) < 0) {
7550         if (errno != ENOENT) {
7551             zerror(doorpath);
7552             goto out;
7553         }
7554     } else {
7555         if (door_info(doorfd, &info) == 0 &&
7556             ((info.di_attributes & DOOR_REVOKED) == 0)) {
7557             error = Z_OK;
7558             (void) close(doorfd);
7559             goto out;
7560         }
7561         (void) close(doorfd);
7562     }

7564     if ((child_pid = fork()) == -1) {
7565         zerror(gettext("could not fork"));

```

```

7566         goto out;
7567     }

7569     if (child_pid == 0) {
7570         const char *argv[6], **ap;

7572         /* child process */
7573         prepare_audit_context(zone_name);

7575         ap = argv;
7576         *ap++ = "zoneadmd";
7577         *ap++ = "-z";
7578         *ap++ = zone_name;
7579         if (zonecfg_in_alt_root()) {
7580             *ap++ = "-R";
7581             *ap++ = zonecfg_get_root();
7582         }
7583         *ap = NULL;

7585         (void) execv("/usr/lib/zones/zoneadmd", (char * const *)argv);
7586         /*
7587          * TRANSLATION_NOTE
7588          * zoneadmd is a literal that should not be translated.
7589          */
7590         zerror(gettext("could not exec zoneadmd"));
7591         _exit(1);
7592     } else {
7593         /* parent process */
7594         pid_t retval;
7595         int pstatus = 0;

7597         do {
7598             retval = waitpid(child_pid, &pstatus, 0);
7599         } while (retval != child_pid);
7600         if (WIFSIGNALED(pstatus) || (WIFEXITED(pstatus) &&
7601             WEXITSTATUS(pstatus) != 0)) {
7602             zerror(zone_name, gettext("could not start %s"),
7603                  "zoneadmd");
7604             goto out;
7605         }
7606     }
7607     error = Z_OK;
7608 out:
7609     if (lock)
7610         zonecfg_release_lock_file(zone_name, lockfd);
7611     return (error);
7612 }

7614 int
7615 zonecfg_ping_zoneadmd(const char *zone_name)
7616 {
7617     char doorpath[PATH_MAX];
7618     int doorfd;
7619     struct door_info info;

7621     if (!get_doorname(zone_name, doorpath))
7622         return (-1);

7624     if ((doorfd = open(doorpath, O_RDONLY)) < 0) {
7625         return (-1);
7626     }
7627     if (door_info(doorfd, &info) == 0 &&
7628         ((info.di_attributes & DOOR_REVOKED) == 0)) {
7629         (void) close(doorfd);
7630         return (Z_OK);
7631     }

```

```

7632     (void) close(doorfd);
7633     return (-1);
7634 }

7636 int
7637 zonecfg_call_zoneadmd(const char *zone_name, zone_cmd_arg_t *arg, char *locale,
7638     boolean_t lock)
7639 {
7640     char doorpath[PATH_MAX];
7641     int doorfd, result;
7642     door_arg_t darg;

7644     zoneid_t zoneid;
7645     uint64_t uniqid = 0;

7647     zone_cmd_rval_t *rvalp;
7648     size_t rlen;
7649     char *cp, *errbuf;

7651     rlen = getpagesize();
7652     if ((rvalp = malloc(rlen)) == NULL) {
7653         zerror(zone_name, gettext("failed to allocate %lu bytes: %s"),
7654             rlen, strerror(errno));
7655         return (-1);
7656     }

7658     if ((zoneid = getzoneidbyname(zone_name)) != ZONE_ID_UNDEFINED) {
7659         (void) zone_getattr(zoneid, ZONE_ATTR_UNIQID, &uniqid,
7660             sizeof(uniqid));
7661     }
7662     arg->uniqid = uniqid;
7663     (void) strncpy(arg->locale, locale, sizeof(arg->locale));
7664     if (!get_doorname(zone_name, doorpath)) {
7665         zerror(zone_name, gettext("alternate root path is too long"));
7666         free(rvalp);
7667         return (-1);
7668     }

7670     /*
7671     * Loop trying to start zoneadmd; if something goes seriously
7672     * wrong we break out and fail.
7673     */
7674     for (;;) {
7675         if (start_zoneadmd(zone_name, lock) != Z_OK)
7676             break;

7678         if ((doorfd = open(doorpath, O_RDONLY)) < 0) {
7679             zerror(gettext("failed to open zone door"));
7680             break;
7681         }

7683         darg.data_ptr = (char *)arg;
7684         darg.data_size = sizeof(*arg);
7685         darg.desc_ptr = NULL;
7686         darg.desc_num = 0;
7687         darg.rbuf = (char *)rvalp;
7688         darg.rsize = rlen;
7689         if (door_call(doorfd, &darg) != 0) {
7690             (void) close(doorfd);
7691             /*
7692             * We'll get EBADF if the door has been revoked.
7693             */
7694             if (errno != EBADF) {
7695                 zerror(gettext("door_call failed"));
7696                 break;
7697             }

```

```

7698         continue;      /* take another lap */
7699     }
7700     (void) close(doorfd);

7702     if (darg.data_size == 0) {
7703         /* Door server is going away; kick it again. */
7704         continue;
7705     }

7707     errbuf = rvalp->errbuf;
7708     while (*errbuf != '\0') {
7709         /*
7710         * Remove any newlines since zerror()
7711         * will append one automatically.
7712         */
7713         cp = strchr(errbuf, '\n');
7714         if (cp != NULL)
7715             *cp = '\0';
7716         zerror(zone_name, "%s", errbuf);
7717         if (cp == NULL)
7718             break;
7719         errbuf = cp + 1;
7720     }
7721     result = rvalp->rval == 0 ? 0 : -1;
7722     free(rvalp);
7723     return (result);
7724 }

7726     free(rvalp);
7727     return (-1);
7728 }

7730 boolean_t
7731 zonecfg_valid_auths(const char *auths, const char *zonename)
7732 {
7733     char *right;
7734     char *tmpauths;
7735     char *lasts;
7736     char authname[MAXAUTHS];
7737     boolean_t status = B_TRUE;

7739     tmpauths = strdup(auths);
7740     if (tmpauths == NULL) {
7741         zerror(zonename, gettext("Out of memory"));
7742         return (B_FALSE);
7743     }
7744     right = strtok_r(tmpauths, ",", &lasts);
7745     while (right != NULL) {
7746         (void) snprintf(authname, MAXAUTHS, "%s%s",
7747             ZONE_AUTH_PREFIX, right);
7748         if (getauthnam(authname) == NULL) {
7749             status = B_FALSE;
7750             zerror(zonename,
7751                 gettext("'%' is not a valid authorization",
7752                     right));
7753         }
7754         right = strtok_r(NULL, ",", &lasts);
7755     }
7756     free(tmpauths);
7757     return (status);
7758 }

7760 int
7761 zonecfg_delete_admins(zone_dochandle_t handle, char *zonename)
7762 {
7763     int err;

```

```

7764 struct zone_admintab adminTAB;
7765 boolean_t changed = B_FALSE;

7767 if ((err = zonecfg_setadminent(handle)) != Z_OK) {
7768     return (err);
7769 }
7770 while (zonecfg_getadminent(handle, &adminTAB) == Z_OK) {
7771     err = zonecfg_delete_admin(handle, &adminTAB,
7772     zonename);
7773     if (err != Z_OK) {
7774         (void) zonecfg_endadminent(handle);
7775         return (err);
7776     } else {
7777         changed = B_TRUE;
7778     }
7779     if ((err = zonecfg_setadminent(handle)) != Z_OK) {
7780         return (err);
7781     }
7782 }
7783 (void) zonecfg_endadminent(handle);
7784 return (changed? Z_OK:Z_NO_ENTRY);
7785 }

7787 /*
7788  * Checks if a long authorization applies to this zone.
7789  * If so, it returns true, after destructively stripping
7790  * the authorization of its prefix and zone suffix.
7791  */
7792 static boolean_t
7793 is_zone_auth(char **auth, char *zonename, char *oldzonename)
7794 {
7795     char *suffix;
7796     size_t offset;

7798     offset = strlen(ZONE_AUTH_PREFIX);
7799     if ((strcmp(*auth, ZONE_AUTH_PREFIX, offset) == 0) &&
7800         ((suffix = strchr(*auth, '/') != NULL)) {
7801         if (strcmp(suffix + 1, zonename) == 0) {
7802             *auth += offset;
7803             suffix[0] = '\0';
7804             return (B_TRUE);
7805         } else if ((oldzonename != NULL) &&
7806             (strcmp(suffix + 1, oldzonename) == 0)) {
7807             *auth += offset;
7808             suffix[0] = '\0';
7809             return (B_TRUE);
7810         }
7811     }
7812     return (B_FALSE);
7813 }

7815 /*
7816  * This function determines whether the zone-specific authorization
7817  * assignments in /etc/user_attr have been changed more recently
7818  * than the equivalent data stored in the zone's configuration file.
7819  * This should only happen if the zone-specific authorizations in
7820  * the user_attr file were modified using a tool other than zonecfg.
7821  * If the configuration file is out-of-date with respect to these
7822  * authorization assignments, it is updated to match those specified
7823  * in /etc/user_attr.
7824  */

7826 int
7827 zonecfg_update_userauths(zone_dochange_t handle, char *zonename)
7828 {
7829     userattr_t *ua_ptr;

```

```

7830 char *authlist;
7831 char *lasts;
7832 FILE *uaf;
7833 struct zone_admintab adminTAB;
7834 struct stat config_st, ua_st;
7835 char config_file[MAXPATHLEN];
7836 boolean_t changed = B_FALSE;
7837 int err;

7839 if ((uaf = fopen(USERATTR_FILENAME, "r")) == NULL) {
7840     zerror(zonename, gettext("could not open file %s: %s"),
7841     USERATTR_FILENAME, strerror(errno));
7842     if (errno == EACCES)
7843         return (Z_ACCES);
7844     if (errno == ENOENT)
7845         return (Z_NO_ZONE);
7846     return (Z_MISC_FS);
7847 }
7848 if ((err = fstat(fileno(uaf), &ua_st)) != 0) {
7849     zerror(zonename, gettext("could not stat file %s: %s"),
7850     USERATTR_FILENAME, strerror(errno));
7851     (void) fclose(uaf);
7852     return (Z_MISC_FS);
7853 }
7854 if (!config_file_path(zonename, config_file)) {
7855     (void) fclose(uaf);
7856     return (Z_MISC_FS);
7857 }

7859 if ((err = stat(config_file, &config_st)) != 0) {
7860     zerror(zonename, gettext("could not stat file %s: %s"),
7861     config_file, strerror(errno));
7862     (void) fclose(uaf);
7863     return (Z_MISC_FS);
7864 }
7865 if (config_st.st_mtime >= ua_st.st_mtime) {
7866     (void) fclose(uaf);
7867     return (Z_NO_ENTRY);
7868 }
7869 if ((err = zonecfg_delete_admins(handle, zonename)) == Z_OK) {
7870     changed = B_TRUE;
7871 } else if (err != Z_NO_ENTRY) {
7872     (void) fclose(uaf);
7873     return (err);
7874 }
7875 while ((ua_ptr = fgetuserattr(uaf)) != NULL) {
7876     if (ua_ptr->name[0] == '#' ) {
7877         continue;
7878     }
7879     authlist = kva_match(ua_ptr->attr, USERATTR_AUTHS_KW);
7880     if (authlist != NULL) {
7881         char *cur_auth;
7882         boolean_t first;

7884         first = B_TRUE;
7885         bzero(&adminTAB.zone_admin_auths, MAXAUTHS);
7886         cur_auth = strtok_r(authlist, ",", &lasts);
7887         while (cur_auth != NULL) {
7888             if (is_zone_auth(&cur_auth, zonename,
7889             NULL)) {
7890                 /*
7891                  * Add auths for this zone
7892                  */
7893                 if (first) {
7894                     first = B_FALSE;
7895                 } else {

```

```

7896         (void) strlcat(
7897             admintab.zone_admin_auths,
7898             ",", MAXAUTHS);
7899     }
7900     (void) strlcat(
7901         admintab.zone_admin_auths,
7902         cur_auth, MAXAUTHS);
7903     }
7904     cur_auth = strtok_r(NULL, ",", &lasts);
7905 }
7906 if (!first) {
7907     /*
7908      * Add this right to config file
7909      */
7910     (void) strlcpy(admintab.zone_admin_user,
7911         ua_ptr->name,
7912         sizeof (admintab.zone_admin_user));
7913     err = zonecfg_add_admin(handle,
7914         &admintab, zonename);
7915     if (err != Z_OK) {
7916         (void) fclose(uaf);
7917         return (err);
7918     } else {
7919         changed = B_TRUE;
7920     }
7921 }
7922 }
7923 } /* end-of-while-loop */
7924 (void) fclose(uaf);
7925 return (changed? Z_OK: Z_NO_ENTRY);
7926 }

7928 static void
7929 update_profiles(char *rbac_profs, boolean_t add)
7930 {
7931     char new_profs[MAXPROFS];
7932     char *cur_prof;
7933     boolean_t first = B_TRUE;
7934     boolean_t found = B_FALSE;
7935     char *lasts;

7937     cur_prof = strtok_r(rbac_profs, ",", &lasts);
7938     while (cur_prof != NULL) {
7939         if (strcmp(cur_prof, ZONE_MGMT_PROF) == 0) {
7940             found = B_TRUE;
7941             if (!add) {
7942                 cur_prof = strtok_r(NULL, ",", &lasts);
7943                 continue;
7944             }
7945         }
7946         if (first) {
7947             first = B_FALSE;
7948         } else {
7949             (void) strlcat(new_profs, ",",
7950                 MAXPROFS);
7951         }
7952         (void) strlcat(new_profs, cur_prof,
7953             MAXPROFS);
7954         cur_prof = strtok_r(NULL, ",", &lasts);
7955     }
7956     /*
7957      * Now prepend the Zone Management profile at the beginning
7958      * of the list if it is needed, and append the rest.
7959      * Return the updated list in the original buffer.
7960      */
7961     if (add && !found) {

```

```

7962         first = B_FALSE;
7963         (void) strlcpy(rbac_profs, ZONE_MGMT_PROF, MAXPROFS);
7964     } else {
7965         first = B_TRUE;
7966         rbac_profs[0] = '\0';
7967     }
7968     if (strlen(new_profs) > 0) {
7969         if (!first)
7970             (void) strlcat(rbac_profs, ",", MAXPROFS);
7971         (void) strlcat(rbac_profs, new_profs, MAXPROFS);
7972     }
7973 }

7975 #define MAX_CMD_LEN    1024

7977 static int
7978 do_subproc(char *zonename, char *cmdbuf)
7979 {
7980     char inbuf[MAX_CMD_LEN];
7981     FILE *file;
7982     int status;

7984     file = popen(cmdbuf, "r");
7985     if (file == NULL) {
7986         zerror(zonename, gettext("Could not launch: %s"), cmdbuf);
7987         return (-1);
7988     }

7990     while (fgets(inbuf, sizeof (inbuf), file) != NULL)
7991         (void) fprintf(stderr, "%s", inbuf);
7992     status = pclose(file);

7994     if (WIFSIGNALED(status)) {
7995         zerror(zonename, gettext("%s unexpectedly terminated "
7996             "due to signal %d"),
7997             cmdbuf, WTERMSIG(status));
7998         return (-1);
7999     }
8000     assert(WIFEXITED(status));
8001     return (WEXITSTATUS(status));
8002 }

8004 /*
8005  * This function updates the local /etc/user_attr file to
8006  * correspond to the admin settings that are currently being
8007  * committed. The updates are done via usermod and/or rolemod
8008  * depending on the type of the specified user. It is also
8009  * invoked to remove entries from user_attr corresponding to
8010  * removed admin assignments, using an empty auths string.
8011  *
8012  * Because the removed entries are no longer included in the
8013  * configuration that is being committed, a linked list of
8014  * removed admin entries is maintained to keep track of such
8015  * transactions. The head of the list is stored in the zone_dh_userauths
8016  * element of the handle structure.
8017  */
8018 static int
8019 zonecfg_authorize_user_impl(zone_dochandle_t handle, char *user,
8020     char *auths, char *zonename)
8021 {
8022     char *right;
8023     char old_auths[MAXAUTHS];
8024     char new_auths[MAXAUTHS];
8025     char rbac_profs[MAXPROFS];
8026     char *lasts;
8027     userattr_t *u;

```

```

8028     boolean_t first = B_TRUE;
8029     boolean_t is_zone_admin = B_FALSE;
8030     char user_cmd[] = "/usr/sbin/usermod";
8031     char role_cmd[] = "/usr/sbin/rolemod";
8032     char *auths_cmd = user_cmd; /* either usermod or rolemod */
8033     char *new_auth_start; /* string containing the new auths */
8034     int new_auth_cnt = 0; /* delta of changed authorizations */

8036 /*
8037  * First get the existing authorizations for this user
8038  */

8040     bzero(&old_auths, sizeof (old_auths));
8041     bzero(&new_auths, sizeof (new_auths));
8042     bzero(&rbac_profs, sizeof (rbac_profs));
8043     if ((u = getusernam(user)) != NULL) {
8044         char *current_auths;
8045         char *current_profs;
8046         char *type;

8048         type = kva_match(u->attr, USERATTR_TYPE_KW);
8049         if (type != NULL) {
8050             if (strcmp(type, USERATTR_TYPE_NONADMIN_KW) == 0)
8051                 auths_cmd = role_cmd;
8052         }

8054         current_auths = kva_match(u->attr, USERATTR_AUTHS_KW);
8055         if (current_auths != NULL) {
8056             char *cur_auth;
8057             char *delete_name;
8058             size_t offset;

8060             offset = strlen(ZONE_AUTH_PREFIX);

8062             (void) strcpy(old_auths, current_auths, MAXAUTHS);
8063             cur_auth = strtok_r(current_auths, ",", &lasts);

8065             /*
8066              * Next, remove any existing authorizations
8067              * for this zone, and determine if the
8068              * user still needs the Zone Management Profile.
8069              */
8070             if (is_renaming(handle))
8071                 delete_name = handle->zone_dh_delete_name;
8072             else
8073                 delete_name = NULL;
8074             while (cur_auth != NULL) {
8075                 if (!is_zone_auth(&cur_auth, zonename,
8076                     delete_name)) {
8077                     if (first) {
8078                         first = B_FALSE;
8079                     } else {
8080                         (void) strcat(new_auths, ",",
8081                             MAXAUTHS);
8082                     }
8083                     (void) strcat(new_auths, cur_auth,
8084                         MAXAUTHS);
8085                     /*
8086                      * If the user has authorizations
8087                      * for other zones, then set a
8088                      * flag indicate that the Zone
8089                      * Management profile should be
8090                      * preserved in user_attr.
8091                      */
8092                     if (strcmp(cur_auth,
8093                         ZONE_AUTH_PREFIX, offset) == 0)

```

```

8094         is_zone_admin = B_TRUE;
8095     } else {
8096         new_auth_cnt++;
8097     }
8098     cur_auth = strtok_r(NULL, ",", &lasts);
8099     }
8100     }
8101     current_profs = kva_match(u->attr, USERATTR_PROFILES_KW);
8102     if (current_profs != NULL) {
8103         (void) strcpy(rbac_profs, current_profs, MAXPROFS);
8104     }
8105     free_userattr(u);
8106 }
8107 /*
8108  * The following is done to avoid revisiting the
8109  * user_attr entry for this user
8110  */
8111     (void) zonecfg_remove_userauths(handle, user, "", B_FALSE);

8113 /*
8114  * Convert each right into a properly formatted authorization
8115  */
8116     new_auth_start = new_auths + strlen(new_auths);
8117     if (!first)
8118         new_auth_start++;
8119     right = strtok_r(auths, ",", &lasts);
8120     while (right != NULL) {
8121         char auth[MAXAUTHS];

8123         (void) snprintf(auth, MAXAUTHS, "%s%s/%s",
8124             ZONE_AUTH_PREFIX, right, zonename);
8125         if (first) {
8126             first = B_FALSE;
8127         } else {
8128             (void) strcat(new_auths, ",", MAXAUTHS);
8129         }
8130         (void) strcat(new_auths, auth, MAXAUTHS);
8131         is_zone_admin = B_TRUE;
8132         new_auth_cnt--;
8133         right = strtok_r(NULL, ",", &lasts);
8134     }

8136 /*
8137  * Need to update the authorizations in user_attr unless
8138  * the number of old and new authorizations is unchanged
8139  * and the new auths are a substrings of the old auths.
8140  *
8141  * If the user's previous authorizations have changed
8142  * execute the usermod program to update them in user_attr.
8143  */
8144     if ((new_auth_cnt != 0) ||
8145         (strcmp(old_auths, new_auth_start) == NULL)) {
8146         char *cmdbuf;
8147         size_t cmd_len;

8149         update_profiles(rbac_profs, is_zone_admin);
8150         cmd_len = snprintf(NULL, 0, "%s -A \"%s\" -P \"%s\" %s",
8151             auths_cmd, new_auths, rbac_profs, user) + 1;
8152         if ((cmdbuf = malloc(cmd_len)) == NULL) {
8153             return (Z_NOMEM);
8154         }
8155         (void) snprintf(cmdbuf, cmd_len, "%s -A \"%s\" -P \"%s\" %s",
8156             auths_cmd, new_auths, rbac_profs, user);
8157         if (do_subproc(zonename, cmdbuf) != 0) {
8158             free(cmdbuf);
8159             return (Z_SYSTEM);

```

```

8160     }
8161     free(cmdbuf);
8162 }

8164     return (Z_OK);
8165 }

8167 int
8168 zonecfg_authorize_users(zone_dochandle_t handle, char *zonename)
8169 {
8170     xmlNodePtr cur;
8171     int err;
8172     char user[MAXUSERNAME];
8173     char auths[MAXAUTHS];

8175     if ((err = operation_prep(handle)) != Z_OK)
8176         return (err);

8178     cur = handle->zone_dh_cur;
8179     for (cur = cur->xmlChildrenNode; cur != NULL; cur = cur->next) {
8180         if (xmlStrcmp(cur->name, DTD_ELEM_ADMIN))
8181             continue;
8182         if (fetchprop(cur, DTD_ATTR_USER, user,
8183             sizeof (user)) != Z_OK)
8184             continue;
8185         if (fetchprop(cur, DTD_ATTR_AUTHS, auths,
8186             sizeof (auths)) != Z_OK)
8187             continue;
8188         if (zonecfg_authorize_user_impl(handle, user, auths, zonename)
8189             != Z_OK)
8190             return (Z_SYSTEM);
8191     }
8192     (void) zonecfg_remove_userauths(handle, "", "", B_TRUE);

8194     return (Z_OK);
8195 }

8197 int
8198 zonecfg_deauthorize_user(zone_dochandle_t handle, char *user, char *zonename)
8199 {
8200     return (zonecfg_authorize_user_impl(handle, user, "", zonename));
8201 }

8203 int
8204 zonecfg_deauthorize_users(zone_dochandle_t handle, char *zonename)
8205 {
8206     xmlNodePtr cur;
8207     int err;
8208     char user[MAXUSERNAME];

8210     if ((err = operation_prep(handle)) != Z_OK)
8211         return (err);

8213     cur = handle->zone_dh_cur;
8214     for (cur = cur->xmlChildrenNode; cur != NULL; cur = cur->next) {
8215         if (xmlStrcmp(cur->name, DTD_ELEM_ADMIN))
8216             continue;
8217         if (fetchprop(cur, DTD_ATTR_USER, user,
8218             sizeof (user)) != Z_OK)
8219             continue;
8220         if ((err = zonecfg_deauthorize_user(handle, user,
8221             zonename)) != Z_OK)
8222             return (err);
8223     }
8224     return (Z_OK);
8225 }

```

```

8227 int
8228 zonecfg_insert_userauths(zone_dochandle_t handle, char *user, char *zonename)
8229 {
8230     zone_userauths_t *new, **prev, *next;

8232     prev = &handle->zone_dh_userauths;
8233     next = *prev;
8234     while (next) {
8235         if ((strncmp(next->user, user, MAXUSERNAME) == 0) &&
8236             (strncmp(next->zonename, zonename,
8237                 ZONENAME_MAX) == 0)) {
8238             /*
8239              * user is already in list
8240              * which isn't supposed to happen!
8241              */
8242             return (Z_OK);
8243         }
8244         prev = &next->next;
8245         next = *prev;
8246     }
8247     new = (zone_userauths_t *)malloc(sizeof (zone_userauths_t));
8248     if (new == NULL)
8249         return (Z_NOMEM);

8251     (void) strncpy(new->user, user, sizeof (new->user));
8252     (void) strncpy(new->zonename, zonename, sizeof (new->zonename));
8253     new->next = NULL;
8254     *prev = new;
8255     return (Z_OK);
8256 }

8258 int
8259 zonecfg_remove_userauths(zone_dochandle_t handle, char *user, char *zonename,
8260     boolean_t deauthorize)
8261 {
8262     zone_userauths_t *new, **prev, *next;

8264     prev = &handle->zone_dh_userauths;
8265     next = *prev;

8267     while (next) {
8268         if ((strlen(user) == 0 ||
8269             strncmp(next->user, user, MAXUSERNAME) == 0) &&
8270             (strlen(zonename) == 0 ||
8271                 (strncmp(next->zonename, zonename, ZONENAME_MAX) == 0))) {
8272             new = next;
8273             *prev = next->next;
8274             next = *prev;
8275             if (deauthorize)
8276                 (void) zonecfg_deauthorize_user(handle,
8277                     new->user, new->zonename);
8278             free(new);
8279             continue;
8280         }
8281         prev = &next->next;
8282         next = *prev;
8283     }
8284     return (Z_OK);
8285 }

```



```

*****
5906 Thu Jun 30 22:02:35 2016
new/usr/src/lib/libzonecfg/common/mapfile-vers
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
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 (c) 2006, 2010, Oracle and/or its affiliates. All rights reserved.
23 #
24 #
25 #
26 # MAPFILE HEADER START
27 #
28 # WARNING: STOP NOW. DO NOT MODIFY THIS FILE.
29 # Object versioning must comply with the rules detailed in
30 #
31 #     usr/src/lib/README.mapfiles
32 #
33 # You should not be making modifications here until you've read the most current
34 # copy of that file. If you need help, contact a gatekeeper for guidance.
35 #
36 # MAPFILE HEADER END
37 #
38 #
39 $mapfile_version 2
40 #
41 SYMBOL_VERSION SUNWprivate_1.1 {
42     global:
43         endzoneent;
44         getzoneent;
45         getzoneent_private;
46         putzoneent;
47         setzoneent;
48         zonecfg_access;
49         zonecfg_add_attr;
50         zonecfg_add_dev;
51         zonecfg_add_ds;
52         zonecfg_add_filesystem;
53         zonecfg_add_fs_option;
54         zonecfg_add_admin;
55         zonecfg_add_nwif;
56         zonecfg_add_pkg;
57         zonecfg_add_pset;
58         zonecfg_add_rctl;

```

```

59     zonecfg_add_rctl_value;
60     zonecfg_add_scratch;
61     zonecfg_add_secflags;
62 #endif /* ! codereview */
63     zonecfg_aliased_rctl_ok;
64     zonecfg_apply_rctls;
65     zonecfg_attach_manifest;
66     zonecfg_authorize_users;
67     zonecfg_bind_pool;
68     zonecfg_bind_tmp_pool;
69     zonecfg_call_zoneadmd;
70     zonecfg_check_handle;
71     zonecfg_close_scratch;
72     zonecfg_construct_rctlblk;
73     zonecfg_create_snapshot;
74     zonecfg_deauthorize_user;
75     zonecfg_deauthorize_users;
76     zonecfg_default_brand;
77     zonecfg_default_privset;
78     zonecfg_delete_admin;
79     zonecfg_delete_admins;
80     zonecfg_delete_attr;
81     zonecfg_delete_dev;
82     zonecfg_delete_ds;
83     zonecfg_delete_filesystem;
84     zonecfg_delete_mcap;
85     zonecfg_delete_nwif;
86     zonecfg_delete_pset;
87     zonecfg_delete_rctl;
88     zonecfg_delete_scratch;
89     zonecfg_delete_secflags;
90 #endif /* ! codereview */
91     zonecfg_del_all_resources;
92     zonecfg_destroy;
93     zonecfg_destroy_snapshot;
94     zonecfg_destroy_tmp_pool;
95     zonecfg_detached;
96     zonecfg_detach_save;
97     zonecfg_devperms_apply;
98     zonecfg_dev_manifest;
99     zonecfg_enable_rcapd;
100    zonecfg_endadminent;
101    zonecfg_endattrent;
102    zonecfg_enddvent;
103    zonecfg_enddevperment;
104    zonecfg_enddsent;
105    zonecfg_endfsent;
106    zonecfg_endnwifent;
107    zonecfg_endrctlent;
108    zonecfg_find_mounts;
109    zonecfg_find_scratch;
110    zonecfg_fini_handle;
111    zonecfg_free_fs_option_list;
112    zonecfg_free_rctl_value_list;
113    zonecfg_get_aliased_rctl;
114    zonecfg_get_attach_handle;
115    zonecfg_get_attr_boolean;
116    zonecfg_getadminent;
117    zonecfg_getattrent;
118    zonecfg_get_attr_int;
119    zonecfg_get_attr_string;
120    zonecfg_get_attr_uint;
121    zonecfg_get_autoboot;
122    zonecfg_get_bootargs;
123    zonecfg_get_brand;
124    zonecfg_get_dflt_sched_class;

```

```

125     zonecfg_getdevent;
126     zonecfg_getdevperment;
127     zonecfg_getdsent;
128     zonecfg_getfsent;
129     zonecfg_get_fs_allowed;
130     zonecfg_get_handle;
131     zonecfg_get_hostid;
132     zonecfg_get_ipstype;
133     zonecfg_get_limitpriv;
134     zonecfg_getmcapent;
135     zonecfg_get_name;
136     zonecfg_get_name_by_uuid;
137     zonecfg_getnwifent;
138     zonecfg_getpkgdata;
139     zonecfg_get_pool;
140     zonecfg_get_poolname;
141     zonecfg_get_privset;
142     zonecfg_getpsetent;
143     zonecfg_getrctlent;
144     zonecfg_getsecflgsent;
145 #endif /* ! codereview */
146     zonecfg_get_root;
147     zonecfg_get_sched_class;
148     zonecfg_get_scratch;
149     zonecfg_get_snapshot_handle;
150     zonecfg_get_template_handle;
151     zonecfg_get_uuid;
152     zonecfg_get_xml_handle;
153     zonecfg_get_zonename;
154     zonecfg_grab_lock_file;
155     zonecfg_ifname_exists;
156     zonecfg_in_alt_root;
157     zonecfg_insert_userauths;
158     zonecfg_init_handle;
159     zonecfg_init_lock_file;
160     zonecfg_is_rctl;
161     zonecfg_is_scratch;
162     zonecfg_lock_file_held;
163     zonecfg_lock_scratch;
164     zonecfg_lookup_admin;
165     zonecfg_lookup_attr;
166     zonecfg_lookup_dev;
167     zonecfg_lookup_ds;
168     zonecfg_lookup_filesystem;
169     zonecfg_lookup_mcap;
170     zonecfg_lookup_nwif;
171     zonecfg_lookup_pset;
172     zonecfg_lookup_rctl;
173     zonecfg_lookup_secflgs;
174 #endif /* ! codereview */
175     zonecfg_modify_admin;
176     zonecfg_modify_attr;
177     zonecfg_modify_dev;
178     zonecfg_modify_ds;
179     zonecfg_modify_filesystem;
180     zonecfg_modify_mcap;
181     zonecfg_modify_nwif;
182     zonecfg_modify_pset;
183     zonecfg_modify_rctl;
184     zonecfg_modify_secflgs;
185 #endif /* ! codereview */
186     zonecfg_notify_bind;
187     zonecfg_notify_critical_abort;
188     zonecfg_notify_critical_enter;
189     zonecfg_notify_critical_exit;
190     zonecfg_notify_unbind;

```

```

191     zonecfg_num_resources;
192     zonecfg_open_scratch;
193     zonecfg_ping_zoneadmd;
194     zonecfg_release_lock_file;
195     zonecfg_remove_fs_option;
196     zonecfg_remove_rctl_value;
197     zonecfg_remove_userauths;
198     zonecfg_reverse_scratch;
199     zonecfg_rm_aliased_rctl;
200     zonecfg_rm_detached;
201     zonecfg_same_net_address;
202     zonecfg_save;
203     zonecfg_setadminent;
204     zonecfg_setattrent;
205     zonecfg_set_aliased_rctl;
206     zonecfg_set_autoboot;
207     zonecfg_set_bootargs;
208     zonecfg_set_brand;
209     zonecfg_setdevent;
210     zonecfg_setdevperment;
211     zonecfg_setdsent;
212     zonecfg_setfsent;
213     zonecfg_set_fs_allowed;
214     zonecfg_set_hostid;
215     zonecfg_set_ipstype;
216     zonecfg_set_limitpriv;
217     zonecfg_set_name;
218     zonecfg_setnwifent;
219     zonecfg_set_pool;
220     zonecfg_setrctlent;
221     zonecfg_set_root;
222     zonecfg_set_sched;
223     zonecfg_set_swinv;
224     zonecfg_set_zonename;
225     zonecfg_strerror;
226     zonecfg_str_to_bytes;
227     zonecfg_update_userauths;
228     zonecfg_validate_zonename;
229     zonecfg_valid_auths;
230     zonecfg_valid_alias_limit;
231     zonecfg_valid_fs_type;
232     zonecfg_valid_importance;
233     zonecfg_valid_memlimit;
234     zonecfg_valid_ncpus;
235     zonecfg_valid_net_address;
236     zonecfg_valid_rctl;
237     zonecfg_valid_rctlblk;
238     zonecfg_valid_rctlname;
239     zonecfg_verify_save;
240     zonecfg_warn_poold;
241     zone_get_brand;
242     zone_get_devroot;
243     zone_get_id;
244     zone_get_rootpath;
245     zone_get_state;
246     zone_get_zonename;
247     zone_set_state;
248     zone_state_str;
249     local:
250     *;
251 };

```

```

*****
4399 Thu Jun 30 22:02:36 2016
new/usr/src/lib/libzonecfg/dtd/zonecfg.dtd.1
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
1 <?xml version='1.0' encoding='UTF-8' ?>
3 <!--
4 CDDL HEADER START

6 The contents of this file are subject to the terms of the
7 Common Development and Distribution License (the "License").
8 You may not use this file except in compliance with the License.

10 You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
11 or http://www.opensolaris.org/os/licensing.
12 See the License for the specific language governing permissions
13 and limitations under the License.

15 When distributing Covered Code, include this CDDL HEADER in each
16 file and include the License file at usr/src/OPENSOLARIS.LICENSE.
17 If applicable, add the following below this CDDL HEADER, with the
18 fields enclosed by brackets "[]" replaced with your own identifying
19 information: Portions Copyright [yyyy] [name of copyright owner]

21 CDDL HEADER END

23 Copyright (c) 2003, 2010, Oracle and/or its affiliates. All rights reserved.

25 -->

27 <!--Element Definitions-->
29 <!ELEMENT fsoption EMPTY>
30 <!ATTLIST fsoption name CDATA #REQUIRED>

32 <!ELEMENT filesystem (fsoption)* >
34 <!ATTLIST filesystem special CDATA #REQUIRED
35 raw CDATA ""
36 directory CDATA #REQUIRED
37 type CDATA #REQUIRED>

39 <!--
40 The "inherited-pkg-dir" element may no longer be specified in a
41 configuration, however we retain the definition to aid with migration
42 of existing configurations.
43 -->

45 <!ELEMENT inherited-pkg-dir EMPTY>
47 <!ATTLIST inherited-pkg-dir directory CDATA #REQUIRED>

49 <!ELEMENT network EMPTY>
51 <!ATTLIST network address CDATA ""
52 allowed-address CDATA ""
53 defrouter CDATA ""
54 physical CDATA #REQUIRED>

56 <!ELEMENT device EMPTY>
58 <!ATTLIST device match CDATA #REQUIRED>

```

```

60 <!--
61 Historically, the deleted-device element denoted a used-to-be
62 device element. This was used to keep track of device elements
63 deleted or modified by the user, and to cleanse /dev of such
64 entries at next zone boot.

66 With the ability to now configure devices dynamically, this
67 requirement no longer exists, but this element MUST remain in
68 perpetuity, since it is possible that an upgraded zone could
69 carry a deleted-device element, and would therefore fail XML
70 validation if removed
71 -->
72 <!ELEMENT deleted-device EMPTY>

74 <!ATTLIST deleted-device match CDATA #REQUIRED>

76 <!ELEMENT rctl-value EMPTY>
78 <!ATTLIST rctl-value priv CDATA #REQUIRED
79 limit CDATA #REQUIRED
80 action CDATA #REQUIRED>

82 <!ELEMENT rctl (rctl-value)*>
84 <!ATTLIST rctl name CDATA #REQUIRED>

86 <!ELEMENT attr EMPTY>
88 <!ATTLIST attr name CDATA #REQUIRED
89 type (boolean | int | string | uint)
90 #REQUIRED
91 value CDATA #REQUIRED>

93 <!ELEMENT dataset EMPTY>
95 <!ATTLIST dataset name CDATA #REQUIRED>

97 <!ELEMENT package EMPTY>
99 <!ATTLIST package name CDATA #REQUIRED
100 version CDATA #REQUIRED>

102 <!ELEMENT obsoletes EMPTY>
103 <!ATTLIST obsoletes id CDATA #REQUIRED>

105 <!ELEMENT incompatible EMPTY>
106 <!ATTLIST incompatible id CDATA #REQUIRED>

108 <!ELEMENT patch (obsoletes | incompatible)* >
110 <!ATTLIST patch id CDATA #REQUIRED>
112 <!ELEMENT dev-perm EMPTY>
114 <!ATTLIST dev-perm name CDATA #REQUIRED
115 uid CDATA #REQUIRED
116 gid CDATA #REQUIRED
117 mode CDATA #REQUIRED
118 acl CDATA #REQUIRED>

120 <!--
121 The tmp_pool element is separate from the pset element so that
122 we can track the importance value at the pool level, where it
123 belongs, instead of at the pset level. Once we have msets this
124 will be important since tmp psets and tmp msets will share a common

```

```
125         pool-level importance.
126 -->
127 <!ELEMENT tmp_pool      EMPTY>

129 <!ATTLIST tmp_pool      importance      CDATA #REQUIRED>

131 <!ELEMENT pset          EMPTY>

133 <!ATTLIST pset          ncpu_min        CDATA #REQUIRED
134                       ncpu_max        CDATA #REQUIRED>

136 <!ELEMENT mcap          EMPTY>

138 <!ATTLIST mcap          physcap         CDATA #REQUIRED>

140 <!ELEMENT admin         EMPTY>

142 <!ATTLIST admin         user           CDATA #REQUIRED
143                       auths          CDATA #REQUIRED>

145 <!ELEMENT security-flags EMPTY>

147 <!ATTLIST security-flags default        CDATA ""
148                       lower            CDATA ""
149                       upper            CDATA "">

151 #endif /* ! codereview */
152 <!ELEMENT zone          (filesystem | inherited-pkg-dir | network | device |
153                       deleted-device | rctl | attr | dataset | package |
154                       patch | dev-perm | tmp_pool | pset |
155                       mcap | admin | security-flags)*>
156 mcap | admin)*>

157 <!ATTLIST zone          name            CDATA #REQUIRED
158                       zonepath         CDATA #REQUIRED
159                       autoboot         (true | false) #REQUIRED
160                       ip-type          CDATA ""
161                       hostid           CDATA ""
162                       pool             CDATA ""
163                       limitpriv        CDATA ""
164                       bootargs         CDATA ""
165                       brand            CDATA ""
166                       scheduling-class  CDATA ""
167                       fs-allowed        CDATA ""
168                       version           NMTOKEN #FIXED '1'>
```

```

*****
13296 Thu Jun 30 22:02:37 2016
new/usr/src/man/man1/Makefile
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
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
9 # at http://www.illumos.org/license/CDDL.
10 #
12 #
13 # Copyright 2011, Richard Lowe
14 # Copyright 2015 Nexenta Systems, Inc. All rights reserved.
15 # Copyright 2014 Garrett D'Amore <garrett@damore.org>
16 #
18 include      $(SRC)/Makefile.master
20 MANSECT=     1
22 MANFILES=    acctcom.1
23              adb.1
24              addbib.1
25              alias.1
26              allocate.1
27              amt.1
28              appcert.1
29              apptrace.1
30              apropos.1
31              ar.1
32              arch.1
33              asa.1
34              at.1
35              atq.1
36              atrm.1
37              audioconvert.1
38              audiocvt.1
39              audioplay.1
40              audiorecord.1
41              audiotest.1
42              auths.1
43              awk.1
44              banner.1
45              basename.1
46              bc.1
47              bdiff.1
48              break.1
49              builtin.1
50              cal.1
51              calendar.1
52              cancel.1
53              cat.1
54              cd.1
55              cdrw.1
56              checknr.1
57              chgrp.1
58              chkey.1

```

```

59              chmod.1
60              chown.1
61              ckdate.1
62              ckgid.1
63              ckint.1
64              ckitem.1
65              ckkeywd.1
66              ckpath.1
67              ckrange.1
68              ckstr.1
69              cksum.1
70              cktime.1
71              ckuid.1
72              ckyorn.1
73              clear.1
74              cmp.1
75              col.1
76              comm.1
77              command.1
78              compress.1
79              cp.1
80              cpio.1
81              cputrack.1
82              crle.1
83              crontab.1
84              crypt.1
85              csh.1
86              csplit.1
87              ctags.1
88              ctrun.1
89              ctstat.1
90              ctwatch.1
91              cut.1
92              date.1
93              dc.1
94              deallocate.1
95              deroff.1
96              dhcpinfo.1
97              diff.1
98              diff3.1
99              diffmk.1
100             digest.1
101             dircmp.1
102             dis.1
103             disown.1
104             dispgid.1
105             dispuid.1
106             dos2unix.1
107             download.1
108             dpost.1
109             du.1
110             dump.1
111             dumpcs.1
112             echo.1
113             ed.1
114             egrep.1
115             eject.1
116             elfdump.1
117             elfedit.1
118             elfsign.1
119             elfwrap.1
120             enable.1
121             encrypt.1
122             enhance.1
123             env.1
124             eqn.1

```

```

125      exec.1
126      exit.1
127      expand.1
128      expr.1
129      exstr.1
130      factor.1
131      fdformat.1
132      fgrep.1
133      file.1
134      filesync.1
135      find.1
136      finger.1
137      fmt.1
138      fmtmsg.1
139      fold.1
140      ftp.1
141      gcore.1
142      gencat.1
143      genmsg.1
144      getconf.1
145      getfacl.1
146      getlabel.1
147      getopt.1
148      getoptcvt.1
149      getopts.1
150      gettext.1
151      gettxt.1
152      getzonepath.1
153      glob.1
154      gprof.1
155      grep.1
156      groups.1
157      hash.1
158      head.1
159      history.1
160      hostid.1
161      hostname.1
162      iconv.1
163      indxbib.1
164      Intro.1
165      ipcrm.1
166      ipcs.1
167      isainfo.1
168      isalist.1
169      jobs.1
170      join.1
171      kbd.1
172      kdestroy.1
173      keylogin.1
174      keylogout.1
175      kill.1
176      kinit.1
177      klist.1
178      kmdb.1
179      kmfcfg.1
180      kpasswd.1
181      krb5-config.1
182      ksh93.1
183      ktutil.1
184      kvmstat.1
185      lari.1
186      last.1
187      lastcomm.1
188      ld.1
189      ldap.1
190      ldapdelete.1

```

```

191      ldaplist.1
192      ldapmodify.1
193      ldapmodrdn.1
194      ldapsearch.1
195      ldd.1
196      ld.so.1.1
197      let.1
198      lex.1
199      lgrpinfo.1
200      limit.1
201      line.1
202      list_devices.1
203      listusers.1
204      ln.1
205      loadkeys.1
206      locale.1
207      localedef.1
208      logger.1
209      login.1
210      logname.1
211      logout.1
212      look.1
213      lookbib.1
214      lorder.1
215      lp.1
216      lpstat.1
217      ls.1
218      m4.1
219      mac.1
220      mach.1
221      madv.so.1.1
222      mail.1
223      mailcompat.1
224      mailq.1
225      mailstats.1
226      mailx.1
227      make.1
228      makekey.1
229      man.1
230      mandoc.1
231      mconnect.1
232      mcs.1
233      mdb.1
234      msg.1
235      mkdir.1
236      mkmsgs.1
237      mktemp.1
238      moe.1
239      more.1
240      mpss.so.1.1
241      msgcc.1
242      msgcpp.1
243      msgcvt.1
244      msgfmt.1
245      msggen.1
246      msgget.1
247      mt.1
248      mv.1
249      nawk.1
250      nc.1
251      nca.1
252      ncab2clf.1
253      ncakmod.1
254      newform.1
255      newgrp.1
256      news.1

```

```

257 newtask.1 \
258 nice.1 \
259 nl.1 \
260 nm.1 \
261 nohup.1 \
262 nroff.1 \
263 od.1 \
264 on.1 \
265 optisa.1 \
266 pack.1 \
267 pagesize.1 \
268 pargs.1 \
269 passwd.1 \
270 paste.1 \
271 pathchk.1 \
272 pax.1 \
273 pfexec.1 \
274 pg.1 \
275 pgrep.1 \
276 pkginfo.1 \
277 pkgmk.1 \
278 pkgparam.1 \
279 pkgproto.1 \
280 pkgtrans.1 \
281 pktool.1 \
282 plabel.1 \
283 plgrp.1 \
284 plimit.1 \
285 pmadvise.1 \
286 pmap.1 \
287 postio.1 \
288 postprint.1 \
289 postreverse.1 \
290 ppgsz.1 \
291 ppriv.1 \
292 pr.1 \
293 praliases.1 \
294 prctl.1 \
295 preap.1 \
296 prex.1 \
297 print.1 \
298 printf.1 \
299 priocntl.1 \
300 proc.1 \
301 prof.1 \
302 profiles.1 \
303 projects.1 \
304 ps.1 \
305 psecflags.1 \
306 #endif /* ! codereview */
307 ptree.1 \
308 pvs.1 \
309 pwd.1 \
310 ranlib.1 \
311 rcapstat.1 \
312 rcp.1 \
313 rdist.1 \
314 read.1 \
315 readonly.1 \
316 refer.1 \
317 regcmp.1 \
318 renice.1 \
319 rev.1 \
320 rlogin.1 \
321 rm.1 \
322 rmformat.1 \

```

```

323 rmount.1 \
324 roffbib.1 \
325 roles.1 \
326 rpcgen.1 \
327 rsh.1 \
328 runat.1 \
329 rup.1 \
330 ruptime.1 \
331 rusers.1 \
332 rwho.1 \
333 sar.1 \
334 scp.sunssh.1 \
335 script.1 \
336 sdiff.1 \
337 sed.1 \
338 set.1 \
339 setfacl.1 \
340 setlabel.1 \
341 setpgrp.1 \
342 sftp.sunssh.1 \
343 shcomp.1 \
344 shell_builtins.1 \
345 shift.1 \
346 size.1 \
347 sleep.1 \
348 smbutil.1 \
349 soelim.1 \
350 sort.1 \
351 sortbib.1 \
352 sotruss.1 \
353 spell.1 \
354 split.1 \
355 srchtxt.1 \
356 ssh.sunssh.1 \
357 ssh-add.sunssh.1 \
358 ssh-agent.sunssh.1 \
359 ssh-http-proxy-connect.sunssh.1 \
360 ssh-keygen.sunssh.1 \
361 ssh-keyscan.sunssh.1 \
362 ssh-socks5-proxy-connect.sunssh.1 \
363 strchg.1 \
364 strings.1 \
365 strip.1 \
366 stty.1 \
367 sum.1 \
368 suspend.1 \
369 svcprop.1 \
370 svcs.1 \
371 symorder.1 \
372 sysV-make.1 \
373 sys-suspend.1 \
374 tabs.1 \
375 tail.1 \
376 talk.1 \
377 tar.1 \
378 tbl.1 \
379 tcopy.1 \
380 tee.1 \
381 telnet.1 \
382 test.1 \
383 tftp.1 \
384 time.1 \
385 times.1 \
386 timex.1 \
387 tip.1 \
388 tnfdump.1 \

```

```

389         tnfxtract.1  \
390         touch.1      \
391         tput.1       \
392         tr.1         \
393         trap.1      \
394         troff.1     \
395         true.1      \
396         truss.1     \
397         tsort.1     \
398         tty.1       \
399         type.1      \
400         typeset.1  \
401         ul.1        \
402         umask.1     \
403         uname.1    \
404         unifdef.1  \
405         uniq.1     \
406         units.1    \
407         unix2dos.1 \
408         uptime.1   \
409         uuidgen.1  \
410         vacation.1 \
411         vgrind.1   \
412         volcheck.1 \
413         volrmmount.1 \
414         w.1        \
415         wait.1     \
416         wc.1       \
417         which.1    \
418         who.1      \
419         whocalls.1 \
420         whois.1    \
421         write.1    \
422         xargs.1    \
423         xgettext.1 \
424         xstr.1     \
425         yacc.1     \
426         yes.1      \
427         ypcat.1    \
428         ypmatch.1  \
429         yppasswd.1 \
430         ypwhich.1  \
431         zlogin.1   \
432         zonename.1 \
433
434 MANLINKS=  batch.1  \
435            bg.1     \
436            case.1   \
437            chdir.1  \
438            checkeq.1 \
439            continue.1 \
440            decrypt.1 \
441            dirname.1 \
442            dirs.1   \
443            disable.1 \
444            dmake.1  \
445            dumpkeys.1 \
446            edit.1   \
447            errange.1 \
448            errdate.1 \
449            errgid.1 \
450            errint.1 \
451            erritem.1 \
452            errpath.1 \
453            errstr.1 \
454            errtime.1 \

```

```

455         erruid.1  \
456         erryorn.1 \
457         eval.1   \
458         export.1 \
459         false.1  \
460         fc.1     \
461         fg.1     \
462         for.1    \
463         foreach.1 \
464         function.1 \
465         goto.1   \
466         hashcheck.1 \
467         hashmake.1 \
468         hashstat.1 \
469         helpdate.1 \
470         helpgid.1  \
471         helpint.1 \
472         helpitem.1 \
473         helppath.1 \
474         helprange.1 \
475         helpstr.1  \
476         helptime.1 \
477         helpuid.1  \
478         helpyorn.1 \
479         hist.1     \
480         if.1       \
481         intro.1    \
482         jsh.1      \
483         ksh.1      \
484         ldapadd.1  \
485         neqn.1     \
486         notify.1   \
487         onintr.1   \
488         page.1     \
489         pcat.1     \
490         pcred.1    \
491         pfcsh.1    \
492         pfiles.1   \
493         pfksh.1    \
494         pflags.1   \
495         pfsh.1     \
496         pkill.1    \
497         pldd.1     \
498         popd.1     \
499         prun.1     \
500         psig.1     \
501         pstack.1   \
502         pstop.1    \
503         ptime.1    \
504         pushd.1    \
505         pwait.1    \
506         pwdx.1     \
507         red.1      \
508         rehash.1   \
509         remote_shell.1 \
510         remsh.1    \
511         repeat.1   \
512         return.1   \
513         rksh.1     \
514         rksh93.1   \
515         rmail.1    \
516         rmdir.1    \
517         rmmount.1  \
518         scp.1      \
519         select.1   \
520         setenv.1   \

```



```

521         settime.1      \
522         sftp.1         \
523         sh.1           \
524         snca.1         \
525         source.1      \
526         spellin.1     \
527         ssh.1          \
528         ssh-add.1      \
529         ssh-agent.1    \
530         ssh-http-proxy-connect.1 \
531         ssh-keygen.1   \
532         ssh-keyscan.1  \
533         ssh-socks5-proxy-connect.1 \
534         stop.1         \
535         strconf.1      \
536         switch.1       \
537         ulimit.1       \
538         unalias.1      \
539         uncompress.1   \
540         unexpand.1     \
541         unhash.1       \
542         unlimit.1      \
543         unpack.1       \
544         unset.1        \
545         unsetenv.1     \
546         until.1        \
547         valdate.1      \
548         valgid.1       \
549         valint.1        \
550         valpath.1      \
551         valrange.1     \
552         valstr.1       \
553         valtime.1      \
554         valuid.1       \
555         valyorn.1      \
556         vedit.1        \
557         whatis.1       \
558         whence.1       \
559         while.1        \
560         zcat.1         \
562 intro.1             := LINKSRC = Intro.1
564 whatis.1           := LINKSRC = apropos.1
566 unalias.1          := LINKSRC = alias.1
568 batch.1             := LINKSRC = at.1
570 dirname.1          := LINKSRC = basename.1
572 continue.1         := LINKSRC = break.1
574 chdir.1            := LINKSRC = cd.1
575 dirs.1             := LINKSRC = cd.1
576 popd.1             := LINKSRC = cd.1
577 pushd.1            := LINKSRC = cd.1
579 errdate.1          := LINKSRC = ckdate.1
580 helpdate.1         := LINKSRC = ckdate.1
581 valdate.1          := LINKSRC = ckdate.1
583 errgid.1           := LINKSRC = ckgid.1
584 helpgid.1          := LINKSRC = ckgid.1
585 valgid.1           := LINKSRC = ckgid.1

```

```

587 errint.1           := LINKSRC = ckint.1
588 helpint.1          := LINKSRC = ckint.1
589 valint.1           := LINKSRC = ckint.1
591 erritem.1          := LINKSRC = ckitem.1
592 helpitem.1         := LINKSRC = ckitem.1
594 errpath.1          := LINKSRC = ckpath.1
595 helppath.1         := LINKSRC = ckpath.1
596 valpath.1          := LINKSRC = ckpath.1
598 errrange.1         := LINKSRC = ckrange.1
599 helprange.1        := LINKSRC = ckrange.1
600 valrange.1         := LINKSRC = ckrange.1
602 errstr.1           := LINKSRC = ckstr.1
603 helpstr.1          := LINKSRC = ckstr.1
604 valstr.1           := LINKSRC = ckstr.1
606 errtime.1          := LINKSRC = cktime.1
607 helptime.1         := LINKSRC = cktime.1
608 valtime.1          := LINKSRC = cktime.1
610 erruid.1           := LINKSRC = ckuid.1
611 helpuid.1          := LINKSRC = ckuid.1
612 valuid.1           := LINKSRC = ckuid.1
614 erryorn.1          := LINKSRC = ckyorn.1
615 helpyorn.1         := LINKSRC = ckyorn.1
616 valyorn.1          := LINKSRC = ckyorn.1
618 uncompress.1       := LINKSRC = compress.1
619 zcat.1             := LINKSRC = compress.1
621 red.1              := LINKSRC = ed.1
623 disable.1         := LINKSRC = enable.1
625 decrypt.1         := LINKSRC = encrypt.1
627 dmake.1           := LINKSRC = make.1
629 checkeq.1          := LINKSRC = eqn.1
630 neqn.1             := LINKSRC = eqn.1
632 eval.1            := LINKSRC = exec.1
633 source.1          := LINKSRC = exec.1
635 goto.1            := LINKSRC = exit.1
636 return.1          := LINKSRC = exit.1
638 unexpand.1        := LINKSRC = expand.1
640 hashstat.1         := LINKSRC = hash.1
641 rehash.1           := LINKSRC = hash.1
642 unhash.1           := LINKSRC = hash.1
644 fc.1              := LINKSRC = history.1
645 hist.1            := LINKSRC = history.1
647 bg.1              := LINKSRC = jobs.1
648 fg.1              := LINKSRC = jobs.1
649 notify.1          := LINKSRC = jobs.1
650 stop.1            := LINKSRC = jobs.1
652 jsh.1             := LINKSRC = ksh93.1

```

```

653 ksh.1      := LINKSRC = ksh93.1
654 rksh.1     := LINKSRC = ksh93.1
655 rksh93.1   := LINKSRC = ksh93.1
656 sh.1       := LINKSRC = ksh93.1

658 ldapadd.1  := LINKSRC = ldapmodify.1

660 ulimit.1   := LINKSRC = limit.1
661 unlimit.1  := LINKSRC = limit.1

663 dumpkeys.1 := LINKSRC = loadkeys.1

665 rmail.1    := LINKSRC = mail.1

667 page.1     := LINKSRC = more.1

669 snca.1     := LINKSRC = nca.1

671 pcat.1     := LINKSRC = pack.1
672 unpack.1   := LINKSRC = pack.1

674 pfcsh.1    := LINKSRC = pfexec.1
675 pfksh.1    := LINKSRC = pfexec.1
676 pfsh.1     := LINKSRC = pfexec.1

678 pkill.1    := LINKSRC = pgrep.1

680 pcred.1    := LINKSRC = proc.1
681 pfiles.1   := LINKSRC = proc.1
682 pflags.1   := LINKSRC = proc.1
683 pldd.1     := LINKSRC = proc.1
684 prun.1     := LINKSRC = proc.1
685 psig.1     := LINKSRC = proc.1
686 pstack.1   := LINKSRC = proc.1
687 pstop.1    := LINKSRC = proc.1
688 ptime.1    := LINKSRC = proc.1
689 pwait.1    := LINKSRC = proc.1
690 pwdx.1     := LINKSRC = proc.1

692 rmdir.1    := LINKSRC = rm.1

694 rmumount.1 := LINKSRC = rmmount.1

696 remote_shell.1 := LINKSRC = rsh.1
697 remsh.1     := LINKSRC = rsh.1

699 export.1   := LINKSRC = set.1
700 setenv.1   := LINKSRC = set.1
701 unset.1    := LINKSRC = set.1
702 unsetenv.1 := LINKSRC = set.1

704 case.1     := LINKSRC = shell_builtins.1
705 for.1      := LINKSRC = shell_builtins.1
706 foreach.1  := LINKSRC = shell_builtins.1
707 function.1 := LINKSRC = shell_builtins.1
708 if.1       := LINKSRC = shell_builtins.1
709 repeat.1   := LINKSRC = shell_builtins.1
710 select.1   := LINKSRC = shell_builtins.1
711 switch.1   := LINKSRC = shell_builtins.1
712 until.1    := LINKSRC = shell_builtins.1
713 while.1    := LINKSRC = shell_builtins.1

715 hashcheck.1 := LINKSRC = spell.1
716 hashmake.1  := LINKSRC = spell.1
717 spellin.1   := LINKSRC = spell.1

```

```

719 scp.1      := LINKSRC = scp.sunssh.1
720 sftp.1     := LINKSRC = sftp.sunssh.1
721 ssh.1      := LINKSRC = ssh.sunssh.1
722 ssh-add.1  := LINKSRC = ssh-add.sunssh.1
723 ssh-agent.1 := LINKSRC = ssh-agent.sunssh.1
724 ssh-http-proxy-connect.1 := LINKSRC = ssh-http-proxy-connect.sunssh.1
725 ssh-keygen.1 := LINKSRC = ssh-keygen.sunssh.1
726 ssh-keyscan.1 := LINKSRC = ssh-keyscan.sunssh.1
727 ssh-socks5-proxy-connect.1 := LINKSRC = ssh-socks5-proxy-connect.sunssh.1

729 strconf.1  := LINKSRC = strchg.1

731 settime.1  := LINKSRC = touch.1

733 onintr.1   := LINKSRC = trap.1

735 false.1   := LINKSRC = true.1

737 whence.1  := LINKSRC = typeset.1

739 # Links to usr/has/man

741 edit.1     := LINKSRC = ../../../has/man/man1has/edit.lhas
743 vedit.1    := LINKSRC = ../../../has/man/man1has/vi.lhas

745 .KEEP_STATE:

747 include   $(SRC)/man/Makefile.man

749 install:  $(ROOTMANFILES) $(ROOTMANLINKS)

```

```

*****
59567 Thu Jun 30 22:02:38 2016
new/usr/src/man/man1/ld.1
Code review comments from jeffpc
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
1 \" te
2 .\\ Copyright 1989 AT&T
3 .\\ Copyright (c) 2009, Sun Microsystems, Inc. All Rights Reserved
4 .\\ Copyright (c) 2012, Joyent, Inc. All Rights Reserved
5 .\\ The contents of this file are subject to the terms of the Common Development
6 .\\ See the License for the specific language governing permissions and limitat
7 .\\ the fields enclosed by brackets \"[]\" replaced with your own identifying info
8 .TH LD 1 \"Jun 6, 2016\"
8 .TH LD 1 \"Sep 10, 2013\"
9 .SH NAME
10 ld \- link-editor for object files
11 .SH SYNOPSIS
12 .LP
13 .nf
14 \fBld\fR [\fB-32\fR | \fB-64\fR] [\fB-a\fR | \fB-r\fR] [\fB-b\fR] [\fB-B\fR]direc
15 [\fB-B\fR]dynamic | static] [\fB-B\fR]eliminate] [\fB-B\fR]group] [\fB-B\fR]loca
16 [\fB-B\fR]reduce] [\fB-B\fR]symbolic] [\fB-B\fR]fC\fR] [\fB-B\fR]C\fR] [\fB-B\fR]d\fR
17 [\fB-B\fR]D\fR] [\fB-B\fR]fItoken\fR,...] [\fB-B\fR]e\fR] [\fB-B\fR]Iepsym\fR] [\fB-B\fR]f\fR] [\fB-B\fR]f\fR] [\fB-B\fR]F\fR]
18 [\fB-B\fR]i\fR] [\fB-B\fR]I\fR] [\fB-B\fR]fname\fR] [\fB-B\fR]l\fR] [\fB-B\fR]Ix\fR] [\fB-B\fR]l\fR] [\fB-B\fR]Ipath\fR] [\fB-B\fR]m
19 [\fB-B\fR]N\fR] [\fB-B\fR]fIstring\fR] [\fB-B\fR]o\fR] [\fB-B\fR]Ioutfile\fR] [\fB-B\fR]p\fR] [\fB-B\fR]Iauditlib\fR] [\fB-B
20 [\fB-B\fR]Q\fR] y | n] [\fB-B\fR]R\fR] [\fB-B\fR]Ipath\fR] [\fB-B\fR]s\fR] [\fB-B\fR]S\fR] [\fB-B\fR]Isupportlib\fR] [
21 [\fB-B\fR]u\fR] [\fB-B\fR]fIname\fR] [\fB-B\fR]V\fR] [\fB-B\fR]Y P\fR] [\fB-B\fR]fi,dirlist\fR] [\fB-B\fR]fR] [\fB-B\fR]fR] [\fB-B
22 [\fB-B\fR]z\fR] allextract | defaultextract | weakextract ] [\fB-B\fR]z\fR] altexec64]
23 [\fB-B\fR]z\fR] aslr[=\fBfIstate\fR]] [\fB-B\fR]z\fR] assert-deflib] [ \fB-B\fR]z\fR] assert-deflib=
24 [\fB-B\fR]z\fR] assert-deflib ] [ \fB-B\fR]z\fR] assert-deflib=\fBfIlibname\fR ]
25 [\fB-B\fR]z\fR] combreloc | nocombreloc ] [\fB-B\fR]z\fR] defs | nodefs]
26 [\fB-B\fR]z\fR] direct | nodirect] [\fB-B\fR]z\fR] endfiltee]
27 [\fB-B\fR]z\fR] fatal-warnings | nofatal-warnings ] [\fB-B\fR]z\fR] finiarrray=\fBfIfunction\fR]
28 [\fB-B\fR]z\fR] globalaudit] [\fB-B\fR]z\fR] groupperm | nogroupperm]
29 [\fB-B\fR]z\fR] guidance[=\fBfIid1\fR,\fBfIid2\fR,...] [\fB-B\fR]z\fR] help ]
30 [\fB-B\fR]z\fR] ignore | record] [\fB-B\fR]z\fR] inittarray=\fBfIfunction\fR] [\fB-B\fR]z\fR] initfir
31 [\fB-B\fR]z\fR] interpose] [\fB-B\fR]z\fR] lazyload | nolazyload]
32 [\fB-B\fR]z\fR] ld32=\fBfIarg1\fR,\fBfIarg2\fR,...] [\fB-B\fR]z\fR] ld64=\fBfIarg1\fR,\fBfIarg2\fR,.
33 [\fB-B\fR]z\fR] loadfltr] [\fB-B\fR]z\fR] muldefs] [\fB-B\fR]z\fR] nocompstrtab] [\fB-B\fR]z\fR] nodefau
34 [\fB-B\fR]z\fR] nodelete] [\fB-B\fR]z\fR] nodlopen] [\fB-B\fR]z\fR] nodump] [\fB-B\fR]z\fR] noldynsym]
35 [\fB-B\fR]z\fR] nopartial] [\fB-B\fR]z\fR] noversion] [\fB-B\fR]z\fR] now] [\fB-B\fR]z\fR] origin]
36 [\fB-B\fR]z\fR] preinittarray=\fBfIfunction\fR] [\fB-B\fR]z\fR] redlocsym] [\fB-B\fR]z\fR] relaxreloc
37 [\fB-B\fR]z\fR] rescanner] [\fB-B\fR]z\fR] rescanner] [\fB-B\fR]z\fR] rescanner-start \fBfI&... \fR] [\fB-B
38 [\fB-B\fR]z\fR] target=sparc|x86] [\fB-B\fR]z\fR] text | textwarn | textoff]
39 [\fB-B\fR]z\fR] verbose] [\fB-B\fR]z\fR] wrap=\fBfIsymbol\fR] [\fB-B\fR]z\fR] filename\fR...
40 .fi
41 .SH DESCRIPTION
42 .sp
43 .LP
44 The link-editor, \fBld\fR, combines relocatable object files by resolving
45 symbol references to symbol definitions, together with performing relocations.
46 \fBld\fR operates in two modes, static or dynamic, as governed by the \fB-d\fR
47 option. In all cases, the output of \fBld\fR is left in the file \fBld.out\fR by
48 default. See NOTES.
49 .sp
50 .LP
51 In dynamic mode, \fBld-dy\fR, the default, relocatable object files that are
52 provided as arguments are combined to produce an executable object file. This
53 file is linked at execution with any shared object files that are provided as
54 arguments. If the \fB-B-G\fR option is specified, relocatable object files are
55 combined to produce a shared object. Without the \fB-B-G\fR option, a dynamic

```

```

55 executable is created.
56 .sp
57 .LP
58 In static mode, \fBld-dn\fR, relocatable object files that are provided as
59 arguments are combined to produce a static executable file. If the \fB-r\fR
60 option is specified, relocatable object files are combined to produce one
61 relocatable object file. See \fBld-static Executables\fR.
62 .sp
63 .LP
64 Dynamic linking is the most common model for combining relocatable objects, and
65 the eventual creation of processes within Solaris. This environment tightly
66 couples the work of the link-editor and the runtime linker, \fBld.so.1\fR(1).
67 Both of these utilities, together with their related technologies and
68 utilities, are extensively documented in the \fBld-linker and Libraries Guide\fR.
69 .sp
70 .LP
71 If any argument is a library, \fBld-l\fR by default searches the library exactly
72 once at the point the library is encountered on the argument list. The library
73 can be either a shared object or relocatable archive. See \fBld-library\fR(3HEAD)).
74 .sp
75 .LP
76 A shared object consists of an indivisible, whole unit that has been generated
77 by a previous link-edit of one or more input files. When the link-editor
78 processes a shared object, the entire contents of the shared object become a
79 logical part of the resulting output file image. The shared object is not
80 physically copied during the link-edit as its actual inclusion is deferred
81 until process execution. This logical inclusion means that all symbol entries
82 defined in the shared object are made available to the link-editing process.
83 See Chapter 4, \fBld-shared Objects,\fR in \fBld-linker and Libraries Guide\fR
84 .sp
85 .LP
86 For an archive library, \fBld-l\fR loads only those routines that define an
87 unresolved external reference. \fBld-l\fR searches the symbol table of the
88 archive library sequentially to resolve external references that can be
89 satisfied by library members. This search is repeated until no external
90 references can be resolved by the archive. Thus, the order of members in the
91 library is functionally unimportant, unless multiple library members exist that
92 define the same external symbol. Archive libraries that have interdependencies
93 can require multiple command line definitions, or the use of one of the
94 \fBld-l\fR \fBld-lBrescan\fR options. See \fBld-lArchive Processing\fR in \fBld-linker and
95 Libraries Guide\fR.
96 .sp
97 .LP
98 \fBld-l\fR is a cross link-editor, able to link 32-bit objects or 64-bit objects,
99 for Sparc or x86 targets. \fBld-l\fR uses the \fBld-lBelf\fR class and machine type of
100 the first relocatable object on the command line to govern the mode in which to
101 operate. The mixing of 32-bit objects and 64-bit objects is not permitted.
102 Similarly, only objects of a single machine type are allowed. See the
103 \fBld-lB-32\fR, \fBld-lB-64\fR and \fBld-lB-z target\fR options, and the \fBld-lB_NOEXEC_64\fR
104 environment variable.
105 .SS "Static Executables"
106 .sp
107 .LP
108 The creation of static executables has been discouraged for many releases. In
109 fact, 64-bit system archive libraries have never been provided. Because a
110 static executable is built against system archive libraries, the executable
111 contains system implementation details. This self-containment has a number of
112 drawbacks.
113 .RS +4
114 .TP
115 .ie t \(\bu
116 .el o
117 The executable is immune to the benefits of system patches delivered as shared
118 objects. The executable therefore, must be rebuilt to take advantage of many
119 system improvements.
120 .RE

```

```

120 .RS +4
121 .TP
122 .ie t \(bu
123 .el o
124 The ability of the executable to run on future releases can be compromised.
125 .RE
126 .RS +4
127 .TP
128 .ie t \(bu
129 .el o
130 The duplication of system implementation details negatively affects system
131 performance.
132 .RE
133 .sp
134 .LP
135 With Solaris 10, 32-bit system archive libraries are no longer provided.
136 Without these libraries, specifically \fBlibc.a\fR, the creation of static
137 executables is no longer achievable without specialized system knowledge.
138 However, the capability of \fBld\fR to process static linking options, and the
139 processing of archive libraries, remains unchanged.
140 .SH OPTIONS
141 .sp
142 The following options are supported.
143 .sp
144 .ne 2
145 .na
146 \fB-32\fR | \fB-64\fR
147 .ad
148 .sp .6
149 .RS 4n
150 Creates a 32-bit, or 64-bit object.
151 .sp
152 By default, the class of the object being generated is determined from the
153 first \fBELF\fR object processed from the command line. If no objects are
154 specified, the class is determined by the first object encountered within the
155 first archive processed from the command line. If there are no objects or
156 archives, the link-editor creates a 32-bit object.
157 .sp
158 The \fB-64\fR option is required to create a 64-bit object solely from a
159 mapfile.
160 .sp
161 This \fB-32\fR or \fB-64\fR options can also be used in the rare case of
162 linking entirely from an archive that contains a mixture of 32 and 64-bit
163 objects. If the first object in the archive is not the class of the object that
164 is required to be created, then the \fB-32\fR or \fB-64\fR option can be used
165 to direct the link-editor. See \fIThe 32-bit link-editor and 64-bit
166 link-editor\fR in \fIlinker and Libraries Guide\fR.
167 .RE
169 .sp
170 .ne 2
171 .na
172 \fB-a\fR
173 .ad
174 .sp .6
175 .RS 4n
176 In static mode only, produces an executable object file. Undefined references
177 are not permitted. This option is the default behavior for static mode. The
178 \fB-a\fR option can not be used with the \fB-r\fR option. See \fBStatic
179 Executables\fR under DESCRIPTION.
180 .RE
182 .sp
183 .ne 2
184 .na

```

```

185 \fB-b\fR
186 .ad
187 .sp .6
188 .RS 4n
189 In dynamic mode only, provides no special processing for dynamic executable
190 relocations that reference symbols in shared objects. Without the \fB-b\fR
191 option, the link-editor applies techniques within a dynamic executable so that
192 the text segment can remain read-only. One technique is the creation of special
193 position-independent relocations for references to functions that are defined
194 in shared objects. Another technique arranges for data objects that are defined
195 in shared objects to be copied into the memory image of an executable at
196 runtime.
197 .sp
198 The \fB-b\fR option is intended for specialized dynamic objects and is not
199 recommended for general use. Its use suppresses all specialized processing
200 required to ensure an object's shareability, and can even prevent the
201 relocation of 64-bit executables.
202 .RE
204 .sp
205 .ne 2
206 .na
207 \fB-B\fR | \fB-nodirect\fR
208 .ad
209 .sp .6
210 .RS 4n
211 These options govern direct binding. \fB-B\fR | \fB-nodirect\fR establishes direct
212 binding information by recording the relationship between each symbol reference
213 together with the dependency that provides the definition. In addition, direct
214 binding information is established between each symbol reference and an
215 associated definition within the object being created. The runtime linker uses
216 this information to search directly for a symbol in the associated object
217 rather than to carry out a default symbol search.
218 .sp
219 Direct binding information can only be established to dependencies specified
220 with the link-edit. Thus, you should use the \fB-z\fR | \fB-undef\fR option.
221 Objects that wish to interpose on symbols in a direct binding environment
222 should identify themselves as interposers with the \fB-z\fR | \fB-binterpose\fR
223 option. The use of \fB-B\fR | \fB-nodirect\fR enables \fB-z\fR | \fB-blazyload\fR for
224 all dependencies.
225 .sp
226 The \fB-B\fR | \fB-nodirect\fR option prevents any direct binding to the
227 interfaces offered by the object being created. The object being created can
228 continue to directly bind to external interfaces by specifying the \fB-z\fR
229 | \fB-nodirect\fR option. See Appendix D, \fIIDL Direct Bindings\fR in \fIlinker and
230 Libraries Guide\fR.
231 .RE
233 .sp
234 .ne 2
235 .na
236 \fB-dynamic\fR | \fB-static\fR
237 .ad
238 .sp .6
239 .RS 4n
240 Options governing library inclusion. \fB-dynamic\fR | \fB-static\fR is valid in dynamic
241 mode only. These options can be specified any number of times on the command
242 line as toggles: if the \fB-dynamic\fR | \fB-static\fR option is given, no shared
243 objects are accepted until \fB-dynamic\fR | \fB-static\fR is seen. See the \fB-l\fR
244 option.
245 .RE
247 .sp
248 .ne 2
249 .na
250 \fB-B\fR | \fB-eliminate\fR

```

```

251 .ad
252 .sp .6
253 .RS 4n
254 Causes any global symbols, not assigned to a version definition, to be
255 eliminated from the symbol table. Version definitions can be supplied by means
256 of a \fBmapfile\fR to indicate the global symbols that should remain visible in
257 the generated object. This option achieves the same symbol elimination as the
258 \fIauto-elimination\fR directive that is available as part of a \fBmapfile\fR
259 version definition. This option can be useful when combining versioned and
260 non-versioned relocatable objects. See also the \fB-B\fR \fBlocal\fR option and
261 the \fB-B\fR \fBreduce\fR option. See \fIDefining Additional Symbols with a
262 mapfile\fR in \fIILinker and Libraries Guide\fR.
263 .RE

265 .sp
266 .ne 2
267 .na
268 \fB-B\fR \fBgroup\fR
269 .ad
270 .sp .6
271 .RS 4n
272 Establishes a shared object and its dependencies as a group. Objects within the
273 group are bound to other members of the group at runtime. This mode is similar
274 to adding the object to the process by using \fBdlopen\fR(3C) with the
275 \fBRTLD_GROUP\fR mode. An object that has an explicit dependency on a object
276 identified as a group, becomes a member of the group.
277 .sp
278 As the group must be self contained, use of the \fB-B\fR \fBgroup\fR option
279 also asserts the \fB-z\fR \fBdefs\fR option.
280 .RE

282 .sp
283 .ne 2
284 .na
285 \fB-B\fR \fBlocal\fR
286 .ad
287 .sp .6
288 .RS 4n
289 Causes any global symbols, not assigned to a version definition, to be reduced
290 to local. Version definitions can be supplied by means of a \fBmapfile\fR to
291 indicate the global symbols that should remain visible in the generated object.
292 This option achieves the same symbol reduction as the \fIauto-reduction\fR
293 directive that is available as part of a \fBmapfile\fR version definition. This
294 option can be useful when combining versioned and non-versioned relocatable
295 objects. See also the \fB-B\fR \fBeliminate\fR option and the \fB-B\fR
296 \fBreduce\fR option. See \fIDefining Additional Symbols with a mapfile\fR in
297 \fIILinker and Libraries Guide\fR.
298 .RE

300 .sp
301 .ne 2
302 .na
303 \fB-B\fR \fBreduce\fR
304 .ad
305 .sp .6
306 .RS 4n
307 When generating a relocatable object, causes the reduction of symbolic
308 information defined by any version definitions. Version definitions can be
309 supplied by means of a \fBmapfile\fR to indicate the global symbols that should
310 remain visible in the generated object. By default, when a relocatable object
311 is generated, version definitions are only recorded in the output image. The
312 actual reduction of symbolic information is carried out when the object is used
313 in the construction of a dynamic executable or shared object. The \fB-B\fR
314 \fBreduce\fR option is applied automatically when a dynamic executable or
315 shared object is created.
316 .RE

```

```

318 .sp
319 .ne 2
320 .na
321 \fB-B\fR \fBsymbolic\fR
322 .ad
323 .sp .6
324 .RS 4n
325 In dynamic mode only. When building a shared object, binds references to global
326 symbols to their definitions, if available, within the object. Normally,
327 references to global symbols within shared objects are not bound until runtime,
328 even if definitions are available. This model allows definitions of the same
329 symbol in an executable or other shared object to override the object's own
330 definition. \fBld\fR issues warnings for undefined symbols unless \fB-z\fR
331 \fBdefs\fR overrides.
332 .sp
333 The \fB-B\fR \fBsymbolic\fR option is intended for specialized dynamic objects
334 and is not recommended for general use. To reduce the runtime relocation
335 processing that is required an object, the creation of a version definition is
336 recommended.
337 .RE

339 .sp
340 .ne 2
341 .na
342 \fB-B\fR \fBc\fR \fIname\fR
343 .ad
344 .sp .6
345 .RS 4n
346 Records the configuration file \fIname\fR for use at runtime. Configuration
347 files can be employed to alter default search paths, provide a directory cache,
348 together with providing alternative object dependencies. See \fBcrle\fR(1).
349 .RE

351 .sp
352 .ne 2
353 .na
354 \fB-B\fR \fBc\fR
355 .ad
356 .sp .6
357 .RS 4n
358 Demangles C++ symbol names displayed in diagnostic messages.
359 .RE

361 .sp
362 .ne 2
363 .na
364 \fB-B\fR \fBd\fR \fBy\fR | \fBn\fR
365 .ad
366 .sp .6
367 .RS 4n
368 When \fB-B\fR \fBy\fR, the default, is specified, \fBld\fR uses dynamic
369 linking. When \fB-B\fR \fBn\fR is specified, \fBld\fR uses static linking. See
370 \fBStatic Executables\fR under DESCRIPTION, and \fB-B\fR
371 \fBdynamic\fR|\fBstatic\fR.
372 .RE

374 .sp
375 .ne 2
376 .na
377 \fB-B\fR \fB-D\fR \fItoken\fR,...\fR
378 .ad
379 .sp .6
380 .RS 4n
381 Prints debugging information as specified by each \fItoken\fR, to the standard
382 error. The special token \fBhelp\fR indicates the full list of tokens

```

```

383 available. See \fIDebugging Aids\fR in \fILinker and Libraries Guide\fR.
384 .RE

386 .sp
387 .ne 2
388 .na
389 \fB\fB-e\fR \fIepsym\fR\fR
390 .ad
391 .br
392 .na
393 \fB\fB--entry\fR \fIepsym\fR\fR
394 .ad
395 .sp .6
396 .RS 4n
397 Sets the entry point address for the output file to be the symbol \fIepsym\fR.
398 .RE

400 .sp
401 .ne 2
402 .na
403 \fB\fB-f\fR \fIname\fR\fR
404 .ad
405 .br
406 .na
407 \fB\fB--auxiliary\fR \fIname\fR\fR
408 .ad
409 .sp .6
410 .RS 4n
411 Useful only when building a shared object. Specifies that the symbol table of
412 the shared object is used as an auxiliary filter on the symbol table of the
413 shared object specified by \fIname\fR. Multiple instances of this option are
414 allowed. This option can not be combined with the \fB-F\fR option. See
415 \fIGenerating Auxiliary Filters\fR in \fILinker and Libraries Guide\fR.
416 .RE

418 .sp
419 .ne 2
420 .na
421 \fB\fB-F\fR \fIname\fR\fR
422 .ad
423 .br
424 .na
425 \fB\fB--filter\fR \fIname\fR\fR
426 .ad
427 .sp .6
428 .RS 4n
429 Useful only when building a shared object. Specifies that the symbol table of
430 the shared object is used as a filter on the symbol table of the shared object
431 specified by \fIname\fR. Multiple instances of this option are allowed. This
432 option can not be combined with the \fB-f\fR option. See \fIGenerating Standard
433 Filters\fR in \fILinker and Libraries Guide\fR.
434 .RE

436 .sp
437 .ne 2
438 .na
439 \fB\fB-G\fR\fR
440 .ad
441 .br
442 .na
443 \fB\fB-shared\fR\fR
444 .ad
445 .sp .6
446 .RS 4n
447 In dynamic mode only, produces a shared object. Undefined symbols are allowed.
448 See Chapter 4, \fIShared Objects,\fR in \fILinker and Libraries Guide\fR.

```

```

449 .RE

451 .sp
452 .ne 2
453 .na
454 \fB\fB-h\fR \fIname\fR\fR
455 .ad
456 .br
457 .na
458 \fB\fB--soname\fR \fIname\fR\fR
459 .ad
460 .sp .6
461 .RS 4n
462 In dynamic mode only, when building a shared object, records \fIname\fR in the
463 object's dynamic section. \fIname\fR is recorded in any dynamic objects that
464 are linked with this object rather than the object's file system name.
465 Accordingly, \fIname\fR is used by the runtime linker as the name of the shared
466 object to search for at runtime. See \fIRecording a Shared Object Name\fR in
467 \fILinker and Libraries Guide\fR.
468 .RE

470 .sp
471 .ne 2
472 .na
473 \fB\fB-i\fR\fR
474 .ad
475 .sp .6
476 .RS 4n
477 Ignores \fBLD_LIBRARY_PATH\fR. This option is useful when an
478 \fBLD_LIBRARY_PATH\fR setting is in effect to influence the runtime library
479 search, which would interfere with the link-editing being performed.
480 .RE

482 .sp
483 .ne 2
484 .na
485 \fB\fB-I\fR \fIname\fR\fR
486 .ad
487 .br
488 .na
489 \fB\fB--dynamic-linker\fR \fIname\fR\fR
490 .ad
491 .sp .6
492 .RS 4n
493 When building an executable, uses \fIname\fR as the path name of the
494 interpreter to be written into the program header. The default in static mode
495 is no interpreter. In dynamic mode, the default is the name of the runtime
496 linker, \fBld.so.1\fR(1). Either case can be overridden by \fB-I\fR \fIname\fR.
497 \fBbexec\fR(2) loads this interpreter when the \fBa.out\fR is loaded, and passes
498 control to the interpreter rather than to the \fBa.out\fR directly.
499 .RE

501 .sp
502 .ne 2
503 .na
504 \fB\fB-l\fR \fIx\fR\fR
505 .ad
506 .br
507 .na
508 \fB\fB--library\fR \fIx\fR\fR
509 .ad
510 .sp .6
511 .RS 4n
512 Searches a library \fBlib\fR\fIx\fR\fB.so\fR or \fBlib\fR\fIx\fR\fB.a\fR,
513 the conventional names for shared object and archive libraries, respectively.
514 In dynamic mode, unless the \fB-B\fR \fBbstatic\fR option is in effect, \fBld\fR

```

515 searches each directory specified in the library search path for a
 516 `\fBlib\fR\fIx\fR\FB\&.so\fR` or `\fBlib\fR\fIx\fR\FB\&.a\fR` file. The directory
 517 search stops at the first directory containing either. `\fBld\fR` chooses the
 518 file ending in `\fB\&.so\fR` if `\fB-l\fR\fIx\fR` expands to two files with names
 519 of the form `\fBlib\fR\fIx\fR\FB\&.so\fR` and `\fBlib\fR\fIx\fR\FB\&.a\fR`. If no
 520 `\fBlib\fR\fIx\fR\FB\&.so\fR` is found, then `\fBld\fR` accepts
 521 `\fBlib\fR\fIx\fR\FB\&.a\fR`. In static mode, or when the `\fB-B\fR \fBstatic\fR`
 522 option is in effect, `\fBld\fR` selects only the file ending in `\fB\&.a\fR`.
 523 `\fBld\fR` searches a library when the library is encountered, so the placement
 524 of `\fB-l\fR` is significant. See `\fILinking With Additional Libraries\fR` in
 525 `\fILinker and Libraries Guide\fR`.
 526 .RE

528 .sp
 529 .ne 2
 530 .na
 531 `\fB\FB-L\fR \fIpath\fR\fR`
 532 .ad
 533 .br
 534 .na
 535 `\fB\FB--library-path\fR \fIpath\fR\fR`
 536 .ad
 537 .sp .6
 538 .RS 4n
 539 Adds `\fIpath\fR` to the library search directories. `\fBld\fR` searches for
 540 libraries first in any directories specified by the `\fB-L\fR` options and then
 541 in the standard directories. This option is useful only if the option precedes
 542 the `\fB-l\fR` options to which the `\fB-L\fR` option applies. See `\fIIDirectories`
 543 Searched by the Link-Editor\fR in `\fILinker and Libraries Guide\fR`.
 544 .sp
 545 The environment variable `\fBLD_LIBRARY_PATH\fR` can be used to supplement the
 546 library search path, however the `\fB-L\fR` option is recommended, as the
 547 environment variable is also interpreted by the runtime environment. See
 548 `\fBLD_LIBRARY_PATH\fR` under ENVIRONMENT VARIABLES.
 549 .RE

551 .sp
 552 .ne 2
 553 .na
 554 `\fB\FB-m\fR\fR`
 555 .ad
 556 .sp .6
 557 .RS 4n
 558 Produces a memory map or listing of the input/output sections, together with
 559 any non-fatal multiply-defined symbols, on the standard output.
 560 .RE

562 .sp
 563 .ne 2
 564 .na
 565 `\fB\FB-M\fR \fImapfile\fR\fR`
 566 .ad
 567 .sp .6
 568 .RS 4n
 569 Reads `\fImapfile\fR` as a text file of directives to `\fBld\fR`. This option can
 570 be specified multiple times. If `\fImapfile\fR` is a directory, then all regular
 571 files, as defined by `\fBstat\fR(2)`, within the directory are processed. See
 572 Chapter 9, `\fIMapfile Option,\fR` in `\fILinker and Libraries Guide\fR`. Example
 573 mapfiles are provided in `\fB/usr/lib/ld\fR`. See FILES.
 574 .RE

576 .sp
 577 .ne 2
 578 .na
 579 `\fB\FB-N\fR \fIstring\fR\fR`
 580 .ad

581 .sp .6
 582 .RS 4n
 583 This option causes a `\fBDT_NEEDED\fR` entry to be added to the `\fB\&.dynamic\fR`
 584 section of the object being built. The value of the `\fBDT_NEEDED\fR` string is
 585 the `\fIstring\fR` that is specified on the command line. This option is position
 586 dependent, and the `\fBDT_NEEDED\fR \fB\&.dynamic\fR` entry is relative to the
 587 other dynamic dependencies discovered on the link-edit line. This option is
 588 useful for specifying dependencies within device driver relocatable objects
 589 when combined with the `\fB-dy\fR` and `\fB-r\fR` options.
 590 .RE

592 .sp
 593 .ne 2
 594 .na
 595 `\fB\FB-o\fR \fIoutfile\fR\fR`
 596 .ad
 597 .br
 598 .na
 599 `\fB\FB--output\fR \fIoutfile\fR\fR`
 600 .ad
 601 .sp .6
 602 .RS 4n
 603 Produces an output object file that is named `\fIoutfile\fR`. The name of the
 604 default object file is `\fBa.out\fR`.
 605 .RE

607 .sp
 608 .ne 2
 609 .na
 610 `\fB\FB-p\fR \fIauditlib\fR\fR`
 611 .ad
 612 .sp .6
 613 .RS 4n
 614 Identifies an audit library, `\fIauditlib\fR`. This audit library is used to
 615 audit the object being created at runtime. A shared object identified as
 616 requiring auditing with the `\fB-p\fR` option, has this requirement inherited by
 617 any object that specifies the shared object as a dependency. See the `\fB-P\fR`
 618 option. See `\fIRuntime Linker Auditing Interface\fR` in `\fILinker and Libraries`
 619 `Guide\fR`.
 620 .RE

622 .sp
 623 .ne 2
 624 .na
 625 `\fB\FB-P\fR \fIauditlib\fR\fR`
 626 .ad
 627 .sp .6
 628 .RS 4n
 629 Identifies an audit library, `\fIauditlib\fR`. This audit library is used to
 630 audit the dependencies of the object being created at runtime. Dependency
 631 auditing can also be inherited from dependencies that are identified as
 632 requiring auditing. See the `\fB-p\fR` option, and the `\fB-z\fR \fBglobalaudit\fR`
 633 option. See `\fIRuntime Linker Auditing Interface\fR` in `\fILinker and Libraries`
 634 `Guide\fR`.
 635 .RE

637 .sp
 638 .ne 2
 639 .na
 640 `\fB\FB-Q\fR \fIby\fR | \fBn\fR\fR`
 641 .ad
 642 .sp .6
 643 .RS 4n
 644 Under `\fB-Q\fR \fIby\fR`, an `\fBident\fR` string is added to the `\fB\&.comment\fR`
 645 section of the output file. This string identifies the version of the `\fBld\fR`
 646 used to create the file. This results in multiple `\fBld\fR \fBidents\fR` when

647 there have been multiple linking steps, such as when using `\fBld\fR` `\fB-r\fR`.
 648 This identification is identical with the default action of the `\fBcc\fR`
 649 command. `\fB-Q\fR` `\fBn\fR` suppresses version identification. `\fB&.comment\fR`
 650 sections can be manipulated by the `\fBmcs\fR(1)` utility.
 651 .RE

653 .sp
 654 .ne 2
 655 .na
 656 `\fB\fB-r\fR\fR`
 657 .ad
 658 .br
 659 .na
 660 `\fB\fB--relocatable\fR\fR`
 661 .ad
 662 .sp .6
 663 .RS 4n
 664 Combines relocatable object files to produce one relocatable object file.
 665 `\fBld\fR` does not complain about unresolved references. This option cannot be
 666 used with the `\fB-a\fR` option.
 667 .RE

669 .sp
 670 .ne 2
 671 .na
 672 `\fB\fB-R\fR` `\fIpath\fR`
 673 .ad
 674 .br
 675 .na
 676 `\fB\fB-rpath\fR` `\fIpath\fR`
 677 .ad
 678 .sp .6
 679 .RS 4n
 680 A colon-separated list of directories used to specify library search
 681 directories to the runtime linker. If present and not NULL, the path is
 682 recorded in the output object file and passed to the runtime linker. Multiple
 683 instances of this option are concatenated together with each `\fIpath\fR`
 684 separated by a colon. See `\fIIDirectories Searched by the Runtime Linker\fR` in
 685 `\fIILinker and Libraries Guide\fR`.
 686 .sp
 687 The use of a `runpath` within an associated object is preferable to setting
 688 global search paths such as through the `\fBLD_LIBRARY_PATH\fR` environment
 689 variable. Only the `runpaths` that are necessary to find the objects dependencies
 690 should be recorded. `\fBldd\fR(1)` can also be used to discover unused `runpaths`
 691 in dynamic objects, when used with the `\fB-U\fR` option.
 692 .sp
 693 Various tokens can also be supplied with a `runpath` that provide a flexible
 694 means of identifying system capabilities or an objects location. See Appendix
 695 C, `\fIEstablishing Dependencies with Dynamic String Tokens,\fR` in `\fIILinker` and
 696 `\fILibraries Guide\fR`. The `\fB$ORIGIN\fR` token is especially useful in allowing
 697 dynamic objects to be relocated to different locations in the file system.
 698 .RE

700 .sp
 701 .ne 2
 702 .na
 703 `\fB\fB-s\fR` `\fR`
 704 .ad
 705 .br
 706 .na
 707 `\fB\fB--strip-all\fR`
 708 .ad
 709 .sp .6
 710 .RS 4n
 711 Strips symbolic information from the output file. Any debugging information,
 712 that is, `\fB&.line\fR`, `\fB&.debug*\fR`, and `\fB&.stab*\fR` sections, and their

713 associated relocation entries are removed. Except for relocatable files, a
 714 symbol table `\fBSHT_SYMTAB\fR` and its associated string table section are not
 715 created in the output object file. The elimination of a `\fBSHT_SYMTAB\fR` symbol
 716 table can reduce the `\fB&.stab*\fR` debugging information that is generated
 717 using the compiler drivers `\fB-g\fR` option. See the `\fB-z\fR` `\fBbredlocsym\fR`
 718 and `\fB-z\fR` `\fBboldynsym\fR` options.
 719 .RE

721 .sp
 722 .ne 2
 723 .na
 724 `\fB\fB-S\fR` `\fIsupportlib\fR`
 725 .ad
 726 .sp .6
 727 .RS 4n
 728 The shared object `\fIsupportlib\fR` is loaded with `\fBld\fR` and given
 729 information regarding the linking process. Shared objects that are defined by
 730 using the `\fB-S` option can also be supplied using the `\fBSGS_SUPPORT`
 731 environment variable. See `\fIILink-Editor Support Interface\fR` in `\fIILinker` and
 732 `\fILibraries Guide\fR`.
 733 .RE

735 .sp
 736 .ne 2
 737 .na
 738 `\fB\fB-t` `\fR`
 739 .ad
 740 .sp .6
 741 .RS 4n
 742 Turns off the warning for multiply-defined symbols that have different sizes or
 743 different alignments.
 744 .RE

746 .sp
 747 .ne 2
 748 .na
 749 `\fB\fB-u` `\fIisymname`
 750 .ad
 751 .br
 752 .na
 753 `\fB\fB--undefined` `\fIisymname`
 754 .ad
 755 .sp .6
 756 .RS 4n
 757 Enters `\fIisymname` as an undefined symbol in the symbol table. This option is
 758 useful for loading entirely from an archive library. In this instance, an
 759 unresolved reference is needed to force the loading of the first routine. The
 760 placement of this option on the command line is significant. This option must
 761 be placed before the library that defines the symbol. See `\fIDefining`
 762 `Additional Symbols with the u option` in `\fIILinker and Libraries Guide`.
 763 .RE

765 .sp
 766 .ne 2
 767 .na
 768 `\fB\fB-V` `\fR`
 769 .ad
 770 .br
 771 .na
 772 `\fB\fB--version`
 773 .ad
 774 .sp .6
 775 .RS 4n
 776 Outputs a message giving information about the version of `\fBld` being used.
 777 .RE


```

779 .sp
780 .ne 2
781 .na
782 \fB\fB-Y\fR \fBPF,\fR\fIdirlist\fR\fR
783 .ad
784 .sp .6
785 .RS 4n
786 Changes the default directories used for finding libraries. \fIdirlist\fR is a
787 colon-separated path list.
788 .RE

790 .sp
791 .ne 2
792 .na
793 \fB\fB-z\fR \fBabsexec\fR\fR
794 .ad
795 .sp .6
796 .RS 4n
797 Useful only when building a dynamic executable. Specifies that references to
798 external absolute symbols should be resolved immediately instead of being left
799 for resolution at runtime. In very specialized circumstances, this option
800 removes text relocations that can result in excessive swap space demands by an
801 executable.
802 .RE

804 .sp
805 .ne 2
806 .na
807 \fB\fB-z\fR \fBalleextract\fR | \fBdefaultextract\fR | \fBweakextract\fR\fR
808 .ad
809 .br
810 .na
811 \fB\fB--whole-archive\fR | \fB--no-whole-archive\fR\fR
812 .ad
813 .sp .6
814 .RS 4n
815 Alters the extraction criteria of objects from any archives that follow. By
816 default, archive members are extracted to satisfy undefined references and to
817 promote tentative definitions with data definitions. Weak symbol references do
818 not trigger extraction. Under the \fB-z\fR \fBalleextract\fR or
819 \fB--whole-archive\fR options, all archive members are extracted from the
820 archive. Under \fB-z\fR \fBweakextract\fR, weak references trigger archive
821 extraction. The \fB-z\fR \fBdefaultextract\fR or \fB--no-whole-archive\fR
822 options provide a means of returning to the default following use of the former
823 extract options. See \fIArchive Processing\fR in \fILinker and Libraries
824 Guide\fR.
825 .RE

827 .sp
828 .ne 2
829 .na
830 \fB\fB-z\fR \fBaltexec64\fR\fR
831 .ad
832 .sp .6
833 .RS 4n
834 Execute the 64-bit \fBld\fR. The creation of very large 32-bit objects can
835 exhaust the virtual memory that is available to the 32-bit \fBld\fR. The
836 \fB-z\fR \fBaltexec64\fR option can be used to force the use of the associated
837 64-bit \fBld\fR. The 64-bit \fBld\fR provides a larger virtual address space
838 for building 32-bit objects. See \fIThe 32-bit link-editor and 64-bit
839 link-editor\fR in \fILinker and Libraries Guide\fR.
840 .RE

842 .sp
843 .ne 2
844 .na

```

```

845 \fB-z\fR \fBbaslr[=\fIstate\fR]\fR
846 .ad
847 .sp .6
848 .RS 4n
849 Specify whether the executable's address space should be randomized on
850 execution. If \fIstate\fR is "enabled" randomization will always occur when
851 this executable is run (regardless of inherited settings). If \fIstate\fR is
852 "disabled" randomization will never occur when this executable is run. If
853 \fIstate\fR is omitted, ASLR is enabled.

855 An executable that should simply use the settings inherited from its
856 environment should not use this flag at all.
857 .RE

859 .sp
860 .ne 2
861 .na
862 #endif /* ! codereview */
863 \fB\fB-z\fR \fBcombrelloc\fR | \fBnocombrelloc\fR\fR
864 .ad
865 .sp .6
866 .RS 4n
867 By default, \fBld\fR combines multiple relocation sections when building
868 executables or shared objects. This section combination differs from
869 relocatable objects, in which relocation sections are maintained in a
870 one-to-one relationship with the sections to which the relocations must be
871 applied. The \fB-z\fR \fBnocombrelloc\fR option disables this merging of
872 relocation sections, and preserves the one-to-one relationship found in the
873 original relocatable objects.
874 .sp
875 \fBld\fR sorts the entries of data relocation sections by their symbol
876 reference. This sorting reduces runtime symbol lookup. When multiple relocation
877 sections are combined, this sorting produces the least possible relocation
878 overhead when objects are loaded into memory, and speeds the runtime loading of
879 dynamic objects.
880 .sp
881 Historically, the individual relocation sections were carried over to any
882 executable or shared object, and the \fB-z\fR \fBcombrelloc\fR option was
883 required to enable the relocation section merging previously described.
884 Relocation section merging is now the default. The \fB-z\fR \fBcombrelloc\fR
885 option is still accepted for the benefit of old build environments, but the
886 option is unnecessary, and has no effect.
887 .RE

889 .sp
890 .ne 2
891 .na
892 \fB\fB-z\fR \fBassert-deflib\fR\fR
893 .ad
894 .br
895 .na
896 \fB\fB-z\fR \fBassert-deflib=\fR\fIlibname\fR\fR
897 .ad
898 .sp .6
899 .RS 4n
900 Enables warnings that check the location of where libraries passed in with
901 \fB-l\fR are found. If the link-editor finds a library on its default search
902 path it will emit a warning. This warning can be made fatal in conjunction with
903 the option \fB-z fatal-warnings\fR. Passing \fIlibname\fR white lists a library
904 from this check. The library must be the full name of the library, e.g.
905 \fIlibc.so\fR. To white list multiple libraries, the \fB-z
906 assert-deflib=\fR\fIlibname\fR option can be repeated multiple times. This
907 option is useful when trying to build self-contained objects where a referenced
908 library might exist in the default system library path and in alternate paths
909 specified by \fB-L\fR, but you only want the alternate paths to be used.
910 .RE

```

```

912 .sp
913 .ne 2
914 .na
915 \fB\fB-z\fR \fBdefs\fR | \fBnodefs\fR\fR
916 .ad
917 .br
918 .na
919 \fB\fB--no-undefined\fR\fR
920 .ad
921 .sp .6
922 .RS 4n
923 The \fB-z\fR \fBdefs\fR option and the \fB--no-undefined\fR option force a
924 fatal error if any undefined symbols remain at the end of the link. This mode
925 is the default when an executable is built. For historic reasons, this mode is
926 \fBnot\fR the default when building a shared object. Use of the \fB-z\fR
927 \fBdefs\fR option is recommended, as this mode assures the object being built
928 is self-contained. A self-contained object has all symbolic references resolved
929 internally, or to the object's immediate dependencies.
930 .sp
931 The \fB-z\fR \fBnodefs\fR option allows undefined symbols. For historic
932 reasons, this mode is the default when a shared object is built. When used with
933 executables, the behavior of references to such undefined symbols is
934 unspecified. Use of the \fB-z\fR \fBnodefs\fR option is not recommended.
935 .RE

937 .sp
938 .ne 2
939 .na
940 \fB\fB-z\fR \fBdirect\fR | \fBnodirect\fR\fR
941 .ad
942 .sp .6
943 .RS 4n
944 Enables or disables direct binding to any dependencies that follow on the
945 command line. These options allow finer control over direct binding than the
946 global counterpart \fB-B\fR \fBdirect\fR. The \fB-z\fR \fBdirect\fR option also
947 differs from the \fB-B\fR \fBdirect\fR option in the following areas. Direct
948 binding information is not established between a symbol reference and an
949 associated definition within the object being created. Lazy loading is not
950 enabled.
951 .RE

953 .sp
954 .ne 2
955 .na
956 \fB\fB-z\fR \fBendfiltee\fR\fR
957 .ad
958 .sp .6
959 .RS 4n
960 Marks a filtee so that when processed by a filter, the filtee terminates any
961 further filtee searches by the filter. See \fIIReducing Filtee Searches\fR in
962 \fIILinker and Libraries Guide\fR.
963 .RE

965 .sp
966 .ne 2
967 .na
968 \fB\fB-z\fR \fBfatal-warnings\fR | \fBnofatal-warnings\fR\fR
969 .ad
970 .br
971 .na
972 \fB\fB--fatal-warnings\fR | \fB--no-fatal-warnings\fR
973 .ad
974 .sp .6
975 .RS 4n
976 Controls the behavior of warnings emitted from the link-editor. Setting \fB-z

```

```

977 fatal-warnings\fR promotes warnings emitted by the link-editor to fatal errors
978 that will cause the link-editor to fail before linking. \fB-z
979 nofatal-warnings\fR instead demotes these warnings such that they will not cause
980 the link-editor to exit prematurely.
981 .RE

984 .sp
985 .ne 2
986 .na
987 \fB\fB-z\fR \fBfiniarray=\fR\fIfunction\fR\fR
988 .ad
989 .sp .6
990 .RS 4n
991 Appends an entry to the \fB\&.finiarray\fR section of the object being built.
992 If no \fB\&.finiarray\fR section is present, a section is created. The new
993 entry is initialized to point to \fIfunction\fR. See \fIInitialization and
994 Termination Sections\fR in \fIILinker and Libraries Guide\fR.
995 .RE

997 .sp
998 .ne 2
999 .na
1000 \fB\fB-z\fR \fBglobalaudit\fR\fR
1001 .ad
1002 .sp .6
1003 .RS 4n
1004 This option supplements an audit library definition that has been recorded with
1005 the \fB-P\fR option. This option is only meaningful when building a dynamic
1006 executable. Audit libraries that are defined within an object with the \fB-P\fR
1007 option typically allow for the auditing of the immediate dependencies of the
1008 object. The \fB-z\fR \fBglobalaudit\fR promotes the auditor to a global
1009 auditor, thus allowing the auditing of all dependencies. See \fIInvoking the
1010 Auditing Interface\fR in \fIILinker and Libraries Guide\fR.
1011 .sp
1012 An auditor established with the \fB-P\fR option and the \fB-z\fR
1013 \fBglobalaudit\fR option, is equivalent to the auditor being established with
1014 the \fBBLD_AUDIT\fR environment variable. See \fBld.so.1\fR(1).
1015 .RE

1017 .sp
1018 .ne 2
1019 .na
1020 \fB\fB-z\fR \fBgroupperm\fR | \fBnogroupperm\fR\fR
1021 .ad
1022 .sp .6
1023 .RS 4n
1024 Assigns, or deassigns each dependency that follows to a unique group. The
1025 assignment of a dependency to a group has the same effect as if the dependency
1026 had been built using the \fB-B\fR \fBgroup\fR option.
1027 .RE

1029 .sp
1030 .ne 2
1031 .na
1032 \fB-z\fR \fBguidance\fR[=\fIid1\fR,\fIid2\fR...]
1033 .ad
1034 .sp .6
1035 .RS 4n
1036 Give messages suggesting link-editor features that could improve the resulting
1037 dynamic object.
1038 .LP
1039 Specific classes of suggestion can be silenced by specifying an optional comma s
1040 list of guidance identifiers.
1041 .LP
1042 The current classes of suggestion provided are:

```

```

1044 .sp
1045 .ne 2
1046 .na
1047 Enable use of direct binding
1048 .ad
1049 .sp .6
1050 .RS 4n
1051 Suggests that \fB-z direct\fR or \fB-B direct\fR be present prior to any
1052 specified dependency. This allows predictable symbol binding at runtime.

1054 Can be disabled with \fB-z guidance=nodirect\fR
1055 .RE

1057 .sp
1058 .ne 2
1059 .na
1060 Enable lazy dependency loading
1061 .ad
1062 .sp .6
1063 .RS 4n
1064 Suggests that \fB-z lazyload\fR be present prior to any specified dependency.
1065 This allows the dynamic object to be loaded more quickly.

1067 Can be disabled with \fB-z guidance=nolazyload\fR.
1068 .RE

1070 .sp
1071 .ne 2
1072 .na
1073 Shared objects should define all their dependencies.
1074 .ad
1075 .sp .6
1076 .RS 4n
1077 Suggests that \fB-z defs\fR be specified on the link-editor command line.
1078 Shared objects that explicitly state all their dependencies behave more
1079 predictably when used.

1081 Can be disabled with \fB-z guidance=noddefs\fR
1082 .RE

1084 .sp
1085 .ne 2
1086 .na
1087 Version 2 mapfile syntax
1088 .ad
1089 .sp .6
1090 .RS 4n
1091 Suggests that any specified mapfiles use the more readable version 2 syntax.

1093 Can be disabled with \fB-z guidance=nomapfile\fR.
1094 .RE

1096 .sp
1097 .ne 2
1098 .na
1099 Read-only text segment
1100 .ad
1101 .sp .6
1102 .RS 4n
1103 Should any runtime relocations within the text segment exist, suggests that
1104 the object be compiled with position independent code (PIC). Keeping large
1105 allocatable sections read-only allows them to be shared between processes
1106 using a given shared object.

1108 Can be disabled with \fB-z guidance=notext\fR

```

```

1109 .RE

1111 .sp
1112 .ne 2
1113 .na
1114 No unused dependencies
1115 .ad
1116 .sp .6
1117 .RS 4n
1118 Suggests that any dependency not referenced by the resulting dynamic object be
1119 removed from the link-editor command line.

1121 Can be disabled with \fB-z guidance=nounused\fR.
1122 .RE
1123 .RE

1125 .sp
1126 .ne 2
1127 .na
1128 \fB\fB-z\fR \fBhhelp\fR\fR
1129 .ad
1130 .br
1131 .na
1132 \fB\fB--help\fR\fR
1133 .ad
1134 .sp .6
1135 .RS 4n
1136 Print a summary of the command line options on the standard output and exit.
1137 .RE

1139 .sp
1140 .ne 2
1141 .na
1142 \fB\fB-z\fR \fBignore\fR | \fBrecord\fR\fR
1143 .ad
1144 .sp .6
1145 .RS 4n
1146 Ignores, or records, dynamic dependencies that are not referenced as part of
1147 the link-edit. Ignores, or records, unreferenced \fB\fBELF\fR sections from the
1148 relocatable objects that are read as part of the link-edit. By default,
1149 \fB\fB-z\fR \fBrecord\fR is in effect.
1150 .sp
1151 If an \fB\fBELF\fR section is ignored, the section is eliminated from the output
1152 file being generated. A section is ignored when three conditions are true. The
1153 eliminated section must contribute to an allocatable segment. The eliminated
1154 section must provide no global symbols. No other section from any object that
1155 contributes to the link-edit, must reference an eliminated section.
1156 .RE

1158 .sp
1159 .ne 2
1160 .na
1161 \fB\fB-z\fR \fBinitarray=\fR\fR\fR
1162 .ad
1163 .sp .6
1164 .RS 4n
1165 Appends an entry to the \fB\fB.&.initarray\fR section of the object being built.
1166 If no \fB\fB.&.initarray\fR section is present, a section is created. The new
1167 entry is initialized to point to \fB\fBifunction\fR. See \fB\fBInitialization and
1168 Termination Sections\fR in \fB\fBLinker and Libraries Guide\fR.
1169 .RE

1171 .sp
1172 .ne 2
1173 .na
1174 \fB\fB-z\fR \fBinitfirst\fR\fR

```

```

1175 .ad
1176 .sp .6
1177 .RS 4n
1178 Marks the object so that its runtime initialization occurs before the runtime
1179 initialization of any other objects brought into the process at the same time.
1180 In addition, the object runtime finalization occurs after the runtime
1181 finalization of any other objects removed from the process at the same time.
1182 This option is only meaningful when building a shared object.
1183 .RE

1185 .sp
1186 .ne 2
1187 .na
1188 \fB\fB-z\fR \fBinterpose\fR\fR
1189 .ad
1190 .sp .6
1191 .RS 4n
1192 Marks the object as an interposer. At runtime, an object is identified as an
1193 explicit interposer if the object has been tagged using the \fB-z interpose\fR
1194 option. An explicit interposer is also established when an object is loaded
1195 using the \fBLD_PRELOAD\fR environment variable. Implicit interposition can
1196 occur because of the load order of objects, however, this implicit
1197 interposition is unknown to the runtime linker. Explicit interposition can
1198 ensure that interposition takes place regardless of the order in which objects
1199 are loaded. Explicit interposition also ensures that the runtime linker
1200 searches for symbols in any explicit interposers when direct bindings are in
1201 effect.
1202 .RE

1204 .sp
1205 .ne 2
1206 .na
1207 \fB\fB-z\fR \fBlazyload\fR | \fBnolazyload\fR\fR
1208 .ad
1209 .sp .6
1210 .RS 4n
1211 Enables or disables the marking of dynamic dependencies to be lazily loaded.
1212 Dynamic dependencies which are marked \fBlazyload\fR are not loaded at initial
1213 process start-up. These dependencies are delayed until the first binding to the
1214 object is made. \fBNote:\fR Lazy loading requires the correct declaration of
1215 dependencies, together with associated runpaths for each dynamic object used
1216 within a process. See \fILazy Loading of Dynamic Dependencies\fR in \fILinker
1217 and Libraries Guide\fR.
1218 .RE

1220 .sp
1221 .ne 2
1222 .na
1223 \fB\fB-z\fR \fBld32\fR=\fIarg1\fR,\fIarg2\fR,...\fR
1224 .ad
1225 .br
1226 .na
1227 \fB\fB-z\fR \fBld64\fR=\fIarg1\fR,\fIarg2\fR,...\fR
1228 .ad
1229 .sp .6
1230 .RS 4n
1231 The class of the link-editor is affected by the class of the output file being
1232 created and by the capabilities of the underlying operating system. The
1233 \fB-z\fR \fBld\fR[\fB32\fR|\fB64\fR] options provide a means of defining any
1234 link-editor argument. The defined argument is only interpreted, respectively,
1235 by the 32-bit class or 64-bit class of the link-editor.
1236 .sp
1237 For example, support libraries are class specific, so the correct class of
1238 support library can be ensured using:
1239 .sp
1240 .in +2

```

```

1241 .nf
1242 \fBld ... -z ld32=-Saudit32.so.1 -z ld64=-Saudit64.so.1 ... \fR
1243 .fi
1244 .in -2
1245 .sp

1247 The class of link-editor that is invoked is determined from the \fBELF\fR class
1248 of the first relocatable file that is seen on the command line. This
1249 determination is carried out \fBprior\fR to any \fB-z\fR
1250 \fBld\fR[\fB32\fR|\fB64\fR] processing.
1251 .RE

1253 .sp
1254 .ne 2
1255 .na
1256 \fB\fB-z\fR \fBloadfltr\fR\fR
1257 .ad
1258 .sp .6
1259 .RS 4n
1260 Marks a filter to indicate that filtees must be processed immediately at
1261 runtime. Normally, filter processing is delayed until a symbol reference is
1262 bound to the filter. The runtime processing of an object that contains this
1263 flag mimics that which occurs if the \fBLD_LOADFLTR\fR environment variable is
1264 in effect. See the \fBld.so.1\fR(1).
1265 .RE

1267 .sp
1268 .ne 2
1269 .na
1270 \fB\fB-z\fR \fBmuldefs\fR\fR
1271 .ad
1272 .br
1273 .na
1274 \fB\fB--allow-multiple-definition\fR\fR
1275 .ad
1276 .sp .6
1277 .RS 4n
1278 Allows multiple symbol definitions. By default, multiple symbol definitions
1279 that occur between relocatable objects result in a fatal error condition. This
1280 option, suppresses the error condition, allowing the first symbol definition to
1281 be taken.
1282 .RE

1284 .sp
1285 .ne 2
1286 .na
1287 \fB\fB-z\fR \fBnocmpstrtab\fR\fR
1288 .ad
1289 .sp .6
1290 .RS 4n
1291 Disables the compression of \fBELF\fR string tables. By default, string
1292 compression is applied to \fBSHT_STRTAB\fR sections, and to \fBSHT_PROGBITS\fR
1293 sections that have their \fBSHF_MERGE\fR and \fBSHF_STRINGS\fR section flags
1294 set.
1295 .RE

1297 .sp
1298 .ne 2
1299 .na
1300 \fB\fB-z\fR \fBnodefaultlib\fR\fR
1301 .ad
1302 .sp .6
1303 .RS 4n
1304 Marks the object so that the runtime default library search path, used after
1305 any \fBLD_LIBRARY_PATH\fR or runpaths, is ignored. This option implies that all
1306 dependencies of the object can be satisfied from its runpath.

```

```

1307 .RE
1309 .sp
1310 .ne 2
1311 .na
1312 \fB\fB-z\fR \fBnodelate\fR\fR
1313 .ad
1314 .sp .6
1315 .RS 4n
1316 Marks the object as non-deletable at runtime. This mode is similar to adding
1317 the object to the process by using \fBdlopen\fR(3C) with the
1318 \fBRTLD_NODELETE\fR mode.
1319 .RE
1321 .sp
1322 .ne 2
1323 .na
1324 \fB\fB-z\fR \fBnodlopen\fR\fR
1325 .ad
1326 .sp .6
1327 .RS 4n
1328 Marks the object as not available to \fBdlopen\fR(3C), either as the object
1329 specified by the \fBdlopen()\fR, or as any form of dependency required by the
1330 object specified by the \fBdlopen()\fR. This option is only meaningful when
1331 building a shared object.
1332 .RE
1334 .sp
1335 .ne 2
1336 .na
1337 \fB\fB-z\fR \fBnodump\fR\fR
1338 .ad
1339 .sp .6
1340 .RS 4n
1341 Marks the object as not available to \fBldump\fR(3C).
1342 .RE
1344 .sp
1345 .ne 2
1346 .na
1347 \fB\fB-z\fR \fBnoldynsym\fR\fR
1348 .ad
1349 .sp .6
1350 .RS 4n
1351 Prevents the inclusion of a \fB&.SUNW_ldynsym\fR section in dynamic
1352 executables or sharable libraries. The \fB&.SUNW_ldynsym\fR section augments
1353 the \fB&.dynsym\fR section by providing symbols for local functions. Local
1354 function symbols allow debuggers to display local function names in stack
1355 traces from stripped programs. Similarly, \fBldaddr\fR(3C) is able to supply
1356 more accurate results.
1357 .sp
1358 The \fB-z\fR \fBnoldynsym\fR option also prevents the inclusion of the two
1359 symbol sort sections that are related to the \fB&.SUNW_ldynsym\fR section. The
1360 \fB&.SUNW_dynsymsort\fR section provides sorted access to regular function and
1361 variable symbols. The \fB&.SUNW_dyntlssort\fR section provides sorted access
1362 to thread local storage (\fBTLS\fR) variable symbols.
1363 .sp
1364 The \fB&.SUNW_ldynsym\fR, \fB&.SUNW_dynsymsort\fR, and
1365 \fB&.SUNW_dyntlssort\fR sections, which becomes part of the allocable text
1366 segment of the resulting file, cannot be removed by \fBstrip\fR(1). Therefore,
1367 the \fB-z\fR \fBnoldynsym\fR option is the only way to prevent their inclusion.
1368 See the \fB-s\fR and \fB-z\fR \fBbredlocs\fR options.
1369 .RE
1371 .sp
1372 .ne 2

```

```

1373 .na
1374 \fB\fB-z\fR \fBnopartial\fR\fR
1375 .ad
1376 .sp .6
1377 .RS 4n
1378 Partially initialized symbols, that are defined within relocatable object
1379 files, are expanded in the output file being generated.
1380 .RE
1382 .sp
1383 .ne 2
1384 .na
1385 \fB\fB-z\fR \fBnoverion\fR\fR
1386 .ad
1387 .sp .6
1388 .RS 4n
1389 Does not record any versioning sections. Any version sections or associated
1390 \fB&.dynamic\fR section entries are not generated in the output image.
1391 .RE
1393 .sp
1394 .ne 2
1395 .na
1396 \fB\fB-z\fR \fBnow\fR\fR
1397 .ad
1398 .sp .6
1399 .RS 4n
1400 Marks the object as requiring non-lazy runtime binding. This mode is similar to
1401 adding the object to the process by using \fBdlopen\fR(3C) with the
1402 \fBRTLD_NOW\fR mode. This mode is also similar to having the \fBBLD_BIND_NOW\fR
1403 environment variable in effect. See \fBld.so.1\fR(1).
1404 .RE
1406 .sp
1407 .ne 2
1408 .na
1409 \fB\fB-z\fR \fBborigin\fR\fR
1410 .ad
1411 .sp .6
1412 .RS 4n
1413 Marks the object as requiring immediate \fB$ORIGIN\fR processing at runtime.
1414 This option is only maintained for historic compatibility, as the runtime
1415 analysis of objects to provide for \fB$ORIGIN\fR processing is now default.
1416 .RE
1418 .sp
1419 .ne 2
1420 .na
1421 \fB\fB-z\fR \fBpreinitarray=\fR\fIfunction\fR\fR
1422 .ad
1423 .sp .6
1424 .RS 4n
1425 Appends an entry to the \fB&.preinitarray\fR section of the object being
1426 built. If no \fB&.preinitarray\fR section is present, a section is created.
1427 The new entry is initialized to point to \fIfunction\fR. See \fIInitialization
1428 and Termination Sections\fR in \fILinker and Libraries Guide\fR.
1429 .RE
1431 .sp
1432 .ne 2
1433 .na
1434 \fB\fB-z\fR \fBbredlocs\fR\fR
1435 .ad
1436 .sp .6
1437 .RS 4n
1438 Eliminates all local symbols except for the \fBISECT\fR symbols from the symbol

```

1439 table \fBSHT_SYMTAB\fR. All relocations that refer to local symbols are updated
 1440 to refer to the corresponding \fISECT\fR symbol. This option allows specialized
 1441 objects to greatly reduce their symbol table sizes. Eliminated local symbols
 1442 can reduce the \fB&.stab*\fR debugging information that is generated using the
 1443 compiler drivers \fB-g\fR option. See the \fB-s\fR and \fB-z\fR \fBnoldynsym\fR
 1444 options.
 1445 .RE

1447 .sp
 1448 .ne 2
 1449 .na
 1450 \fB\fB-z\fR \fBrelaxreloc\fR
 1451 .ad
 1452 .sp .6
 1453 .RS 4n
 1454 \fBld\fR normally issues a fatal error upon encountering a relocation using a
 1455 symbol that references an eliminated COMDAT section. If \fB-z\fR
 1456 \fBrelaxreloc\fR is enabled, \fBld\fR instead redirects such relocations to the
 1457 equivalent symbol in the COMDAT section that was kept. \fB-z\fR
 1458 \fBrelaxreloc\fR is a specialized option, mainly of interest to compiler
 1459 authors, and is not intended for general use.
 1460 .RE

1462 .sp
 1463 .ne 2
 1464 .na
 1465 \fB\fB-z\fR \fBrescan-now\fR
 1466 .ad
 1467 .br
 1468 .na
 1469 \fB\fB-z\fR \fBrescan\fR
 1470 .ad
 1471 .sp .6
 1472 .RS 4n
 1473 These options rescan the archive files that are provided to the link-edit. By
 1474 default, archives are processed once as the archives appear on the command
 1475 line. Archives are traditionally specified at the end of the command line so
 1476 that their symbol definitions resolve any preceding references. However,
 1477 specifying archives multiple times to satisfy their own interdependencies can
 1478 be necessary.
 1479 .sp
 1480 \fB-z\fR \fBrescan-now\fR is a positional option, and is processed by the
 1481 link-editor immediately when encountered on the command line. All archives seen
 1482 on the command line up to that point are immediately reprocessed in an attempt
 1483 to locate additional archive members that resolve symbol references. This
 1484 archive rescanning is repeated until a pass over the archives occurs in which
 1485 no new members are extracted.
 1486 .sp
 1487 \fB-z\fR \fBrescan\fR is a position independent option. The link-editor defers
 1488 the rescan operation until after it has processed the entire command line, and
 1489 then initiates a final rescan operation over all archives seen on the command
 1490 line. The \fB-z\fR \fBrescan\fR operation can interact incorrectly
 1491 with objects that contain initialization (.init) or finalization (.fini)
 1492 sections, preventing the code in those sections from running. For this reason,
 1493 \fB-z\fR \fBrescan\fR is deprecated, and use of \fB-z\fR \fBrescan-now\fR is
 1494 advised.
 1495 .RE

1497 .sp
 1498 .ne 2
 1499 .na
 1500 \fB\fB-z\fR \fBrescan-start\fR ... \fB-z\fR \fBrescan-end\fR
 1501 .ad
 1502 .br
 1503 .na
 1504 \fB\fB--start-group\fR ... \fB--end-group\fR

1505 .ad
 1506 .br
 1507 .na
 1508 \fB\fB-(\fR ... \fB-)\fR
 1509 .ad
 1510 .sp .6
 1511 .RS 4n
 1512 Defines an archive rescan group. This is a positional construct, and is
 1513 processed by the link-editor immediately upon encountering the closing
 1514 delimiter option. Archives found within the group delimiter options are
 1515 reprocessed as a group in an attempt to locate additional archive members that
 1516 resolve symbol references. This archive rescanning is repeated until a pass
 1517 over the archives occurs in which no new members are extracted.
 1518 Archive rescan groups cannot be nested.
 1519 .RE

1521 .sp
 1522 .ne 2
 1523 .na
 1524 \fB\fB-z\fR \fBtarget=sparc|x86\fR \fI\fR
 1525 .ad
 1526 .sp .6
 1527 .RS 4n
 1528 Specifies the machine type for the output object. Supported targets are Sparc
 1529 and x86. The 32-bit machine type for the specified target is used unless the
 1530 \fB-64\fR option is also present, in which case the corresponding 64-bit
 1531 machine type is used. By default, the machine type of the object being
 1532 generated is determined from the first \fBELF\fR object processed from the
 1533 command line. If no objects are specified, the machine type is determined by
 1534 the first object encountered within the first archive processed from the
 1535 command line. If there are no objects or archives, the link-editor assumes the
 1536 native machine. This option is useful when creating an object directly with
 1537 \fBld\fR whose input is solely from a \fBmapfile\fR. See the \fB-M\fR option.
 1538 It can also be useful in the rare case of linking entirely from an archive that
 1539 contains objects of different machine types for which the first object is not
 1540 of the desired machine type. See \fIThe 32-bit link-editor and 64-bit
 1541 link-editor\fR in \fILinker and Libraries Guide\fR.
 1542 .RE

1544 .sp
 1545 .ne 2
 1546 .na
 1547 \fB\fB-z\fR \fBtext\fR
 1548 .ad
 1549 .sp .6
 1550 .RS 4n
 1551 In dynamic mode only, forces a fatal error if any relocations against
 1552 non-writable, allocatable sections remain. For historic reasons, this mode is
 1553 not the default when building an executable or shared object. However, its use
 1554 is recommended to ensure that the text segment of the dynamic object being
 1555 built is shareable between multiple running processes. A shared text segment
 1556 incurs the least relocation overhead when loaded into memory. See
 1557 \fIPosition-Independent Code\fR in \fILinker and Libraries Guide\fR.
 1558 .RE

1560 .sp
 1561 .ne 2
 1562 .na
 1563 \fB\fB-z\fR \fBtextoff\fR
 1564 .ad
 1565 .sp .6
 1566 .RS 4n
 1567 In dynamic mode only, allows relocations against all allocatable sections,
 1568 including non-writable ones. This mode is the default when building a shared
 1569 object.
 1570 .RE

```

1572 .sp
1573 .ne 2
1574 .na
1575 \fB\fB-z\fR \fBtextwarn\fR\fR
1576 .ad
1577 .sp .6
1578 .RS 4n
1579 In dynamic mode only, lists a warning if any relocations against non-writable,
1580 allocatable sections remain. This mode is the default when building an
1581 executable.
1582 .RE

1584 .sp
1585 .ne 2
1586 .na
1587 \fB\fB-z\fR \fBverbose\fR\fR
1588 .ad
1589 .sp .6
1590 .RS 4n
1591 This option provides additional warning diagnostics during a link-edit.
1592 Presently, this option conveys suspicious use of displacement relocations. This
1593 option also conveys the restricted use of static \fBTLS\fR relocations when
1594 building shared objects. In future, this option might be enhanced to provide
1595 additional diagnostics that are deemed too noisy to be generated by default.
1596 .RE

1598 .sp
1599 .ne 2
1600 .na
1601 \fB\fB-z\fR \fBwrap=\fR \fIsymbol\fR\fR
1602 .ad
1603 .br
1604 .na
1605 \fB\fB-wrap=\fR \fIsymbol\fR\fR
1606 .ad
1607 .br
1608 .na
1609 \fB\fB--wrap=\fR \fIsymbol\fR\fR
1610 .ad
1611 .sp .6
1612 .RS 4n
1613 Rename undefined references to \fIsymbol\fR in order to allow wrapper code to
1614 be linked into the output object without having to modify source code. When
1615 \fB-z wrap\fR is specified, all undefined references to \fIsymbol\fR are
1616 modified to reference \fB__wrap_\fR \fIsymbol\fR, and all references to
1617 \fB__real_\fR \fIsymbol\fR are modified to reference \fIsymbol\fR. The user is
1618 expected to provide an object containing the \fB__wrap_\fR \fIsymbol\fR
1619 function. This wrapper function can call \fB__real_\fR \fIsymbol\fR in order to
1620 reference the actual function being wrapped.
1621 .sp
1622 The following is an example of a wrapper for the \fBmalloc\fR(3C) function:
1623 .sp
1624 .in +2
1625 .nf
1626 void *
1627 __wrap_malloc(size_t c)
1628 {
1629     (void) printf("malloc called with %zu\n", c);
1630     return (__real_malloc(c));
1631 }
1632 .fi
1633 .in -2

1635 If you link other code with this file using \fB-z\fR \fBwrap=malloc\fR to
1636 compile all the objects, then all calls to \fBmalloc\fR will call the function

```

```

1637 \fB__wrap_malloc\fR instead. The call to \fB__real_malloc\fR will call the real
1638 \fBmalloc\fR function.
1639 .sp
1640 The real and wrapped functions should be maintained in separate source files.
1641 Otherwise, the compiler or assembler may resolve the call instead of leaving
1642 that operation for the link-editor to carry out, and prevent the wrap from
1643 occurring.
1644 .RE

1646 .SH ENVIRONMENT VARIABLES
1647 .sp
1648 .ne 2
1649 .na
1649 \fB\fBLD_ALTEXEC\fR\fR
1650 .ad
1651 .sp .6
1652 .RS 4n
1653 An alternative link-editor path name. \fBld\fR executes, and passes control to
1654 this alternative link-editor. This environment variable provides a generic
1655 means of overriding the default link-editor that is called from the various
1656 compiler drivers. See the \fB-z altextec64\fR option.
1657 .RE

1659 .sp
1660 .ne 2
1661 .na
1662 \fB\fBLD_LIBRARY_PATH\fR\fR
1663 .ad
1664 .sp .6
1665 .RS 4n
1666 A list of directories in which to search for the libraries specified using the
1667 \fB-L\fR option. Multiple directories are separated by a colon. In the most
1668 general case, this environment variable contains two directory lists separated
1669 by a semicolon:
1670 .sp
1671 .in +2
1672 .nf
1673 \fIdirlist1\fR \fB;\fR \fIdirlist2\fR
1674 .fi
1675 .in -2
1676 .sp

1678 If \fBld\fR is called with any number of occurrences of \fB-L\fR, as in:
1679 .sp
1680 .in +2
1681 .nf
1682 \fBld ... -L\fIpath1\fR ... -L\fIpathn\fR ... \fR
1683 .fi
1684 .in -2
1685 .sp

1687 then the search path ordering is:
1688 .sp
1689 .in +2
1690 .nf
1691 \fB\fIdirlist1 path1\fR ... \fIpathn dirlist2\fR LIBPATH\fR
1692 .fi
1693 .in -2
1694 .sp

1696 When the list of directories does not contain a semicolon, the list is
1697 interpreted as \fIdirlist2\fR.
1698 .sp
1699 The \fBLD_LIBRARY_PATH\fR environment variable also affects the runtime linkers
1700 search for dynamic dependencies.
1701 .sp

```

1702 This environment variable can be specified with a `_32` or `_64` suffix. This makes
 1703 the environment variable specific, respectively, to 32-bit or 64-bit processes
 1704 and overrides any non-suffixed version of the environment variable that is in
 1705 effect.
 1706 .RE

1708 .sp
 1709 .ne 2
 1710 .na
 1711 \fB\fBLD_NOEXEC_64\fR\fR
 1712 .ad
 1713 .sp .6
 1714 .RS 4n
 1715 Suppresses the automatic execution of the 64-bit link-editor. By default, the
 1716 link-editor executes the 64-bit version when the `\fBELF\fR` class of the first
 1717 relocatable file identifies a 64-bit object. The 64-bit image that a 32-bit
 1718 link-editor can create, has some limitations. However, some link-edits might
 1719 find the use of the 32-bit link-editor faster.
 1720 .RE

1722 .sp
 1723 .ne 2
 1724 .na
 1725 \fB\fBLD_OPTIONS\fR\fR
 1726 .ad
 1727 .sp .6
 1728 .RS 4n
 1729 A default set of options to `\fBld\fR`. `\fBLD_OPTIONS\fR` is interpreted by
 1730 `\fBld\fR` just as though its value had been placed on the command line,
 1731 immediately following the name used to invoke `\fBld\fR`, as in:
 1732 .sp
 1733 .in +2
 1734 .nf
 1735 \fBld \$LD_OPTIONS ... \fIother-arguments\fR ... \fR
 1736 .fi
 1737 .in -2
 1738 .sp

1740 .RE

1742 .sp
 1743 .ne 2
 1744 .na
 1745 \fB\fBLD_RUN_PATH\fR\fR
 1746 .ad
 1747 .sp .6
 1748 .RS 4n
 1749 An alternative mechanism for specifying a runpath to the link-editor. See the
 1750 `\fB-R\fR` option. If both `\fBLD_RUN_PATH\fR` and the `\fB-R\fR` option are
 1751 specified, `\fB-R\fR` supersedes.
 1752 .RE

1754 .sp
 1755 .ne 2
 1756 .na
 1757 \fB\fBSGS_SUPPORT\fR\fR
 1758 .ad
 1759 .sp .6
 1760 .RS 4n
 1761 Provides a colon-separated list of shared objects that are loaded with the
 1762 link-editor and given information regarding the linking process. This
 1763 environment variable can be specified with a `_32` or `_64` suffix. This makes the
 1764 environment variable specific, respectively, to the 32-bit or 64-bit class of
 1765 `\fBld\fR` and overrides any non-suffixed version of the environment variable
 1766 that is in effect. See the `\fB-S\fR` option.
 1767 .RE

1769 .sp
 1770 .LP
 1771 Notice that environment variable-names that begin with the
 1772 characters `'\fBLD_\fR'` are reserved for possible future enhancements to `\fBld\fR`
 1773 `\fBld.so.1\fR(1)`.
 1774 .SH FILES
 1775 .sp
 1776 .ne 2
 1777 .na
 1778 \fB\fBlib\fIx\fR.so\fR\fR
 1779 .ad
 1780 .RS 15n
 1781 shared object libraries.
 1782 .RE

1783 .sp
 1784 .ne 2
 1785 .na
 1786 \fB\fBlib\fIx\fR.a\fR\fR
 1787 .ad
 1788 .RS 15n
 1789 archive libraries.
 1790 .RE

1792 .sp
 1793 .ne 2
 1794 .na
 1795 \fB\fBa.out\fR\fR
 1796 .ad
 1797 .RS 15n
 1798 default output file.
 1799 .RE

1801 .sp
 1802 .ne 2
 1803 .na
 1804 \fB\fLILIBPATH\fR\fR
 1805 .ad
 1806 .RS 15n
 1807 For 32-bit libraries, the default search path is `\fB/usr/ccs/lib\fR`, followed
 1808 by `\fB/lib\fR`, and finally `\fB/usr/lib\fR`. For 64-bit libraries, the default
 1809 search path is `\fB/lib/64\fR`, followed by `\fB/usr/lib/64\fR`.
 1810 .RE

1812 .sp
 1813 .ne 2
 1814 .na
 1815 \fB\fB/usr/lib/ld\fR\fR
 1816 .ad
 1817 .RS 15n
 1818 A directory containing several `\fBmapfiles\fR` that can be used during
 1819 link-editing. These `\fBmapfiles\fR` provide various capabilities, such as
 1820 defining memory layouts, aligning bss, and defining non-executable stacks.
 1821 .RE

1823 .SH ATTRIBUTES
 1824 .sp
 1825 See `\fBattributes\fR(5)` for descriptions of the following attributes:
 1826 .sp

1828 .sp
 1829 .TS
 1830 box;
 1831 c | c


```
1832 l | l .
1833 ATTRIBUTE TYPE    ATTRIBUTE VALUE
1834 -
1835 Interface Stability    Committed
1836 .TE

1838 .SH SEE ALSO
1839 .sp
1840 \fBas\fR(1), \fBcrle\fR(1), \fBgprof\fR(1), \fBld.so.1\fR(1), \fBldd\fR(1),
1841 \fBmcs\fR(1), \fBpvs\fR(1), \fBpexec\fR(2), \fBstat\fR(2), \fBdlopen\fR(3C),
1842 \fBlddump\fR(3C), \fBelf\fR(3ELF), \fBbar.h\fR(3HEAD), \fBa.out\fR(4),
1843 \fBattributes\fR(5)
1844 .sp
1845 .LP
1846 \fILinker and Libraries Guide\fR
1847 .SH NOTES
1848 .sp
1849 .LP
1849 Default options applied by \fBld\fR are maintained for historic reasons. In
1850 today's programming environment, where dynamic objects dominate, alternative
1851 defaults would often make more sense. However, historic defaults must be
1852 maintained to ensure compatibility with existing program development
1853 environments. Historic defaults are called out wherever possible in this
1854 manual. For a description of the current recommended options, see Appendix A,
1855 \fILink-Editor Quick Reference,\fR in \fILinker and Libraries Guide\fR.
1856 .sp
1857 .LP
1858 If the file being created by \fBld\fR already exists, the file is unlinked
1859 after all input files have been processed. A new file with the specified name
1860 is then created. This allows \fBld\fR to create a new version of the file,
1861 while simultaneously allowing existing processes that are accessing the old
1862 file contents to continue running. If the old file has no other links, the disk
1863 space of the removed file is freed when the last process referencing the file
1864 terminates.
1865 .sp
1866 .LP
1867 The behavior of \fBld\fR when the file being created already exists was changed
1868 with \fBSXCE\fR build \fB43\fR. In older versions, the existing file was
1869 rewritten in place, an approach with the potential to corrupt any running
1870 processes that is using the file. This change has an implication for output
1871 files that have multiple hard links in the file system. Previously, all links
1872 would remain intact, with all links accessing the new file contents. The new
1873 \fBld\fR behavior \fBbreaks\fR such links, with the result that only the
1874 specified output file name references the new file. All the other links
1875 continue to reference the old file. To ensure consistent behavior, applications
1876 that rely on multiple hard links to linker output files should explicitly
1877 remove and relink the other file names.
```

```

*****
5791 Thu Jun 30 22:02:40 2016
new/usr/src/man/man1/psecflags.1
Code review comments from jeffpc
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
1 \" te
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 2015, Richard Lowe.
12.\"
13.TH \"PSECFLAGS\" \"1\" \"June 6, 2016\"
14.SH \"NAME\"
15 \fBpsecflags\fR - inspect or modify process security flags
16.SH \"SYNOPSIS\"
17.LP
18.nf
19 \fB/usr/bin/psecflags\fR \fI-s\fR \fI-spec\fR \fI-e\fR \fI-command\fR \
20 [\fIarg\fR]...
21.fi
22.LP
23.nf
24 \fB/usr/bin/psecflags\fR \fI-s\fR \fI-spec\fR [\fI-i\fR \fIidtype\fR] \
25 \fIid\fR ...
26.fi
27.LP
28.nf
29 \fB/usr/bin/psecflags\fR [\fI-F\fR] { \fIpid\fR | \fIcore\fR }
30.fi
31.LP
32.nf
33 \fB/usr/bin/psecflags\fR \fI-l\fR
34.fi

36.SH \"DESCRIPTION\"
37 The first invocation of the \fBpsecflags\fR command runs the specified
38 \fIcommand\fR with the security-flags modified as described by the \fI-s\fR
39 argument.
40 .P
41 The second invocation modifies the security-flags of the processes described
42 by \fIidtype\fR and \fIid\fR according as described by the \fI-s\fR argument.
43 .P
44 The third invocation describes the security-flags of the specified processes
45 or core files. The effective set is signified by '\fBE\fR', the inheritable
46 set by '\fBI\fR', the lower set by '\fBL\fR', and the upper set by '\fBU\fR'.
47 .P
48 The fourth invocation lists the supported process security-flags, documented
49 in \fBsecurity-flags\fR(5).

51.SH \"OPTIONS\"
52 The following options are supported:
53 .sp
54 .ne 2
55 .na
56 \fB-e\fR
57 .ad

```

```

58 .RS 11n
59 Interpret the remaining arguments as a command line and run the command with
60 the security-flags specified with the \fI-s\fR flag.
61 .RE

63 .sp
64 .ne 2
65 .na
66 \fB-F\fR
67 .ad
68 .RS 11n
69 Force. Grab the target process even if another process has control.
70 .RE

72 .sp
73 .ne 2
74 .na
75 \fB-i\fR \fIidtype\fR
76 .ad
77 .RS 11n
78 This option, together with the \fIid\fR arguments specify one or more
79 processes whose security-flags will be modified. The interpretation of the
80 \fIid\fR arguments is based on \fIidtype\fR. If \fIidtype\fR is omitted the
81 default is \fBpid\fR.

83 Valid \fIidtype\fR options are:
84 .sp
85 .ne 2
86 .na
87 \fBBall\fR
88 .ad
89 .RS 11n
90 The \fBpsecflags\fR command applies to all processes
91 .RE

93 .sp
94 .ne 2
95 .na
96 \fBContract\fR, \fBctid\fR
97 .ad
98 .RS 11n
99 The security-flags of any process with a contract ID matching the \fIid\fR
100 arguments are modified.
101 .RE

103 .sp
104 .ne 2
105 .na
106 \fBGroup\fR, \fBgid\fR
107 .ad
108 .RS 11n
109 The security-flags of any process with a group ID matching the \fIid\fR
110 arguments are modified.
111 .RE

113 .sp
114 .ne 2
115 .na
116 \fBpid\fR
117 .ad
118 .RS 11n
119 The security-flags of any process with a process ID matching the \fIid\fR
120 arguments are modified. This is the default.
121 .RE

123 .sp

```

```

124 .ne 2
125 .na
126 \fBppid\fR
127 .ad
128 .RS 11n
129 The security-flags of any processes whose parent process ID matches the
130 \fIid\fR arguments are modified.
131 .RE

133 .sp
134 .ne 2
135 .na
136 \fBproject\fR, \fBprojid\fR
137 .ad
138 .RS 11n
139 The security-flags of any process whose project ID matches the \fIid\fR
140 arguments are modified.
141 .RE

143 .sp
144 .ne 2
145 .na
146 \fBsession\fR, \fBsid\fR
147 .ad
148 .RS 11n
149 The security-flags of any process whose session ID matches the \fIid\fR
150 arguments are modified.
151 .RE

153 .sp
154 .ne 2
155 .na
156 \fBtaskid\fR
157 .ad
158 .RS 11n
159 The security-flags of any process whose task ID matches the \fIid\fR arguments
160 are modified.
161 .RE

163 .sp
164 .ne 2
165 .na
166 \fBuser\fR, \fBuid\fR
167 .ad
168 .RS 11n
169 The security-flags of any process belonging to the users matching the \fIid\fR
170 arguments are modified.
171 .RE

173 .sp
174 .ne 2
175 .na
176 \fBzone\fR, \fBzoneid\fR
177 .ad
178 .RS 11n
179 The security-flags of any process running in the zones matching the given
180 \fIid\fR arguments are modified.
181 .RE
182 .RE

184 .sp
185 .ne 2
186 .na
187 \fB-1\fR
188 .ad
189 .RS 11n

```

```

190 List all supported process security-flags, described in
191 \fBsecurity-flags\fR(5).
192 .RE

194 .sp
195 .ne 2
196 .na
197 \fB-s\fR \fIspecification\fR
198 .ad
199 .RS 11n
200 Modify the process security-flags according to
201 \fIspecification\fR. Specifications take the form of a comma-separated list of
202 flags, optionally preceded by a '-' or '!'. Where '-' and '!' indicate that the
203 given flag should be removed from the specification. The pseudo-flags "all",
204 "none" and "current" are supported, to indicate that all flags, no flags, or
205 the current set of flags (respectively) are to be included.
206 .P
207 By default, the inheritable flags are changed. You may optionally specify the
208 set to change using their single-letter identifiers and an equals sign.
209 .P
210 For a list of valid security-flags, see \fBpsecflags -1\fR.
211 .RE

213 .SH "EXAMPLES"
214 .LP
215 \fBExample 1\fR Display the security-flags of the current shell.
216 .sp
217 .in +2
218 .nf
219 example$ \fBpsecflags $$\fR
220 100718: -sh
221      E:      aslr
222      I:      aslr
223      L:      none
224      U:      aslr,forbidnullmap,noexecstack
225 .fi
226 .in -2
227 .sp

229 .LP
230 \fBExample 2\fR Run a user command with ASLR enabled in addition to any
231 inherited security flags.
232 .sp
233 .in +2
234 .nf
235 example$ \fBpsecflags -s current,aslr -e /bin/sh\fR
236 $ psecflags $$
237 100724: -sh
238      E:      none
239      I:      aslr
240      L:      none
241      U:      aslr,forbidnullmap,noexecstack
242 .fi
243 .in -2
244 .sp

246 .LP
247 \fBExample 3\fR Remove aslr from the inheritable flags of all Bob's processes.
248 .sp
249 .in +2
250 .nf
251 example# \fBpsecflags -s current,-aslr -i uid bob\fR
252 .fi
253 .in -2

255 .LP

```

```
256 \fBExample 4\fR Add the aslr flag to the lower set, so that all future
257 child processes must have this flag set.
258 .sp
259 .in +2
260 .nf
261 example# \fBpsecflags -s L=current,aslr $$\fR
262 .fi
263 .in -2

265 .SH "EXIT STATUS"
266 The following exit values are returned:

268 .TP
269 \fB0\fR
270 .IP
271 Success.

273 .TP
274 \fBnon-zero\fR
275 .IP
276 An error has occurred.

278 .SH "ATTRIBUTES"
279 .LP
280 See \fBattributes\fR(5) for descriptions of the following attributes:
281 .sp

283 .sp
284 .TS
285 box;
286 c | c
287 l | l .
288 ATTRIBUTE TYPE    ATTRIBUTE VALUE
289 -
290 Interface Stability    Volatile
291 .TE

293 .SH "SEE ALSO"
294 .BR exec (2),
295 .BR attributes (5),
296 .BR contract (4),
297 .BR security-flags (5),
298 .BR zones (5)
299 #endif /* ! codereview */
```

```

*****
44021 Thu Jun 30 22:02:41 2016
new/usr/src/man/man1m/zonecfg.1m
Code review comments from jeffpc
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
1 \" te
2 .\\ Copyright (c) 2004, 2009 Sun Microsystems, Inc. All Rights Reserved.
3 .\\ Copyright 2013 Joyent, Inc. All Rights Reserved.
4 .\\ The contents of this file are subject to the terms of the Common Development
5 .\\ See the License for the specific language governing permissions and limitati
6 .\\ fields enclosed by brackets \"[]\" replaced with your own identifying informat
7 .TH ZONECFG 1M \"Jun 6, 2016\"
8 .TH ZONECFG 1M \"Feb 28, 2014\"
9 .SH NAME
9 zonecfg \- set up zone configuration
10 .SH SYNOPSIS
11 .LP
12 .nf
13 \\fbzonecfg\\fR \\fB-z\\fR \\fIzonename\\fR
14 .fi

16 .LP
17 .nf
18 \\fbzonecfg\\fR \\fB-z\\fR \\fIzonename\\fR \\fIsubcommand\\fR
19 .fi

21 .LP
22 .nf
23 \\fbzonecfg\\fR \\fB-z\\fR \\fIzonename\\fR \\fB-f\\fR \\fIcommand_file\\fR
24 .fi

26 .LP
27 .nf
28 \\fbzonecfg\\fR help
29 .fi

31 .SH DESCRIPTION
32 .sp
32 .LP
33 The \\fbzonecfg\\fR utility creates and modifies the configuration of a zone.
34 Zone configuration consists of a number of resources and properties.
35 .sp
36 .LP
37 To simplify the user interface, \\fbzonecfg\\fR uses the concept of a scope. The
38 default scope is global.
39 .sp
40 .LP
41 The following synopsis of the \\fbzonecfg\\fR command is for interactive usage:
42 .sp
43 .in +2
44 .nf
45 zonecfg \\fB-z\\fR \\fIzonename subcommand\\fR
46 .fi
47 .in -2
48 .sp

50 .sp
51 .LP
52 Parameters changed through \\fbzonecfg\\fR do not affect a running zone. The zone
53 must be rebooted for the changes to take effect.
54 .sp
55 .LP

```

```

56 In addition to creating and modifying a zone, the \\fbzonecfg\\fR utility can
57 also be used to persistently specify the resource management settings for the
58 global zone.
59 .sp
60 .LP
61 In the following text, "rctl" is used as an abbreviation for "resource
62 control". See \\fBresource_controls\\fR(5).
63 .sp
64 .LP
65 Every zone is configured with an associated brand. The brand determines the
66 user-level environment used within the zone, as well as various behaviors for
67 the zone when it is installed, boots, or is shutdown. Once a zone has been
68 installed the brand cannot be changed. The default brand is determined by the
69 installed distribution in the global zone. Some brands do not support all of
70 the \\fbzonecfg\\fR properties and resources. See the brand-specific man page for
71 more details on each brand. For an overview of brands, see the \\fBbrands\\fR(5)
72 man page.
73 .SS "Resources"
74 .sp
75 .LP
75 The following resource types are supported:
76 .sp
77 .ne 2
78 .na
79 \\fB\\fbattr\\fR\\fR
80 .ad
81 .sp .6
82 .RS 4n
83 Generic attribute.
84 .RE

86 .sp
87 .ne 2
88 .na
89 \\fB\\fbcapped-cpu\\fR\\fR
90 .ad
91 .sp .6
92 .RS 4n
93 Limits for CPU usage.
94 .RE

96 .sp
97 .ne 2
98 .na
99 \\fB\\fbcapped-memory\\fR\\fR
100 .ad
101 .sp .6
102 .RS 4n
103 Limits for physical, swap, and locked memory.
104 .RE

106 .sp
107 .ne 2
108 .na
109 \\fB\\fbdataset\\fR\\fR
110 .ad
111 .sp .6
112 .RS 4n
113 \\fBZFS\\fR dataset.
114 .RE

116 .sp
117 .ne 2
118 .na
119 \\fB\\fbdedicated-cpu\\fR\\fR
120 .ad

```

```

121 .sp .6
122 .RS 4n
123 Subset of the system's processors dedicated to this zone while it is running.
124 .RE

126 .sp
127 .ne 2
128 .na
129 \fB\fBdevice\fR\fR
130 .ad
131 .sp .6
132 .RS 4n
133 Device.
134 .RE

136 .sp
137 .ne 2
138 .na
139 \fB\fBfs\fR\fR
140 .ad
141 .sp .6
142 .RS 4n
143 file-system
144 .RE

146 .sp
147 .ne 2
148 .na
149 \fB\fBnet\fR\fR
150 .ad
151 .sp .6
152 .RS 4n
153 Network interface.
154 .RE

156 .sp
157 .ne 2
158 .na
159 \fB\fBrcctl\fR\fR
160 .ad
161 .sp .6
162 .RS 4n
163 Resource control.
164 .RE

166 .sp
167 .ne 2
168 .na
169 \fB\fBsecurity-flags\fR\fR
170 .ad
171 .sp .6
172 .RS 4n
173 Process security flag settings.
174 .RE

176 #endif /* ! codereview */
177 .SS "Properties"
178 .sp
179 .LP
179 Each resource type has one or more properties. There are also some global
180 properties, that is, properties of the configuration as a whole, rather than of
181 some particular resource.
182 .sp
183 .LP
184 The following properties are supported:
185 .sp

```

```

186 .ne 2
187 .na
188 \fB(global)\fR
189 .ad
190 .sp .6
191 .RS 4n
192 \fBzonename\fR
193 .RE

195 .sp
196 .ne 2
197 .na
198 \fB(global)\fR
199 .ad
200 .sp .6
201 .RS 4n
202 \fBzonepath\fR
203 .RE

205 .sp
206 .ne 2
207 .na
208 \fB(global)\fR
209 .ad
210 .sp .6
211 .RS 4n
212 \fBautoboot\fR
213 .RE

215 .sp
216 .ne 2
217 .na
218 \fB(global)\fR
219 .ad
220 .sp .6
221 .RS 4n
222 \fBbootargs\fR
223 .RE

225 .sp
226 .ne 2
227 .na
228 \fB(global)\fR
229 .ad
230 .sp .6
231 .RS 4n
232 \fBpool\fR
233 .RE

235 .sp
236 .ne 2
237 .na
238 \fB(global)\fR
239 .ad
240 .sp .6
241 .RS 4n
242 \fBlimitpriv\fR
243 .RE

245 .sp
246 .ne 2
247 .na
248 \fB(global)\fR
249 .ad
250 .sp .6
251 .RS 4n

```

```

252 \fBbrand\fR
253 .RE

255 .sp
256 .ne 2
257 .na
258 \fB(global)\fR
259 .ad
260 .sp .6
261 .RS 4n
262 \fBcpu-shares\fR
263 .RE

265 .sp
266 .ne 2
267 .na
268 \fB(global)\fR
269 .ad
270 .sp .6
271 .RS 4n
272 \fBhostid\fR
273 .RE

275 .sp
276 .ne 2
277 .na
278 \fB(global)\fR
279 .ad
280 .sp .6
281 .RS 4n
282 \fBmax-lwps\fR
283 .RE

285 .sp
286 .ne 2
287 .na
288 \fB(global)\fR
289 .ad
290 .sp .6
291 .RS 4n
292 \fBmax-msg-ids\fR
293 .RE

295 .sp
296 .ne 2
297 .na
298 \fB(global)\fR
299 .ad
300 .sp .6
301 .RS 4n
302 \fBmax-sem-ids\fR
303 .RE

305 .sp
306 .ne 2
307 .na
308 \fB(global)\fR
309 .ad
310 .sp .6
311 .RS 4n
312 \fBmax-shm-ids\fR
313 .RE

315 .sp
316 .ne 2
317 .na

```

```

318 \fB(global)\fR
319 .ad
320 .sp .6
321 .RS 4n
322 \fBmax-shm-memory\fR
323 .RE

325 .sp
326 .ne 2
327 .na
328 \fB(global)\fR
329 .ad
330 .sp .6
331 .RS 4n
332 \fB scheduling-class\fR
333 .RE

335 .sp
336 .ne 2
337 .na
338 .B (global)
339 .ad
340 .sp .6
341 .RS 4n
342 .B fs-allowed
343 .RE

345 .sp
346 .ne 2
347 .na
348 \fB\fBfs\fR\fR
349 .ad
350 .sp .6
351 .RS 4n
352 \fBdir\fR, \fBspecial\fR, \fBraw\fR, \fBtype\fR, \fBoptions\fR
353 .RE

355 .sp
356 .ne 2
357 .na
358 \fB\fBnet\fR\fR
359 .ad
360 .sp .6
361 .RS 4n
362 \fBaddress\fR, \fBphysical\fR, \fBdefrouter\fR
363 .RE

365 .sp
366 .ne 2
367 .na
368 \fB\fBdevice\fR\fR
369 .ad
370 .sp .6
371 .RS 4n
372 \fBmatch\fR
373 .RE

375 .sp
376 .ne 2
377 .na
378 \fB\fBctl\fR\fR
379 .ad
380 .sp .6
381 .RS 4n
382 \fBname\fR, \fBvalue\fR
383 .RE

```

```

385 .sp
386 .ne 2
387 .na
388 \fB\fBattr\fR\fR
389 .ad
390 .sp .6
391 .RS 4n
392 \fBname\fR, \fBtype\fR, \fBvalue\fR
393 .RE

395 .sp
396 .ne 2
397 .na
398 \fB\fBdataset\fR\fR
399 .ad
400 .sp .6
401 .RS 4n
402 \fBname\fR
403 .RE

405 .sp
406 .ne 2
407 .na
408 \fB\fBdedicated-cpu\fR\fR
409 .ad
410 .sp .6
411 .RS 4n
412 \fBncpus\fR, \fBimportance\fR
413 .RE

415 .sp
416 .ne 2
417 .na
418 \fB\fBcapped-memory\fR\fR
419 .ad
420 .sp .6
421 .RS 4n
422 \fBphysical\fR, \fBswap\fR, \fBblocked\fR
423 .RE

425 .sp
426 .ne 2
427 .na
428 \fB\fBcapped-cpu\fR\fR
429 .ad
430 .sp .6
431 .RS 4n
432 \fBncpus\fR
433 .RE

435 .sp
436 .ne 2
437 .na
438 \fB\fBsecurity-flags\fB\fB
439 .ad
440 .sp .6
441 .RS 4n
442 \fBblower\fR, \fBdefault\fR, \fBupper\fR.
443 .RE

445 .sp
446 #endif /* ! codereview */
447 .LP
448 As for the property values which are paired with these names, they are either
449 simple, complex, or lists. The type allowed is property-specific. Simple values

```

```

450 are strings, optionally enclosed within quotation marks. Complex values have
451 the syntax:
452 .sp
453 .in +2
454 .nf
455 (<\fIname\fR>=<\fIvalue\fR>,<\fIname\fR>=<\fIvalue\fR>,...)
456 .fi
457 .in -2
458 .sp

460 .sp
461 .LP
462 where each <\fIvalue\fR> is simple, and the <\fIname\fR> strings are unique
463 within a given property. Lists have the syntax:
464 .sp
465 .in +2
466 .nf
467 [<\fIvalue\fR>,...]
468 .fi
469 .in -2
470 .sp

472 .sp
473 .LP
474 where each <\fIvalue\fR> is either simple or complex. A list of a single value
475 (either simple or complex) is equivalent to specifying that value without the
476 list syntax. That is, "foo" is equivalent to "[foo]". A list can be empty
477 (denoted by "[]").
478 .sp
479 .LP
480 In interpreting property values, \fBzoncfg\fR accepts regular expressions as
481 specified in \fBfnmatch\fR(5). See \fBEXAMPLES\fR.
482 .sp
483 .LP
484 The property types are described as follows:
485 .sp
486 .ne 2
487 .na
488 \fBglobal: \fBzonename\fR\fR
489 .ad
490 .sp .6
491 .RS 4n
492 The name of the zone.
493 .RE

495 .sp
496 .ne 2
497 .na
498 \fBglobal: \fBzonepath\fR\fR
499 .ad
500 .sp .6
501 .RS 4n
502 Path to zone's file system.
503 .RE

505 .sp
506 .ne 2
507 .na
508 \fBglobal: \fBautoboot\fR\fR
509 .ad
510 .sp .6
511 .RS 4n
512 Boolean indicating that a zone should be booted automatically at system boot.
513 Note that if the zones service is disabled, the zone will not autoboot,
514 regardless of the setting of this property. You enable the zones service with a
515 \fBsvcadm\fR command, such as:

```



```

516 .sp
517 .in +2
518 .nf
519 # \fBsvcadm enable svc:/system/zones:default\fR
520 .fi
521 .in -2
522 .sp

524 Replace \fBenable\fR with \fBdisable\fR to disable the zones service. See
525 \fBsvcadm\fR(1M).
526 .RE

528 .sp
529 .ne 2
530 .na
531 \fBglobal: \fBbootargs\fR\fR
532 .ad
533 .sp .6
534 .RS 4n
535 Arguments (options) to be passed to the zone bootup, unless options are
536 supplied to the "\fBzoneadm boot\fR" command, in which case those take
537 precedence. The valid arguments are described in \fBzoneadm\fR(1M).
538 .RE

540 .sp
541 .ne 2
542 .na
543 \fBglobal: \fBpool\fR\fR
544 .ad
545 .sp .6
546 .RS 4n
547 Name of the resource pool that this zone must be bound to when booted. This
548 property is incompatible with the \fBdedicated-cpu\fR resource.
549 .RE

551 .sp
552 .ne 2
553 .na
554 \fBglobal: \fBlimitpriv\fR\fR
555 .ad
556 .sp .6
557 .RS 4n
558 The maximum set of privileges any process in this zone can obtain. The property
559 should consist of a comma-separated privilege set specification as described in
560 \fBpriv_str_to_set\fR(3C). Privileges can be excluded from the resulting set by
561 preceding their names with a dash (-) or an exclamation point (!). The special
562 privilege string "zone" is not supported in this context. If the special string
563 "default" occurs as the first token in the property, it expands into a safe set
564 of privileges that preserve the resource and security isolation described in
565 \fBzones\fR(5). A missing or empty property is equivalent to this same set of
566 safe privileges.
567 .sp
568 The system administrator must take extreme care when configuring privileges for
569 a zone. Some privileges cannot be excluded through this mechanism as they are
570 required in order to boot a zone. In addition, there are certain privileges
571 which cannot be given to a zone as doing so would allow processes inside a zone
572 to unduly affect processes in other zones. \fBzoneadm\fR(1M) indicates when an
573 invalid privilege has been added or removed from a zone's privilege set when an
574 attempt is made to either "boot" or "ready" the zone.
575 .sp
576 See \fBprivileges\fR(5) for a description of privileges. The command "\fBpriv
577 -l\fR" (see \fBpriv\fR(1)) produces a list of all Solaris privileges. You can
578 specify privileges as they are displayed by \fBpriv\fR. In
579 \fBprivileges\fR(5), privileges are listed in the form
580 PRIV_\fIprivilege_name\fR. For example, the privilege \fIisys_time\fR, as you
581 would specify it in this property, is listed in \fBprivileges\fR(5) as

```

```

582 \fBPRIV_SYS_TIME\fR.
583 .RE

585 .sp
586 .ne 2
587 .na
588 \fBglobal: \fBbrand\fR\fR
589 .ad
590 .sp .6
591 .RS 4n
592 The zone's brand type.
593 .RE

595 .sp
596 .ne 2
597 .na
598 \fBglobal: \fBip-type\fR\fR
599 .ad
600 .sp .6
601 .RS 4n
602 A zone can either share the IP instance with the global zone, which is the
603 default, or have its own exclusive instance of IP.
604 .sp
605 This property takes the values \fBshared\fR and \fBexclusive\fR.
606 .RE

608 .sp
609 .ne 2
610 .na
611 \fBglobal: \fBhostid\fR\fR
612 .ad
613 .sp .6
614 .RS 4n
615 A zone can emulate a 32-bit host identifier to ease system consolidation. A
616 zone's \fBhostid\fR property is empty by default, meaning that the zone does
617 not emulate a host identifier. Zone host identifiers must be hexadecimal values
618 between 0 and FFFFFFFE. A \fB0x\fR or \fB0X\fR prefix is optional. Both
619 uppercase and lowercase hexadecimal digits are acceptable.
620 .RE

622 .sp
623 .ne 2
624 .na
625 \fB\fBfs\fR: dir, special, raw, type, options\fR
626 .ad
627 .sp .6
628 .RS 4n
629 Values needed to determine how, where, and so forth to mount file systems. See
630 \fBmount\fR(1M), \fBmount\fR(2), \fBfsck\fR(1M), and \fBfstab\fR(4).
631 .RE

633 .sp
634 .ne 2
635 .na
636 \fB\fBnet\fR: address, physical, defrouter\fR
637 .ad
638 .sp .6
639 .RS 4n
640 The network address and physical interface name of the network interface. The
641 network address is one of:
642 .RS +4
643 .TP
644 .ie t \(\bu
645 .el o
646 a valid IPv4 address, optionally followed by "\fB/\fR" and a prefix length;
647 .RE

```

```

648 .RS +4
649 .TP
650 .ie t \(bu
651 .el o
652 a valid IPv6 address, which must be followed by "\fB\fR" and a prefix length;
653 .RE
654 .RS +4
655 .TP
656 .ie t \(bu
657 .el o
658 a host name which resolves to an IPv4 address.
659 .RE
660 Note that host names that resolve to IPv6 addresses are not supported.
661 .sp
662 The physical interface name is the network interface name.
663 .sp
664 The default router is specified similarly to the network address except that it
665 must not be followed by a \fB\fR (slash) and a network prefix length.
666 .sp
667 A zone can be configured to be either exclusive-IP or shared-IP. For a
668 shared-IP zone, you must set both the physical and address properties; setting
669 the default router is optional. The interface specified in the physical
670 property must be plumbed in the global zone prior to booting the non-global
671 zone. However, if the interface is not used by the global zone, it should be
672 configured \fBdown\fR in the global zone, and the default router for the
673 interface should be specified here.
674 .sp
675 For an exclusive-IP zone, the physical property must be set and the address and
676 default router properties cannot be set.
677 .RE

679 .sp
680 .ne 2
681 .na
682 \fB\fBdevice\fR: match\fR
683 .ad
684 .sp .6
685 .RS 4n
686 Device name to match.
687 .RE

689 .sp
690 .ne 2
691 .na
692 \fB\fBrcctl\fR: name, value\fR
693 .ad
694 .sp .6
695 .RS 4n
696 The name and \fBpriv\fR/\fBlimit\fR/\fBaction\fR triple of a resource control.
697 See \fBprctl\fR(1) and \fBrcctladm\fR(1M). The preferred way to set rctl values
698 is to use the global property name associated with a specific rctl.
699 .RE

701 .sp
702 .ne 2
703 .na
704 \fB\fBattr\fR: name, type, value\fR
705 .ad
706 .sp .6
707 .RS 4n
708 The name, type and value of a generic attribute. The \fBtype\fR must be one of
709 \fBint\fR, \fBuint\fR, \fBboolean\fR or \fBstring\fR, and the value must be of
710 that type. \fBuint\fR means unsigned, that is, a non-negative integer.
711 .RE

713 .sp

```

```

714 .ne 2
715 .na
716 \fB\fBdataset\fR: name\fR
717 .ad
718 .sp .6
719 .RS 4n
720 The name of a \fBZFS\fR dataset to be accessed from within the zone. See
721 \fBzfs\fR(1M).
722 .RE

724 .sp
725 .ne 2
726 .na
727 \fB\fBglobal: \fBcpu-shares\fR\fR
728 .ad
729 .sp .6
730 .RS 4n
731 The number of Fair Share Scheduler (FSS) shares to allocate to this zone. This
732 property is incompatible with the \fBdedicated-cpu\fR resource. This property
733 is the preferred way to set the \fBzone.cpu-shares\fR rctl.
734 .RE

736 .sp
737 .ne 2
738 .na
739 \fB\fBglobal: \fBmax-lwps\fR\fR
740 .ad
741 .sp .6
742 .RS 4n
743 The maximum number of LWPs simultaneously available to this zone. This property
744 is the preferred way to set the \fBzone.max-lwps\fR rctl.
745 .RE

747 .sp
748 .ne 2
749 .na
750 \fB\fBglobal: \fBmax-msg-ids\fR\fR
751 .ad
752 .sp .6
753 .RS 4n
754 The maximum number of message queue IDs allowed for this zone. This property is
755 the preferred way to set the \fBzone.max-msg-ids\fR rctl.
756 .RE

758 .sp
759 .ne 2
760 .na
761 \fB\fBglobal: \fBmax-sem-ids\fR\fR
762 .ad
763 .sp .6
764 .RS 4n
765 The maximum number of semaphore IDs allowed for this zone. This property is the
766 preferred way to set the \fBzone.max-sem-ids\fR rctl.
767 .RE

769 .sp
770 .ne 2
771 .na
772 \fB\fBglobal: \fBmax-shm-ids\fR\fR
773 .ad
774 .sp .6
775 .RS 4n
776 The maximum number of shared memory IDs allowed for this zone. This property is
777 the preferred way to set the \fBzone.max-shm-ids\fR rctl.
778 .RE

```

```

780 .sp
781 .ne 2
782 .na
783 \fBglobal: \fBmax-shm-memory\fR\fR
784 .ad
785 .sp .6
786 .RS 4n
787 The maximum amount of shared memory allowed for this zone. This property is the
788 preferred way to set the \fBzone.max-shm-memory\fR rctl. A scale (K, M, G, T)
789 can be applied to the value for this number (for example, 1M is one megabyte).
790 .RE

792 .sp
793 .ne 2
794 .na
795 \fBglobal: \fBscheduling-class\fR\fR
796 .ad
797 .sp .6
798 .RS 4n
799 Specifies the scheduling class used for processes running in a zone. When this
800 property is not specified, the scheduling class is established as follows:
801 .RS +4
802 .TP
803 .ie t \(\bu
804 .el o
805 If the \fBcpu-shares\fR property or equivalent rctl is set, the scheduling
806 class FSS is used.
807 .RE
808 .RS +4
809 .TP
810 .ie t \(\bu
811 .el o
812 If neither \fBcpu-shares\fR nor the equivalent rctl is set and the zone's pool
813 property references a pool that has a default scheduling class, that class is
814 used.
815 .RE
816 .RS +4
817 .TP
818 .ie t \(\bu
819 .el o
820 Under any other conditions, the system default scheduling class is used.
821 .RE
822 .RE

826 .sp
827 .ne 2
828 .na
829 \fB\fBdedicated-cpu\fR: ncpus, importance\fR
830 .ad
831 .sp .6
832 .RS 4n
833 The number of CPUs that should be assigned for this zone's exclusive use. The
834 zone will create a pool and processor set when it boots. See \fBpooladm\fR(1M)
835 and \fBpoolcfg\fR(1M) for more information on resource pools. The \fBncpu\fR
836 property can specify a single value or a range (for example, 1-4) of
837 processors. The \fBimportance\fR property is optional; if set, it will specify
838 the \fBpset.importance\fR value for use by \fBpoold\fR(1M). If this resource is
839 used, there must be enough free processors to allocate to this zone when it
840 boots or the zone will not boot. The processors assigned to this zone will not
841 be available for the use of the global zone or other zones. This resource is
842 incompatible with both the \fBpool\fR and \fBcpu-shares\fR properties. Only a
843 single instance of this resource can be added to the zone.
844 .RE

```

```

846 .sp
847 .ne 2
848 .na
849 \fB\fBcapped-memory\fR: physical, swap, locked\fR
850 .ad
851 .sp .6
852 .RS 4n
853 The caps on the memory that can be used by this zone. A scale (K, M, G, T) can
854 be applied to the value for each of these numbers (for example, 1M is one
855 megabyte). Each of these properties is optional but at least one property must
856 be set when adding this resource. Only a single instance of this resource can
857 be added to the zone. The \fBphysical\fR property sets the \fBmax-rss\fR for
858 this zone. This will be enforced by \fBrcapd\fR(1M) running in the global zone.
859 The \fBswap\fR property is the preferred way to set the \fBzone.max-swap\fR
860 rctl. The \fBblocked\fR property is the preferred way to set the
861 \fBzone.max-locked-memory\fR rctl.
862 .RE

864 .sp
865 .ne 2
866 .na
867 \fB\fBcapped-cpu\fR: ncpus\fR
868 .ad
869 .sp .6
870 .RS 4n
871 Sets a limit on the amount of CPU time that can be used by a zone. The unit
872 used translates to the percentage of a single CPU that can be used by all user
873 threads in a zone, expressed as a fraction (for example, \fB&.75\fR) or a
874 mixed number (whole number and fraction, for example, \fB1.25\fR). An
875 \fBncpu\fR value of \fB1\fR means 100% of a CPU, a value of \fB1.25\fR means
876 125%, \fB&.75\fR mean 75%, and so forth. When projects within a capped zone
877 have their own caps, the minimum value takes precedence.
878 .sp
879 The \fBcapped-cpu\fR property is an alias for \fBzone.cpu-cap\fR resource
880 control and is related to the \fBzone.cpu-cap\fR resource control. See
881 \fBresource_controls\fR(5).
882 .RE

884 .sp
885 .ne 2
886 .na
887 \fB\fBsecurity-flags\fR: lower, default, upper\fR
888 .ad
889 .sp .6
890 .RS 4n
891 Set the process security flags associated with the zone. The \fBblower\fR and
892 \fBbupper\fR fields set the limits, the \fBdefault\fR field is set of flags all
893 zone processes inherit.
894 .RE

896 .sp
897 .ne 2
898 .na
899 #endif /* ! codereview */
900 \fBglobal: \fBfs-allowed\fR\fR
901 .ad
902 .sp .6
903 .RS 4n
904 A comma-separated list of additional filesystems that may be mounted within
905 the zone; for example "ufs,pcfs". By default, only hfs(7fs) and network
906 filesystems can be mounted. If the first entry in the list is "-" then
907 that disables all of the default filesystems. If any filesystems are listed
908 after "-" then only those filesystems can be mounted.

910 This property does not apply to filesystems mounted into the zone via "add fs"
911 or "add dataset".

```

913 WARNING: allowing filesystem mounts other than the default may allow the zone
 914 administrator to compromise the system with a malicious filesystem image, and
 915 is not supported.
 916 .RE

918 .sp
 919 .LP
 920 The following table summarizes resources, property-names, and types:
 921 .sp
 922 .in +2
 923 .nf

resource	property-name	type
(global)	zonename	simple
(global)	zonepath	simple
(global)	autoboot	simple
(global)	bootargs	simple
(global)	pool	simple
(global)	limitpriv	simple
(global)	brand	simple
(global)	ip-type	simple
(global)	hostid	simple
(global)	cpu-shares	simple
(global)	max-lwps	simple
(global)	max-msg-ids	simple
(global)	max-sem-ids	simple
(global)	max-shm-ids	simple
(global)	max-shm-memory	simple
(global)	scheduling-class	simple
fs	dir	simple
	special	simple
	raw	simple
	type	simple
	options	list of simple
net	address	simple
	physical	simple
device	match	simple
rctl	name	simple
	value	list of complex
attr	name	simple
	type	simple
	value	simple
dataset	name	simple
dedicated-cpu	ncpus	simple or range
	importance	simple
capped-memory	physical	simple with scale
	swap	simple with scale
	locked	simple with scale

962 capped-cpu ncpus simple
 963 security-flags lower simple
 964 default simple
 965 upper simple

966 #endif /* ! codereview */
 967 .fi
 968 .in -2
 969 .sp

971 .sp
 972 .LP
 973 To further specify things, the breakdown of the complex property "value" of the
 974 "rctl" resource type, it consists of three name/value pairs, the names being
 975 "priv", "limit" and "action", each of which takes a simple value. The "name"
 976 property of an "attr" resource is syntactically restricted in a fashion similar
 977 but not identical to zone names: it must begin with an alphanumeric, and can

978 contain alphanumerics plus the hyphen (\fB-\fR), underscore (\fB_\fR), and dot
 979 (\fB&.\fR) characters. Attribute names beginning with "zone" are reserved for
 980 use by the system. Finally, the "autoboot" global property must have a value of
 981 "true" or "false".

982 .SS "Using Kernel Statistics to Monitor CPU Caps"

427 .sp
 983 .LP

984 Using the kernel statistics (\fBkstat\fR(3KSTAT)) module \fBcaps\fR, the system
 985 maintains information for all capped projects and zones. You can access this
 986 information by reading kernel statistics (\fBkstat\fR(3KSTAT)), specifying
 987 \fBcaps\fR as the \fBkstat\fR module name. The following command displays
 988 kernel statistics for all active CPU caps:

989 .sp
 990 .in +2
 991 .nf
 992 # \fBkstat caps::'/cpucaps/'\fR
 993 .fi
 994 .in -2
 995 .sp

997 .sp
 998 .LP
 999 A \fBkstat\fR(1M) command running in a zone displays only CPU caps relevant for
 1000 that zone and for projects in that zone. See \fBEXAMPLES\fR.

1001 .sp
 1002 .LP
 1003 The following are cap-related arguments for use with \fBkstat\fR(1M):
 1004 .sp

1005 .ne 2
 1006 .na
 1007 \fBcaps\fR
 1008 .ad
 1009 .sp .6
 1010 .RS 4n
 1011 The \fBkstat\fR module.
 1012 .RE

1014 .sp
 1015 .ne 2
 1016 .na
 1017 \fBproject_caps\fR or \fBzone_caps\fR
 1018 .ad
 1019 .sp .6
 1020 .RS 4n
 1021 \fBclass\fR, for use with the \fBkstat\fR \fB-c\fR option.
 1022 .RE

1024 .sp
 1025 .ne 2
 1026 .na
 1027 \fBproject_id\fR or \fBzone_id\fR
 1028 .ad
 1029 .sp .6
 1030 .RS 4n
 1031 \fBname\fR, for use with the \fBkstat\fR \fB-n\fR option. \fBid\fR is the
 1032 project or zone identifier.
 1033 .RE

1035 .sp
 1036 .LP
 1037 The following fields are displayed in response to a \fBkstat\fR(1M) command
 1038 requesting statistics for all CPU caps.

1039 .sp
 1040 .ne 2
 1041 .na
 1042 \fBmodule\fR

```

1043 .ad
1044 .sp .6
1045 .RS 4n
1046 In this usage of \fBkstat\fR, this field will have the value \fBcaps\fR.
1047 .RE

1049 .sp
1050 .ne 2
1051 .na
1052 \fB\fBname\fR\fR
1053 .ad
1054 .sp .6
1055 .RS 4n
1056 As described above, \fBcpucaps_project_\fR\fIid\fR or
1057 \fBcpucaps_zone_\fR\fIid\fR
1058 .RE

1060 .sp
1061 .ne 2
1062 .na
1063 \fB\fBabove_sec\fR\fR
1064 .ad
1065 .sp .6
1066 .RS 4n
1067 Total time, in seconds, spent above the cap.
1068 .RE

1070 .sp
1071 .ne 2
1072 .na
1073 \fB\fBbelow_sec\fR\fR
1074 .ad
1075 .sp .6
1076 .RS 4n
1077 Total time, in seconds, spent below the cap.
1078 .RE

1080 .sp
1081 .ne 2
1082 .na
1083 \fB\fBmaxusage\fR\fR
1084 .ad
1085 .sp .6
1086 .RS 4n
1087 Maximum observed CPU usage.
1088 .RE

1090 .sp
1091 .ne 2
1092 .na
1093 \fB\fBnwait\fR\fR
1094 .ad
1095 .sp .6
1096 .RS 4n
1097 Number of threads on cap wait queue.
1098 .RE

1100 .sp
1101 .ne 2
1102 .na
1103 \fB\fBusage\fR\fR
1104 .ad
1105 .sp .6
1106 .RS 4n
1107 Current aggregated CPU usage for all threads belonging to a capped project or
1108 zone, in terms of a percentage of a single CPU.

```

```

1109 .RE

1111 .sp
1112 .ne 2
1113 .na
1114 \fB\fBvalue\fR\fR
1115 .ad
1116 .sp .6
1117 .RS 4n
1118 The cap value, in terms of a percentage of a single CPU.
1119 .RE

1121 .sp
1122 .ne 2
1123 .na
1124 \fB\fBzonename\fR\fR
1125 .ad
1126 .sp .6
1127 .RS 4n
1128 Name of the zone for which statistics are displayed.
1129 .RE

1131 .sp
1132 .LP
1133 See \fBEXAMPLES\fR for sample output from a \fBkstat\fR command.
1134 .SH OPTIONS
1135 .LP
1136 The following options are supported:
1137 .sp
1138 .ne 2
1139 .na
1140 \fB\fB-f\fR \fIcommand_file\fR
1141 .ad
1142 .sp .6
1143 .RS 4n
1144 Specify the name of \fBzonecfg\fR command file. \fIcommand_file\fR is a text
1145 file of \fBzonecfg\fR subcommands, one per line.
1146 .RE

1148 .sp
1149 .ne 2
1150 .na
1151 \fB\fB-z\fR \fIzonename\fR
1152 .ad
1153 .sp .6
1154 .RS 4n
1155 Specify the name of a zone. Zone names are case sensitive. Zone names must
1156 begin with an alphanumeric character and can contain alphanumeric characters,
1157 the underscore (\fB_\fR) the hyphen (\fB-\fR), and the dot (\fB&.\fR). The
1158 name \fBglobal\fR and all names beginning with \fBSUNW\fR are reserved and
1159 cannot be used.
1160 .RE

1162 .SH SUBCOMMANDS
1163 .LP
1164 You can use the \fBadd\fR and \fBselect\fR subcommands to select a specific
1165 resource, at which point the scope changes to that resource. The \fBend\fR and
1166 \fBcancel\fR subcommands are used to complete the resource specification, at
1167 which time the scope is reverted back to global. Certain subcommands, such as
1168 \fBadd\fR, \fBremove\fR and \fBset\fR, have different semantics in each scope.
1169 .sp
1170 .LP
1171 \fBzonecfg\fR supports a semicolon-separated list of subcommands. For example:
1172 .sp

```

```

1173 .in +2
1174 .nf
1175 # \fBzonecfg -z myzone "add net; set physical=myvnic; end"\fR
1176 .fi
1177 .in -2
1178 .sp

1180 .sp
1181 .LP
1182 Subcommands which can result in destructive actions or loss of work have an
1183 \fB-F\fR option to force the action. If input is from a terminal device, the
1184 user is prompted when appropriate if such a command is given without the
1185 \fB-F\fR option otherwise, if such a command is given without the \fB-F\fR
1186 option, the action is disallowed, with a diagnostic message written to standard
1187 error.
1188 .sp
1189 .LP
1190 The following subcommands are supported:
1191 .sp
1192 .ne 2
1193 .na
1194 \fB\fBadd\fR \fIresource-type\fR (global scope)\fR
1195 .ad
1196 .br
1197 .na
1198 \fB\fBadd\fR \fIproperty-name property-value\fR (resource scope)\fR
1199 .ad
1200 .sp .6
1201 .RS 4n
1202 In the global scope, begin the specification for a given resource type. The
1203 scope is changed to that resource type.
1204 .sp
1205 In the resource scope, add a property of the given name with the given value.
1206 The syntax for property values varies with different property types. In
1207 general, it is a simple value or a list of simple values enclosed in square
1208 brackets, separated by commas (\fB[foo,bar,baz]\fR). See \fBPROPERTIES\fR.
1209 .RE

1211 .sp
1212 .ne 2
1213 .na
1214 \fB\fBcancel\fR\fR
1215 .ad
1216 .sp .6
1217 .RS 4n
1218 End the resource specification and reset scope to global. Abandons any
1219 partially specified resources. \fBcancel\fR is only applicable in the resource
1220 scope.
1221 .RE

1223 .sp
1224 .ne 2
1225 .na
1226 \fB\fBclear\fR \fIproperty-name\fR\fR
1227 .ad
1228 .sp .6
1229 .RS 4n
1230 Clear the value for the property.
1231 .RE

1233 .sp
1234 .ne 2
1235 .na
1236 \fB\fBcommit\fR\fR
1237 .ad
1238 .sp .6

```

```

1239 .RS 4n
1240 Commit the current configuration from memory to stable storage. The
1241 configuration must be committed to be used by \fBzoneadm\fR. Until the
1242 in-memory configuration is committed, you can remove changes with the
1243 \fBbrevert\fR subcommand. The \fBcommit\fR operation is attempted automatically
1244 upon completion of a \fBzonecfg\fR session. Since a configuration may be
1245 correct to be committed, this operation automatically does a verify.
1246 .RE

1248 .sp
1249 .ne 2
1250 .na
1251 \fB\fBcreate [\fR\fB-F\fR\fB] [\fR \fB-a\fR \fIpath\fR |\fB-b\fR \fB|\fR
1252 \fB-t\fR \fItemplate\fR\fB]\fR
1253 .ad
1254 .sp .6
1255 .RS 4n
1256 Create an in-memory configuration for the specified zone. Use \fBcreate\fR to
1257 begin to configure a new zone. See \fBcommit\fR for saving this to stable
1258 storage.
1259 .sp
1260 If you are overwriting an existing configuration, specify the \fB-F\fR option
1261 to force the action. Specify the \fB-t\fR \fItemplate\fR option to create a
1262 configuration identical to \fItemplate\fR, where \fItemplate\fR is the name of
1263 a configured zone.
1264 .sp
1265 Use the \fB-a\fR \fIpath\fR option to facilitate configuring a detached zone on
1266 a new host. The \fIpath\fR parameter is the zonepath location of a detached
1267 zone that has been moved on to this new host. Once the detached zone is
1268 configured, it should be installed using the "\fBzoneadm attach\fR" command
1269 (see \fBzoneadm\fR(1M)). All validation of the new zone happens during the
1270 \fBattach\fR process, not during zone configuration.
1271 .sp
1272 Use the \fB-b\fR option to create a blank configuration. Without arguments,
1273 \fBcreate\fR applies the Sun default settings.
1274 .RE

1276 .sp
1277 .ne 2
1278 .na
1279 \fB\fBdelete [\fR\fB-F\fR\fB]\fR
1280 .ad
1281 .sp .6
1282 .RS 4n
1283 Delete the specified configuration from memory and stable storage. This action
1284 is instantaneous, no commit is necessary. A deleted configuration cannot be
1285 reverted.
1286 .sp
1287 Specify the \fB-F\fR option to force the action.
1288 .RE

1290 .sp
1291 .ne 2
1292 .na
1293 \fB\fBend\fR
1294 .ad
1295 .sp .6
1296 .RS 4n
1297 End the resource specification. This subcommand is only applicable in the
1298 resource scope. \fBzonecfg\fR checks to make sure the current resource is
1299 completely specified. If so, it is added to the in-memory configuration (see
1300 \fBcommit\fR for saving this to stable storage) and the scope reverts to
1301 global. If the specification is incomplete, it issues an appropriate error
1302 message.
1303 .RE

```

```

1305 .sp
1306 .ne 2
1307 .na
1308 \fB\fBexport [\fR\fB-f\fR \fIoutput-file\fR\fB]\fR\fR
1309 .ad
1310 .sp .6
1311 .RS 4n
1312 Print configuration to standard output. Use the \fB-f\fR option to print the
1313 configuration to \fIoutput-file\fR. This option produces output in a form
1314 suitable for use in a command file.
1315 .RE

1317 .sp
1318 .ne 2
1319 .na
1320 \fB\fBhelp [usage] [\fIsubcommand\fR] [syntax] [\fR\fIcommand-name\fR\fB]\fR\fR
1321 .ad
1322 .sp .6
1323 .RS 4n
1324 Print general help or help about given topic.
1325 .RE

1327 .sp
1328 .ne 2
1329 .na
1330 \fB\fBinfo zonename | zonepath | autoboot | brand | pool | limitpriv\fR\fR
1331 .ad
1332 .br
1333 .na
1334 \fB\fBinfo [\fR\fIresource-type\fR
1335 \fB[\fR\fIproperty-name\fR\fB=\fR\fIproperty-value\fR\fB]*\fR]\fR
1336 .ad
1337 .sp .6
1338 .RS 4n
1339 Display information about the current configuration. If \fIresource-type\fR is
1340 specified, displays only information about resources of the relevant type. If
1341 any \fIproperty-name\fR value pairs are specified, displays only information
1342 about resources meeting the given criteria. In the resource scope, any
1343 arguments are ignored, and \fBinfo\fR displays information about the resource
1344 which is currently being added or modified.
1345 .RE

1347 .sp
1348 .ne 2
1349 .na
1350 \fB\fBremove \fR \fIresource-type\fR\fB{\fR\fIproperty-name\fR\fB=\fR\fIproperty
1351 -value\fR\fB}\fR(global scope)\fR
1352 .ad
1353 .sp .6
1354 .RS 4n
1355 In the global scope, removes the specified resource. The \fB[]\fR syntax means
1356 0 or more of whatever is inside the square braces. If you want only to remove a
1357 single instance of the resource, you must specify enough property name-value
1358 pairs for the resource to be uniquely identified. If no property name-value
1359 pairs are specified, all instances will be removed. If there is more than one
1360 pair is specified, a confirmation is required, unless you use the \fB-F\fR
1361 option.
1362 .RE

1364 .sp
1365 .ne 2
1366 .na
1367 \fB\fBselect \fR \fIresource-type\fR
1368 \fB[\fR\fIproperty-name\fR\fB=\fR\fIproperty-value\fR\fB]\fR
1369 .ad
1370 .sp .6

```

```

1371 .RS 4n
1372 Select the resource of the given type which matches the given
1373 \fIproperty-name\fR \fIproperty-value\fR pair criteria, for modification. This
1374 subcommand is applicable only in the global scope. The scope is changed to that
1375 resource type. The \fB{}\fR syntax means 1 or more of whatever is inside the
1376 curly braces. You must specify enough \fIproperty -name property-value\fR pairs
1377 for the resource to be uniquely identified.
1378 .RE

1380 .sp
1381 .ne 2
1382 .na
1383 \fB\fBset \fR \fIproperty-name\fR\fB=\fR\fIproperty\fR\fB-\fR\fIvalue\fR\fR
1384 .ad
1385 .sp .6
1386 .RS 4n
1387 Set a given property name to the given value. Some properties (for example,
1388 \fBzonename\fR and \fBzonepath\fR) are global while others are
1389 resource-specific. This subcommand is applicable in both the global and
1390 resource scopes.
1391 .RE

1393 .sp
1394 .ne 2
1395 .na
1396 \fB\fBverify\fR
1397 .ad
1398 .sp .6
1399 .RS 4n
1400 Verify the current configuration for correctness:
1401 .RS +4
1402 .TP
1403 .ie t \(\bu
1404 .el o
1405 All resources have all of their required properties specified.
1406 .RE
1407 .RS +4
1408 .TP
1409 .ie t \(\bu
1410 .el o
1411 A \fBzonepath\fR is specified.
1412 .RE
1413 .RE

1415 .sp
1416 .ne 2
1417 .na
1418 \fB\fBrevert \fR \fB[\fR\fB-F\fR\fB]\fR
1419 .ad
1420 .sp .6
1421 .RS 4n
1422 Revert the configuration back to the last committed state. The \fB-F\fR option
1423 can be used to force the action.
1424 .RE

1426 .sp
1427 .ne 2
1428 .na
1429 \fB\fBexit [\fR\fB-F\fR\fB]\fR
1430 .ad
1431 .sp .6
1432 .RS 4n
1433 Exit the \fBzonecfg\fR session. A commit is automatically attempted if needed.
1434 You can also use an \fBEOF\fR character to exit \fBzonecfg\fR. The \fB-F\fR
1435 option can be used to force the action.
1436 .RE

```

```

1438 .SH EXAMPLES
1439 .LP
1440 \fBExample 1 \fRCreating the Environment for a New Zone
1441 .sp
1442 .LP
1443 In the following example, \fBzoncfg\fR creates the environment for a new zone.
1444 \fB/usr/local\fR is loopback mounted from the global zone into
1445 \fB/opt/local\fR. \fB/opt/sfw\fR is loopback mounted from the global zone,
1446 three logical network interfaces are added, and a limit on the number of
1447 fair-share scheduler (FSS) CPU shares for a zone is set using the \fBBrctl\fR
1448 resource type. The example also shows how to select a given resource for
1449 modification.

```

```

1451 .sp
1452 .in +2
1453 .nf
1454 example# \fBzoncfg -z myzone3\fR
1455 my-zone3: No such zone configured
1456 Use 'create' to begin configuring a new zone.
1457 zoncfg:myzone3> \fBcreate\fR
1458 zoncfg:myzone3> \fBset zonepath=/export/home/my-zone3\fR
1459 zoncfg:myzone3> \fBset autoboot=true\fR
1460 zoncfg:myzone3> \fBadd fs\fR
1461 zoncfg:myzone3:fs> \fBset dir=/usr/local\fR
1462 zoncfg:myzone3:fs> \fBset special=/opt/local\fR
1463 zoncfg:myzone3:fs> \fBset type=lofs\fR
1464 zoncfg:myzone3:fs> \fBadd options [ro,nodevices]\fR
1465 zoncfg:myzone3:fs> \fBend\fR
1466 zoncfg:myzone3> \fBadd fs\fR
1467 zoncfg:myzone3:fs> \fBset dir=/mnt\fR
1468 zoncfg:myzone3:fs> \fBset special=/dev/dsk/c0t0d0s7\fR
1469 zoncfg:myzone3:fs> \fBset raw=/dev/rdisk/c0t0d0s7\fR
1470 zoncfg:myzone3:fs> \fBset type=ufs\fR
1471 zoncfg:myzone3:fs> \fBend\fR
1472 zoncfg:myzone3> \fBadd net\fR
1473 zoncfg:myzone3:net> \fBset address=192.168.0.1/24\fR
1474 zoncfg:myzone3:net> \fBset physical=eri0\fR
1475 zoncfg:myzone3:net> \fBend\fR
1476 zoncfg:myzone3> \fBadd net\fR
1477 zoncfg:myzone3:net> \fBset address=192.168.1.2/24\fR
1478 zoncfg:myzone3:net> \fBset physical=eri0\fR
1479 zoncfg:myzone3:net> \fBend\fR
1480 zoncfg:myzone3> \fBadd net\fR
1481 zoncfg:myzone3:net> \fBset address=192.168.2.3/24\fR
1482 zoncfg:myzone3:net> \fBset physical=eri0\fR
1483 zoncfg:myzone3:net> \fBend\fR
1484 zoncfg:my-zone3> \fBset cpu-shares=5\fR
1485 zoncfg:my-zone3> \fBadd capped-memory\fR
1486 zoncfg:my-zone3:capped-memory> \fBset physical=50m\fR
1487 zoncfg:my-zone3:capped-memory> \fBset swap=100m\fR
1488 zoncfg:my-zone3:capped-memory> \fBend\fR
1489 zoncfg:myzone3> \fBexit\fR
1490 .fi
1491 .in -2
1492 .sp

```

```

1494 .LP
1495 \fBExample 2 \fRCreating a Non-Native Zone
1496 .sp
1497 .LP
1498 The following example creates a new Linux zone:

```

```

1500 .sp
1501 .in +2
1502 .nf

```

```

1503 example# \fBzoncfg -z lxzone\fR
1504 lxzone: No such zone configured
1505 Use 'create' to begin configuring a new zone
1506 zoncfg:lxzone> \fBcreate -t SUNWlx\fR
1507 zoncfg:lxzone> \fBset zonepath=/export/zones/lxzone\fR
1508 zoncfg:lxzone> \fBset autoboot=true\fR
1509 zoncfg:lxzone> \fBexit\fR
1510 .fi
1511 .in -2
1512 .sp

```

```

1514 .LP
1515 \fBExample 3 \fRCreating an Exclusive-IP Zone
1516 .sp
1517 .LP
1518 The following example creates a zone that is granted exclusive access to
1519 \fBbge1\fR and \fBbge33000\fR and that is isolated at the IP layer from the
1520 other zones configured on the system.

```

```

1522 .sp
1523 .LP
1524 The IP addresses and routing is configured inside the new zone using
1525 \fBsysidtool\fR(1M).

```

```

1527 .sp
1528 .in +2
1529 .nf
1530 example# \fBzoncfg -z excl\fR
1531 excl: No such zone configured
1532 Use 'create' to begin configuring a new zone
1533 zoncfg:excl> \fBcreate\fR
1534 zoncfg:excl> \fBset zonepath=/export/zones/excl\fR
1535 zoncfg:excl> \fBset ip-type=exclusive\fR
1536 zoncfg:excl> \fBadd net\fR
1537 zoncfg:excl:net> \fBset physical=bge1\fR
1538 zoncfg:excl:net> \fBend\fR
1539 zoncfg:excl> \fBadd net\fR
1540 zoncfg:excl:net> \fBset physical=bge33000\fR
1541 zoncfg:excl:net> \fBend\fR
1542 zoncfg:excl> \fBexit\fR
1543 .fi
1544 .in -2
1545 .sp

```

```

1547 .LP
1548 \fBExample 4 \fRAssociating a Zone with a Resource Pool
1549 .sp
1550 .LP
1551 The following example shows how to associate an existing zone with an existing
1552 resource pool:

```

```

1554 .sp
1555 .in +2
1556 .nf
1557 example# \fBzoncfg -z myzone\fR
1558 zoncfg:myzone> \fBset pool=mypool\fR
1559 zoncfg:myzone> \fBexit\fR
1560 .fi
1561 .in -2
1562 .sp

```

```

1564 .sp
1565 .LP
1566 For more information about resource pools, see \fBpooladm\fR(1M) and
1567 \fBpoolcfg\fR(1M).

```



```

1569 .LP
1570 \fBExample 5 \fRChanging the Name of a Zone
1571 .sp
1572 .LP
1573 The following example shows how to change the name of an existing zone:

1575 .sp
1576 .in +2
1577 .nf
1578 example# \fBzonecfg -z myzone\fR
1579 zonecfg:myzone> \fBset zonename=myzone2\fR
1580 zonecfg:myzone2> \fBexit\fR
1581 .fi
1582 .in -2
1583 .sp

1585 .LP
1586 \fBExample 6 \fRChanging the Privilege Set of a Zone
1587 .sp
1588 .LP
1589 The following example shows how to change the set of privileges an existing
1590 zone's processes will be limited to the next time the zone is booted. In this
1591 particular case, the privilege set will be the standard safe set of privileges
1592 a zone normally has along with the privilege to change the system date and
1593 time:

1595 .sp
1596 .in +2
1597 .nf
1598 example# \fBzonecfg -z myzone\fR
1599 zonecfg:myzone> \fBset limitpriv="default,sys_time"\fR
1600 zonecfg:myzone2> \fBexit\fR
1601 .fi
1602 .in -2
1603 .sp

1605 .LP
1606 \fBExample 7 \fRSetting the \fBzone.cpu-shares\fR Property for the Global Zone
1607 .sp
1608 .LP
1609 The following command sets the \fBzone.cpu-shares\fR property for the global
1610 zone:

1612 .sp
1613 .in +2
1614 .nf
1615 example# \fBzonecfg -z global\fR
1616 zonecfg:global> \fBset cpu-shares=5\fR
1617 zonecfg:global> \fBexit\fR
1618 .fi
1619 .in -2
1620 .sp

1622 .LP
1623 \fBExample 8 \fRUsing Pattern Matching
1624 .sp
1625 .LP
1626 The following commands illustrate \fBzonecfg\fR support for pattern matching.
1627 In the zone \fBflexlm\fR, enter:

1629 .sp
1630 .in +2
1631 .nf
1632 zonecfg:flexlm> \fBadd device\fR
1633 zonecfg:flexlm:device> \fBset match="/dev/cua/a00[2-5]"\fR
1634 zonecfg:flexlm:device> \fBend\fR

```

```

1635 .fi
1636 .in -2
1637 .sp

1639 .sp
1640 .LP
1641 In the global zone, enter:

1643 .sp
1644 .in +2
1645 .nf
1646 global# \fBls /dev/cua\fR
1647 a      a000 a001 a002 a003 a004 a005 a006 a007 b
1648 .fi
1649 .in -2
1650 .sp

1652 .sp
1653 .LP
1654 In the zone \fBflexlm\fR, enter:

1656 .sp
1657 .in +2
1658 .nf
1659 flexlm# \fBls /dev/cua\fR
1660 a002 a003 a004 a005
1661 .fi
1662 .in -2
1663 .sp

1665 .LP
1666 \fBExample 9 \fRSetting a Cap for a Zone to Three CPUs
1667 .sp
1668 .LP
1669 The following sequence uses the \fBzonecfg\fR command to set the CPU cap for a
1670 zone to three CPUs.

1672 .sp
1673 .in +2
1674 .nf
1675 zonecfg:myzone> \fBadd capped-cpu\fR
1676 zonecfg:myzone>capped-cpu> \fBset ncpus=3\fR
1677 zonecfg:myzone>capped-cpu>capped-cpu> \fBend\fR
1678 .fi
1679 .in -2
1680 .sp

1682 .sp
1683 .LP
1684 The preceding sequence, which uses the capped-cpu property, is equivalent to
1685 the following sequence, which makes use of the \fBzone.cpu-cap\fR resource
1686 control.

1688 .sp
1689 .in +2
1690 .nf
1691 zonecfg:myzone> \fBadd rctl\fR
1692 zonecfg:myzone:rctl> \fBset name=zone.cpu-cap\fR
1693 zonecfg:myzone:rctl> \fBadd value (priv=privileged,limit=300,action=none)\fR
1694 zonecfg:myzone:rctl> \fBend\fR
1695 .fi
1696 .in -2
1697 .sp

1699 .LP
1700 \fBExample 10 \fRUsing \fBkstat\fR to Monitor CPU Caps

```

```

1701 .sp
1702 .LP
1703 The following command displays information about all CPU caps.

1705 .sp
1706 .in +2
1707 .nf
1708 # \fBkstat -n /cpucaps/\fR
1709 module: caps                instance: 0
1710 name:   cpucaps_project_0   class:   project_caps
1711        above_sec            0
1712        below_sec            2157
1713        crtime                821.048183159
1714        maxusage              2
1715        nwait                  0
1716        snaptime              235885.637253027
1717        usage                  0
1718        value                  18446743151372347932
1719        zonename              global

1721 module: caps                instance: 0
1722 name:   cpucaps_project_1   class:   project_caps
1723        above_sec            0
1724        below_sec            0
1725        crtime                225339.192787265
1726        maxusage              5
1727        nwait                  0
1728        snaptime              235885.637591677
1729        usage                  5
1730        value                  18446743151372347932
1731        zonename              global

1733 module: caps                instance: 0
1734 name:   cpucaps_project_201 class:   project_caps
1735        above_sec            0
1736        below_sec            235105
1737        crtime                780.37961782
1738        maxusage              100
1739        nwait                  0
1740        snaptime              235885.637789687
1741        usage                  43
1742        value                  100
1743        zonename              global

1745 module: caps                instance: 0
1746 name:   cpucaps_project_202 class:   project_caps
1747        above_sec            0
1748        below_sec            235094
1749        crtime                791.72983782
1750        maxusage              100
1751        nwait                  0
1752        snaptime              235885.637967512
1753        usage                  48
1754        value                  100
1755        zonename              global

1757 module: caps                instance: 0
1758 name:   cpucaps_project_203 class:   project_caps
1759        above_sec            0
1760        below_sec            235034
1761        crtime                852.104401481
1762        maxusage              75
1763        nwait                  0
1764        snaptime              235885.638144304
1765        usage                  47
1766        value                  100

```

```

1767        zonename              global

1769 module: caps                instance: 0
1770 name:   cpucaps_project_86710 class:   project_caps
1771        above_sec            22
1772        below_sec            235166
1773        crtime                698.441717859
1774        maxusage              101
1775        nwait                  0
1776        snaptime              235885.638319871
1777        usage                  54
1778        value                  100
1779        zonename              global

1781 module: caps                instance: 0
1782 name:   cpucaps_zone_0      class:   zone_caps
1783        above_sec            100733
1784        below_sec            134332
1785        crtime                821.048177123
1786        maxusage              207
1787        nwait                  2
1788        snaptime              235885.638497731
1789        usage                  199
1790        value                  200
1791        zonename              global

1793 module: caps                instance: 1
1794 name:   cpucaps_project_0   class:   project_caps
1795        above_sec            0
1796        below_sec            0
1797        crtime                225360.256448422
1798        maxusage              7
1799        nwait                  0
1800        snaptime              235885.638714404
1801        usage                  7
1802        value                  18446743151372347932
1803        zonename              test_001

1805 module: caps                instance: 1
1806 name:   cpucaps_zone_1     class:   zone_caps
1807        above_sec            2
1808        below_sec            10524
1809        crtime                225360.256440278
1810        maxusage              106
1811        nwait                  0
1812        snaptime              235885.638896443
1813        usage                  7
1814        value                  100
1815        zonename              test_001
1816 .fi
1817 .in -2
1818 .sp

1820 .LP
1821 \fBExample 11 \fRDisplaying CPU Caps for a Specific Zone or Project
1822 .sp
1823 .LP
1824 Using the \fBkstat\fR \fB-c\fR and \fB-i\fR options, you can display CPU caps
1825 for a specific zone or project, as below. The first command produces a display
1826 for a specific project, the second for the same project within zone 1.

1828 .sp
1829 .in +2
1830 .nf
1831 # \fBkstat -c project_caps\fR

```

```

1833 # \fBkstat -c project_caps -i 1\fR
1834 .fi
1835 .in -2
1836 .sp

1838 .SH EXIT STATUS
1286 .sp
1839 .LP
1840 The following exit values are returned:
1841 .sp
1842 .ne 2
1843 .na
1844 \fB\fB0\fR\fR
1845 .ad
1846 .sp .6
1847 .RS 4n
1848 Successful completion.
1849 .RE

1851 .sp
1852 .ne 2
1853 .na
1854 \fB\fB1\fR\fR
1855 .ad
1856 .sp .6
1857 .RS 4n
1858 An error occurred.
1859 .RE

1861 .sp
1862 .ne 2
1863 .na
1864 \fB\fB2\fR\fR
1865 .ad
1866 .sp .6
1867 .RS 4n
1868 Invalid usage.
1869 .RE

1871 .SH ATTRIBUTES
1320 .sp
1872 .LP
1873 See \fBattributes\fR(5) for descriptions of the following attributes:
1874 .sp

1876 .sp
1877 .TS
1878 box;
1879 c | c
1880 l | l .
1881 ATTRIBUTE TYPE ATTRIBUTE VALUE
1882 _
1883 Interface Stability Volatile
1884 .TE

1886 .SH SEE ALSO
1336 .sp
1887 .LP
1888 \fBbpriv\fR(1), \fBprctl\fR(1), \fBzlogin\fR(1), \fBkstat\fR(1M),
1889 \fBmount\fR(1M), \fBpooladm\fR(1M), \fBpoolcfg\fR(1M), \fBpoold\fR(1M),
1890 \fBbrcapd\fR(1M), \fBbrctladm\fR(1M), \fBsvcadm\fR(1M), \fBsysidtool\fR(1M),
1891 \fBzfs\fR(1M), \fBzoneadm\fR(1M), \fBpriv_str_to_set\fR(3C),
1892 \fBkstat\fR(3KSTAT), \fBvfstab\fR(4), \fBattributes\fR(5), \fBbrands\fR(5),
1893 \fBfnmatch\fR(5), \fBlx\fR(5), \fBprivileges\fR(5), \fBresource_controls\fR(5),
1894 \fBsecurity-flags\fR(5), \fBzones\fR(5)
1344 \fBzones\fR(5)

```

```

1895 .sp
1896 .LP
1897 \fISystem Administration Guide: Solaris Containers-Resource Management, and
1898 Solaris Zones\fR
1899 .SH NOTES
1350 .sp
1900 .LP
1901 All character data used by \fBzonecfg\fR must be in US-ASCII encoding.

```

```

*****
34589 Thu Jun 30 22:02:43 2016
new/usr/src/man/man3lib/libproc.3lib
Code review comments from jeffpc
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
1 .\"
2 .\" This file and its contents are supplied under the terms of the
3 .\" Common Development and Distribution License ("CDDL"), version 1.0.
4 .\" You may only use this file in accordance with the terms of version
5 .\" 1.0 of the CDDL.
6 .\"
7 .\" A full copy of the text of the CDDL should have accompanied this
8 .\" source. A copy of the CDDL is also available via the Internet at
9 .\" http://www.illumos.org/license/CDDL.
10 .\"
11 .\"
12 .\" Copyright 2015 Joyent, Inc.
13 .\"
14 .Dd June 06, 2016
14 .Dd May 08, 2016
15 .Dt LIBPROC 3LIB
16 .Os
17 .Sh NAME
18 .Nm libproc
19 .Nd process control library
20 .Sh SYNOPSIS
21 .Lb libproc
22 .In libproc.h
23 .Sh DESCRIPTION
24 The
25 .Nm
26 library provides consumers a general series of interfaces to inspect
27 and control both live processes and core files. It is intended for
28 introspection tools such as debuggers by providing a high-level
29 interface to the /proc file system
30 .Pf ( Xr proc 4 ) .
31 .Pp
32 The
33 .Nm
34 library provides interfaces that focus on:
35 .Bl -bullet -offset indent
36 .It
37 Creating and attaching to live process, core files, and arbitrary ELF
38 objects.
39 .It
40 Interrogating the state of a process or core file.
41 .It
42 Manipulating the current state of a process or thread.
43 .It
44 Interrogating the state of threads of a process or core file.
45 .It
46 Running system calls in the context of another process.
47 .It
48 Various utilities for iterating process and core file file descriptors,
49 mappings, symbols, and more.
50 .It
51 Various utilities to support debugging tools.
52 .El
53 .Ss Live Processes
54 The
55 .Nm
56 library can be used to manipulate running processes and to create new

```

```

57 ones. To manipulate an existing process first
58 .Em grab
59 it with the
60 .Em Pgrab
61 function. A process is generally stopped as a side effect of grabbing
62 it. Callers must exercise caution, as if they do not use the library
63 correctly, or they terminate unexpectedly, a process may remain
64 stopped.
65 .Pp
66 Unprivileged users may only grab their own processes. Users with the
67 privilege
68 .Sy PRIV_PROC_OWNER
69 may manipulate processes that they do not own; however, additional
70 restrictions as described in
71 .Xr privileges 5
72 apply.
73 .Pp
74 In addition, the
75 .Fn Pcreate
76 and
77 .Fn Pxcreate
78 functions may be used to create processes which are always controlled by
79 the library.
80 .Ss Core Files
81 The
82 .Nm
83 library has the ability to open and interpret core files produced by
84 processes on the system. Process core dump generation is controlled by
85 the
86 .Xr coreadm 1M
87 command. In addition, the library has the ability to understand and
88 interpret core dumps generated by Linux kernel and can provide a subset
89 of its functionality on such core files, provided the original binary is
90 also present.
91 .Pp
92 Not all functions in the
93 .Nm
94 library are valid for core files. In general, none of the commands
95 which manipulate the current state of a process or thread or that try
96 to force system calls on a victim process will work. Furthermore
97 several of the information and iteration interfaces are limited based
98 on the data that is available in the core file. For example, if the
99 core file is of a process that omits the frame pointer, the ability to
100 iterate the stack will be limited.
101 .Pp
102 Use the
103 .Fn Pgrab_core
104 or
105 .Fn Pfgrab_core
106 function to open a core file. Use the
107 .Fn Pgrab_file
108 function to open an ELF object file.
109 This is useful for obtaining information stored in ELF headers and
110 sections.
111 .Ss Debug Information
112 Many of the operations in the library rely on debug information being
113 present in a process and its associated libraries. The library
114 leverages symbol table information, CTF data
115 .Pf ( Xr CTF 4 )
116 sections, and frame unwinding information based on the use of an ABI
117 defined frame pointer, eg.
118 .Sy %ebp
119 and
120 .Sy %rbp
121 on x86 systems.
122 .Pp

```

123 Some software providers strip programs of this information or build
 124 their executables such that the information will not be present in a
 125 core dump. To deal with this fact, the library is able to consume
 126 information that is not present in the core file or the running
 127 process. It can both consume it from the underlying executable and it
 128 also supports finding it from related ELF objects that are linked to
 129 it via the
 130 .Sy .gnu_debuglink
 131 and the
 132 .Sy .note.gnu.build-id
 133 ELF sections.
 134 .Ss Iteration Interfaces
 135 The
 136 .Nm
 137 library provides the ability to iterate over the following aspects of a
 138 process or core file:
 139 .Bl -bullet -offset indent
 140 .It
 141 Active threads
 142 .It
 143 Active and zombie threads
 144 .It
 145 All non-system processes
 146 .It
 147 All process mappings
 148 .It
 149 All objects in a process
 150 .It
 151 The environment
 152 .It
 153 The symbol table
 154 .It
 155 Stack frames
 156 .It
 157 File Descriptors
 158 .El
 159 .Ss System Call Injection
 160 The
 161 .Nm
 162 library allows the caller to force system calls to be executed in the
 163 context of the running process. This can be used both as a tool for
 164 introspection, allowing one to get information outside its current
 165 context as well as performing modifications to a process.
 166 .Pp
 167 These functions run in the context of the calling process. This is
 168 often an easier way of getting non-exported information about a
 169 process from the system. For example, the
 170 .Xr pfiles 1
 171 command uses this interface to get more detailed information about a
 172 process's open file descriptors, which it would not have access to
 173 otherwise.
 174 .Sh INTERFACES
 175 The shared object
 176 .Sy libproc.so.1
 177 provides the public interfaces defined below. See
 178 .Xr Intro 3
 179 for additional information on shared object interfaces. Functions are
 180 organized into categories that describe their purpose. Individual
 181 functions are documented in their own manual pages.
 182 .Ss Creation, Grabbing, and Releasing
 183 The following routines are related to creating library handles,
 184 grabbing cores, processes, and threads, and releasing those resources.
 185 .Bl -column -offset indent ".Sy Pmapping_iter_resolved" ".Sy Psymbol_iter_by_add
 186 .It Sy Lfree Ta Sy Lgrab
 187 .It Sy Lgrab_error Ta Sy Pcreate
 188 .It Sy Pcreate_agent Ta Sy Pcreate_callback

189 .It Sy Pcreate_error Ta Sy Pdestroy_agent
 190 .It Sy Pgrab_core Ta Sy Pfree
 191 .It Sy Pgrab Ta Sy Pgrab_core
 192 .It Sy Pgrab_error Ta Sy Pgrab_file
 193 .It Sy Pgrab_ops Ta Sy Prelease
 194 .It Sy Preopen Ta Sy Pxcreate
 195 .El
 196 .Ss Process interrogation and manipulation
 197 The following routines obtain information about a process and allow
 198 manipulation of the process itself.
 199 .Bl -column -offset indent ".Sy Pmapping_iter_resolved" ".Sy Psymbol_iter_by_add
 200 .It Sy Paddr_to_ctf Ta Sy Paddr_to_loadobj
 201 .It Sy Paddr_to_map Ta Sy Paddr_to_text_map
 202 .It Sy Pasfd Ta Sy Pclearfault
 203 .It Sy Pclearsig Ta Sy Pcontent
 204 .It Sy Pcred Ta Sy Pctldf
 205 .It Sy Pdelbkpt Ta Sy Pdelwapt
 206 .It Sy Pdstop Ta Sy Pexecname
 207 .It Sy Pfault Ta Sy Pfgcore
 208 .It Sy Pgcore Ta Sy Pgetareg
 209 .It Sy Pgetauxval Ta Sy Pgetauxvec
 210 .It Sy Pgetenv Ta Sy Pispdir
 211 .It Sy Pissyscall_prev Ta Sy Plmid
 212 .It Sy Plmid_to_loadobj Ta Sy Plmid_to_map
 213 .It Sy Plookup_by_addr Ta Sy Plookup_by_name
 214 .It Sy Plwp_alt_stack Ta Sy Plwp_getfregs
 215 .It Sy Plwp_getpsinfo Ta Sy Plwp_getregs
 216 .It Sy Plwp_getspymaster Ta Sy Plwp_main_stack
 217 .It Sy Plwp_setfregs Ta Sy Plwp_setregs
 218 .It Sy Plwp_stack Ta Sy Pname_to_ctf
 219 .It Sy Pname_to_loadobj Ta Sy Pname_to_map
 220 .It Sy Pobjname Ta Sy Pobjname_resolved
 221 .It Sy Pplatform Ta Sy Ppltdest
 222 .It Sy Ppriv Ta Sy Ppsinfo
 223 .It Sy Pputareg Ta Sy Prd_agent
 224 .It Sy Pread Ta Sy Pread_string
 225 .It Sy Preset_maps Ta Sy Psetbkpt
 226 .It Sy Psetflags Ta Sy Psetcred
 227 .It Sy Psetfault Ta Sy Psetflags
 228 .It Sy Psetpriv Ta Sy Psetrun
 229 .It Sy Psetsignal Ta Sy Psetsysentry
 230 .It Sy Psetsexit Ta Sy Psetwapt
 231 .It Sy Psetzoneid Ta Sy Psignal
 232 .It Sy Pstate Ta Sy Pstatus
 233 .It Sy Pstop Ta Sy Pstopstatus
 234 .It Sy Psync Ta Sy Psysentry
 235 .It Sy Psysexit Ta Sy Puname
 236 .It Sy Punsetflags Ta Sy Pupdate_maps
 237 .It Sy Pupdate_syms Ta Sy Pwait
 238 .It Sy Pwrite Ta Sy Pxecbkpt
 239 .It Sy Pxecwapt Ta Sy Pxlookup_by_addr
 240 .It Sy Pxlookup_by_addr_resolved Ta Sy Pxlookup_by_name
 241 .It Sy Pzonename Ta Sy Pzonepath
 242 .It Sy Pzoneroot Ta
 243 .It Sy Psetcred Ta Sy Psetfault
 244 .It Sy Psetflags Ta Sy Psetpriv
 245 .It Sy Psetrun Ta Sy Psetsignal
 246 .It Sy Psetsysentry Ta Sy Psetsexit
 247 .It Sy Psetwapt Ta Sy Psetzoneid
 248 .It Sy Psignal Ta Sy Pstate
 249 .It Sy Pstatus Ta Sy Pstop
 250 .It Sy Pstopstatus Ta Sy Psync
 251 .It Sy Psysentry Ta Sy Psysexit
 252 .It Sy Puname Ta Sy Punsetflags
 253 .It Sy Pupdate_maps Ta Sy Pupdate_syms
 254 .It Sy Pwait Ta Sy Pwrite

```

238 .It Sy Pxecbkpt Ta Sy Pxecwapt
239 .It Sy Pxlookup_by_addr Ta Sy Pxlookup_by_addr_resolved
240 .It Sy Pxlookup_by_name Ta Sy Pzonename
241 .It Sy Pzonepath Ta Sy Pzoneroot
242 .El
243 .El
244 .Ss Thread interrogation and manipulation
245 The following routines obtain information about a thread and allow
246 manipulation of the thread itself.
247 .Bl -column -offset indent ".Sy Pmapping_iter_resolved" ".Sy Psymbol_iter_by_add
248 .It Sy Lalt_stack Ta Sy Lclearfault
249 .It Sy Lclearsig Ta Sy Lctlfd
250 .It Sy Ldstop Ta Sy Lgetareg
251 .It Sy Lmain_stack Ta Sy Lprochandle
252 .It Sy Lpsinfo Ta Sy Lputareg
253 .It Sy Lsetrun Ta Sy Lstack
254 .It Sy Lstate Ta Sy Lstatus
255 .It Sy Lstop Ta Sy Lsync
256 .It Sy Lwait Ta Sy Lxecbkpt
257 .It Sy Lxecwapt Ta ""
258 .El
259 .Ss System Call Injection
260 The following routines are used to inject specific system calls and have
261 them run in the context of a process.
262 .Bl -column -offset indent ".Sy Pmapping_iter_resolved" ".Sy Psymbol_iter_by_add
263 .It Sy pr_access Ta Sy pr_close
264 .It Sy pr_creat Ta Sy pr_door_info
265 .It Sy pr_exit Ta Sy pr_fcntl
266 .It Sy pr_fstat Ta Sy pr_fstat64
267 .It Sy pr_fstatvfs Ta Sy pr_getitimer
268 .It Sy pr_getpeername Ta Sy pr_getpeerucred
269 .It Sy pr_getprojid Ta Sy pr_getrctl
270 .It Sy pr_getrlimit Ta Sy pr_getrlimit64
271 .It Sy pr_getsockname Ta Sy pr_getsockopt
272 .It Sy pr_gettaskid Ta Sy pr_getzoneid
273 .It Sy pr_ioctl Ta Sy pr_link
274 .It Sy pr_llseek Ta Sy pr_lseek
275 .It Sy pr_lstat Ta Sy pr_lstat64
276 .It Sy pr_memcntl Ta Sy pr_meminfo
277 .It Sy pr_mmap Ta Sy pr_munmap
278 .It Sy pr_open Ta Sy pr_processor_bind
279 .It Sy pr_rename Ta Sy pr_setitimer
280 .It Sy pr_setrctl Ta Sy pr_setrlimit
281 .It Sy pr_setrlimit64 Ta Sy pr_settaskid
282 .It Sy pr_sigaction Ta Sy pr_stat
283 .It Sy pr_stat64 Ta Sy pr_statvfs
284 .It Sy pr_unlink Ta Sy pr_waitid
285 .El
286 .Ss Iteration routines
287 These routines are used to iterate over the contents of a process.
288 .Bl -column -offset indent ".Sy Pmapping_iter_resolved" ".Sy Psymbol_iter_by_add
289 .It Sy Penv_iter Ta Sy Plwp_iter
290 .It Sy Plwp_iter_all Ta Sy Pmapping_iter
291 .It Sy Pmapping_iter_resolved Ta Sy Pobject_iter
292 .It Sy Pobject_iter_resolved Ta Sy Pstack_iter
293 .It Sy Psymbol_iter Ta Sy Psymbol_iter_by_addr
294 .It Sy Psymbol_iter_by_lmid Ta Sy Psymbol_iter_by_name
295 .It Sy Pxsymbol_iter Ta Sy Pfdinfo_iter
296 .El
297 .Ss Utility routines
298 The following routines are utilities that are useful to consumers of the
299 library.
300 .Bl -column -offset indent ".Sy Pmapping_iter_resolved" ".Sy Psymbol_iter_by_add
301 .It Sy Perror_printf Ta Sy proc_arg_grab
302 .It Sy proc_arg_psinfo Ta Sy proc_arg_xgrab
303 .It Sy proc_arg_xpsinfo Ta Sy proc_content2str
304 .It Sy proc_finistdio Ta Sy proc_fltname

```

```

305 .It Sy proc_fltset2str Ta Sy proc_flushstdio
306 .It Sy proc_get_auxv Ta Sy proc_get_cred
307 .It Sy proc_get_priv Ta Sy proc_get_psinfo
308 .It Sy proc_get_status Ta Sy proc_initstdio
309 .It Sy proc_lwp_in_set Ta Sy proc_lwp_range_valid
310 .It Sy proc_signame Ta Sy proc_sigset2str
311 .It Sy proc_str2content Ta Sy proc_str2flt
312 .It Sy proc_str2fltset Ta Sy proc_str2sig
313 .It Sy proc_str2sigset Ta Sy proc_str2sys
314 .It Sy proc_str2sysset Ta Sy proc_sysname
315 .It Sy proc_sysset2str Ta Sy proc_unctrl_psinfo
316 .It Sy proc_walk Ta ""
317 .El
318 .Ss x86 Specific Routines
319 The following routines are specific to the x86, 32-bit and 64-bit,
320 versions of the
321 .Nm
322 library.
323 .Bl -column -offset indent ".Sy Pmapping_iter_resolved" ".Sy Psymbol_iter_by_add
324 .It Sy Pldt Ta Sy proc_get_ldt
325 .El
326 .Ss SPARC specific Routines
327 The following functions are specific to the SPARC, 32-bit and 64-bit,
328 versions of the
329 .Nm
330 library.
331 .Bl -column -offset indent ".Sy Pmapping_iter_resolved" ".Sy Psymbol_iter_by_add
332 .It Sy Plwp_getgwindows Ta Sy Plwp_getxregs
333 .It Sy Plwp_setxregs Ta Sy ""
334 .El
335 .Pp
336 The following functions are specific to the 64-bit SPARC version of the
337 .Nm
338 library.
339 .Bl -column -offset indent ".Sy Pmapping_iter_resolved" ".Sy Psymbol_iter_by_add
340 .It Sy Plwp_getasrs Ta Sy Plwp_setasrs
341 .El
342 .Sh PROCESS STATES
343 Every process handle that exists in
344 .Nm
345 has a state. In some cases, such as for core files, these states are
346 static. In other cases, such as handles that correspond to a
347 running process or a created process, these states are dynamic and
348 change based on actions taken in the library. The state can be obtained
349 with the
350 .Xr Pstate 3PROC
351 function.
352 .Pp
353 The various states are:
354 .Bl -tag -width Dv -offset indent
355 .It Dv PS_RUN
356 An actively running process. This may be a process that was obtained
357 by creating it with functions such as
358 .Xr Pcreate 3PROC
359 or by grabbing an existing process such as
360 .Xr Pgrab 3PROC .
361 .It Dv PS_STOP
362 An active process that is no longer executing. A process may stop for
363 many reasons such as an explicit stop request (through
364 .Xr pstop 1
365 for example) or if a tracing event is hit.
366 .Pp
367 The reason a process is stopped may be obtained through the thread's
368 .Sy lwpstatus_t
369 structure read directly from /proc or obtained through the
370 .Xr Lstatus 3PROC

```

```

371 function.
372 .It Dv PS_LOST
373 Control over the process has been lost. This may happen when the
374 process executes a new image requiring a different set of privileges.
375 To resume control call
376 .Xr Preopen 3PROC . For more information on losing control of a process,
377 see
378 .Xr proc 4 .
379 .It Dv PS_UNDEAD
380 A zombie process. It has terminated, but it has not been cleaned up
381 yet by its parent. For more on the conditions of becoming a zombie,
382 see
383 .Xr exec 2 .
384 .It Dv PS_DEAD
385 Processes in this state are always core files. See the earlier section
386 .Sx Core Files
387 for more information on working with core files.
388 .It Dv PS_IDLE
389 A process that has never been run. This is always the case for handles
390 that refer to files as the files cannot be executed. Those process
391 handles are obtained through calling
392 .Xr Pgrab_file 3PROC .
393 .El
394 .Pp
395 Many functions relating to tracing processes, for example
396 .Xr Psignal 3PROC ,
397 .Xr Psetsignal 3PROC ,
398 .Xr Psetfault 3PROC ,
399 .Xr Psetentry 3PROC ,
400 and others, mention that they only act upon
401 .Em Active Processes .
402 This specifically refers to processes whose state are in
403 .Dv PS_RUN
404 and
405 .Dv PS_STOP .
406 Process handles in the other states have no notion of settable tracing
407 flags, though core files
408 .Pf ( type Dv PS_DEAD ) ,
409 =====
410 may have a read-only snapshot of their tracing settings available.
411 .Sh TYPES
412 The
413 .Nm
414 library uses many types that come from the /proc file system
415 .Pf ( Xr proc 4 )
416 and the ELF format
417 .Pf ( Xr elf 3ELF ) .
418 However, it also defines the following types:
419 .Pp
420 .Sy struct ps_prochandle
421 .Pp
422 The
423 .Sy struct ps_prochandle
424 is an opaque handle to the library and the core element of control for a
425 process. Consumers obtain pointers to a handle through the use of the
426 .Fn Pcreate ,
427 .Fn Pgrab ,
428 and related functions. When a caller is done with a handle, then it
429 should call one of the
430 .Fn Pfree
431 and
432 .Fn Prelease
433 functions to relinquish the handle, release associated resources, and
434 potentially set the process to run again.
435 .Pp
436 .Sy struct ps_lwphandle

```

```

437 .Pp
438 The
439 .Sy struct ps_lwphandle
440 is analogous to the
441 .Sy struct ps_prochandle ,
442 but it represents the control of an individual thread, rather than a
443 process. Consumers obtain pointers to a handle through the
444 .Fn Lgrab
445 function and relinquish it with the
446 .Fn Lfree
447 function.
448 .Pp
449 .Sy core_content_t
450 .Pp
451 The
452 .Sy core_content_t
453 is a value which describes the various content types of core files.
454 These are used in functions such as
455 .Xr Pcontent 3PROC
456 and
457 .Xr Pgc core 3PROC
458 to describe and control the types of content that get included. Various
459 content types may be included together through a bitwise-inclusive-OR.
460 The default system core contents are controlled with the
461 .Xr coreadm 1M
462 tool. The following table lists the current set of core contents in the
463 system, though the set may increase over time. The string after the
464 macro is the human readable string that corresponds with the constant
465 and is used by
466 .Xr coreadm 1M ,
467 .Xr proc_content2str 3PROC ,
468 and
469 .Xr proc_str2content 3PROC .
470 .Bl -tag -offset indent -width indent
471 .It Dv CC_CONTENT_STACK ("stack")
472 The contents include the process stack. Note, this only covers the main
473 thread's stack. The stack of other threads is covered by
474 .Dv CC_CONTENT_ANON .
475 .It Dv CC_CONTENT_HEAP ("heap")
476 The contents include the process heap.
477 .It Dv CC_CONTENT_SHFILE ("shfile")
478 The contents include shared mappings that are backed by files (e.g.
479 mapped through
480 .Xr mmap 2
481 with the
482 .Dv MAP_SHARED
483 flag).
484 .It Dv CC_CONTENT_SHANNON ("shannon")
485 The contents include shared mappings that are backed by anonymous memory
486 (e.g. mapped through
487 .Xr mmap 2
488 with the
489 .Dv MAP_SHARED
490 and
491 .Dv MAP_ANON
492 flags).
493 .It Dv CC_CONTENT_RODATA ("rodata")
494 The contents include private read-only file mappings, such as shared
495 library text.
496 .It Dv CC_CONTENT_ANON ("anon")
497 The contents include private anonymous mappings. This includes the
498 stacks of threads which are not the main thread.
499 .It Dv CC_CONTENT_SHM ("shm")
500 The contents include system V shared memory.
501 .It Dv CC_CONTENT_ISM ("ism")
502 The contents include ISM (intimate shared memory) mappings.

```

```

503 .It Dv CC_CONTENT_DISM ("dism")
504 The contents include DISM (dynamic shared memory) mappings.
505 .It Dv CC_CONTENT_CTF ("ctf")
506 The contents include
507 .Xr ctf 4
508 (Compact C Type Format) information. Note, not all objects in the
509 process may have CTF information available.
510 .It Dv CC_CONTENT_SYMTAB ("symtab")
511 The contents include the symbol table. Note, not all objects in the
512 process may have a symbol table available.
513 .It Dv CC_CONTENT_ALL ("all")
514 This value indicates that all of the above content values are present.
515 Note that additional values may be added in the future, in which case
516 the value of the symbol will be updated to include them. Comparisons
517 with
518 .Dv CC_CONTENT_ALL
519 should validate all the expected bits are set by an expression such as
520 .Li (c & CC_CONTENT_ALL) == CC_CONTENT_ALL .
521 .It Dv CC_CONTENT_NONE ("none")
522 This value indicates that there is no content present.
523 .It Dv CC_CONTENT_DEFAULT ("default")
524 The content includes the following set of default values:
525 .Dv CC_CONTENT_STACK ,
526 .Dv CC_CONTENT_HEAP ,
527 .Dv CC_CONTENT_ISM ,
528 .Dv CC_CONTENT_DISM ,
529 .Dv CC_CONTENT_SHM ,
530 .Dv CC_CONTENT_SHANON ,
531 .Dv CC_CONTENT_TEXT ,
532 .Dv CC_CONTENT_DATA ,
533 .Dv CC_CONTENT_RODATA ,
534 .Dv CC_CONTENT_ANON ,
535 .Dv CC_CONTENT_CTF ,
536 and
537 .Dv CC_CONTENT_SYMTAB.
538 Note that the default may change. Comparisons with CC_CONTENT_DEFAULT
539 should validate that all of the expected bits are set with an expression
540 such as
541 .Li (c\ &\ CC_CONTENT_DEFAULT)\ ==\ CC_CONTENT_DEFAULT.
542 .It Dv CC_CONTENT_INVALID
543 This indicates that the contents are invalid.
544 .El
545 .Pp
546 .Sy prfdinfo_t
547 .Pp
548 The
549 .Sy prfdinfo_t
550 structure is used with the
551 .Fn Pfdinfo_iter
552 function which describes information about a file descriptor. The
553 structure is defined as follows:
554 .Bd -literal
555 typedef struct prfdinfo {
556     int         pr_fd;
557     mode_t      pr_mode;
558     uid_t       pr_uid;
559     gid_t       pr_gid;
560     major_t     pr_major;    /* think stat.st_dev */
561     minor_t     pr_minor;
562     major_t     pr_rmajor;   /* think stat.st_rdev */
563     minor_t     pr_rminor;
564     ino64_t     pr_ino;
565     off64_t     pr_offset;
566     off64_t     pr_size;
567     int         pr_fileflags; /* fcntl(F_GETXFL), etc */
568     int         pr_fdflags;  /* fcntl(F_GETFD), etc. */

```

```

569     char        pr_path[MAXPATHLEN];
570 } prfdinfo_t;

```

```

655 .Ed
656 .Pp
657 The member
658 .Sy prs_object
659 points to a string that contains the name of the object file, if known,
660 that the symbol comes from. The member
661 .Sy prs_name
662 points to the name of the symbol, if known. This may be unknown due to a
663 stripped binary that contains no symbol table. The member
664 .Sy prs_lmid
665 indicates the link map identifier that the symbol was found on. For more
666 information on link map identifiers refer to the
667 .Em Linker and Libraries Guide
668 and
669 .Xr dlopen 3C .
670 .Pp
671 The members
672 .Sy prs_id
673 and
674 .Sy prs_table
675 can be used to determine both the symbol table that the entry came from
676 and which entry in the table it corresponds to. If the value of
677 .Sy prs_table
678 is
679 .Dv PR_SYMTAB
680 then it came from the ELF standard symbol table. However, if it is instead
681 .Dv PR_DYNSYM ,
682 then that indicates that it comes from the process's dynamic section.
683 .Pp
684 .Sy proc_lwp_f
685 .Pp
686 The
687 .Sy proc_lwp_f
688 is a function pointer type that is used with the
689 .Fn Plwp_iter
690 function. It is defined as
691 .Sy typedef
692 .Ft int
693 .Fo proc_lwp_f
694 .Fa "void *"
695 .Fa "const lwpstatus_t *"
696 .Fc .
697 The first argument is a pointer to an argument that the user specifies,
698 while the second has the thread's status information and is defined in
699 .Xr proc 4 .
700 For additional information on using this type, see
701 .Xr Plwp_iter 3PROC .
702 .Pp
703 .Sy proc_lwp_all_f
704 .Pp
705 The
706 .Sy proc_lwp_all_f
707 is a function pointer type that is used with the
708 .Fn Plwp_iter_all
709 function. It is defined as
710 .Sy typedef
711 .Ft int
712 .Fo proc_lwp_all_f
713 .Fa "void *"
714 .Fa "const lwpstatus_t *"
715 .Fa "const lwpsinfo_t *"
716 .Fc .
717 The first argument is a pointer to an argument that the user specifies.

```



```

718 The second and third arguments contain the thread's status and
719 thread-specific
720 .Xr ps 1
721 information respectively. Both structures are defined in
722 .Xr proc 4 .
723 For additional information on using this type, see
724 .Xr Plwp_iter_all 3PROC .
725 .Pp
726 .Sy proc_walk_f
727 .Pp
728 The
729 .Sy proc_walk_f
730 is a function pointer type that is used with the
731 .Fn proc_walk
732 function. It is defined as
733 .Sy typedef
734 .Ft int
735 .Fo proc_walk_f
736 .Fa "psinfo_t *"
737 .Fa "lwpsinfo_t *"
738 .Fa "void *"
739 .Fc .
740 The first argument contains the process
741 .Xr ps 1
742 information and the second argument contains the representative thread's
743 .Xr ps 1
744 information. Both structures are defined in
745 .Xr proc 4 .
746 The final argument is a pointer to an argument that the user specifies.
747 For more information on using this, see
748 .Xr proc_walk 3PROC .
749 .Pp
750 .Sy proc_map_f
751 .Pp
752 The
753 .Sy proc_map_f
754 is a function pointer type that is used with the
755 .Fn Pmapping_iter ,
756 .Fn Pmapping_iter_resolved ,
757 .Fn Pobject_iter ,
758 and
759 .Fn Pobject_iter_resolved
760 functions. It is defined as
761 .Sy typedef
762 .Ft int
763 .Fo proc_map_f
764 .Fa "void *"
765 .Fa "const prmap_t *"
766 .Fa "const char *"
767 .Fc .
768 The first argument is a pointer to an argument that the user specifies.
769 The second argument is describes the mapping information and is defined
770 in
771 .Xr proc 4 .
772 The final argument contains the name of the mapping or object file in
773 question. For additional information on using this type, see
774 .Xr Pmapping_iter 3PROC .
775 .Pp
776 .Sy proc_env_f
777 .Pp
778 The
779 .Sy proc_env_f
780 is a function pointer type that is used with the
781 .Fn Penv_iter
782 function. It is defined as
783 .Sy typedef

```

```

784 .Ft int
785 .Fo proc_env_f
786 .Fa "void *"
787 .Fa "struct ps_prochandle *"
788 .Fa "uintptr_t"
789 .Fa "const char *"
790 .Fc .
791 The first argument is a pointer to an argument that the user specifies.
792 The second argument is a pointer to the
793 .Sy struct ps_prochandle
794 that the callback was passed to. The third argument is the address of
795 the environment variable in the process. The fourth argument is the
796 environment variable. Values in the environment follow the convention of
797 the form
798 .Em variable=value .
799 For more information on environment variables see
800 .Xr exec 2
801 and
802 .Xr environ 5 .
803 For additional information on using this type, see
804 .Xr Penv_iter 3PROC .
805 .Pp
806 .Sy proc_sym_f
807 .Pp
808 The
809 .Sy proc_sym_f
810 is a function pointer type that is used with the
811 .Fn Psymbol_iter ,
812 .Fn Psymbol_iter_by_addr ,
813 .Fn Psymbol_iter_by_name ,
814 and
815 .Fn Psymbol_iter_by_lmid
816 functions. It is defined as
817 .Sy typedef
818 .Ft int
819 .Fo proc_sym_f
820 .Fa "void *"
821 .Fa "const GElf_Sym *"
822 .Fa "const char *"
823 .Fc .
824 The first argument is a pointer to an argument that the user supplies.
825 The second argument is a pointer to the ELF symbol information in a
826 32-bit and 64-bit neutral form. See
827 .Xr elf 3ELF
828 and
829 .Xr gelf 3ELF
830 for more information on it. The final argument points to a character
831 string that has the name of the symbol. For additional information on
832 using this type, see
833 .Xr Psymbol_iter 3PROC ,
834 .Xr Psymbol_iter_by_addr 3PROC ,
835 .Xr Psymbol_iter_by_name 3PROC ,
836 and
837 .Xr Psymbol_iter_by_lmid 3PROC .
838 .Pp
839 .Sy proc_xsym_f
840 .Pp
841 The
842 .Sy proc_xsym_f
843 is a function pointer type that is used with the
844 .Fn Pxsymbol_iter
845 function. It is defined as
846 .Sy typedef
847 .Ft int
848 .Fo proc_xsym_f
849 .Fa "void *"

```

```

850 .Fa "const GElf_Sym *"
851 .Fa "const char *"
852 .Fa "const prsyminfo_t *"
853 .Fc .
854 The first three arguments are identical to those of
855 .Sy proc_sym_f .
856 The final argument contains additional information about the symbol
857 itself. The members of the
858 .Sy prsyminfo_t
859 are defined earlier in this section. For additional information on using
860 this type, see
861 .Xr Pxsymbol_iter 3PROC .
862 .Pp
863 .Sy proc_stack_f
864 .Pp
865 The
866 .Sy proc_stack_f
867 is a function pointer type that is used with the
868 .Fn Pstack_iter
869 function. It is defined as
870 .Sy typedef
871 .Ft int
872 .Fo proc_stack_f
873 .Fa "void *"
874 .Fa "prgregset_t"
875 .Fa "uint_t"
876 .Fa "const long *"
877 .Fc .
878 The first argument is a pointer to an argument that the user specifies.
879 The second argument's contents are platform specific. The registers that
880 contain stack information, usually the stack pointer and frame pointer,
881 will be filled in to point to an entry. The
882 .Sy prgregset_t
883 is defined in
884 .Xr proc 4 .
885 .Pp
886 The third argument contains the number of arguments to the current stack
887 frame and the fourth argument contains an array of addresses that
888 correspond to the arguments to that stack function. The value of the
889 third argument dictates the number of entries in the fourth argument.
890 For additional information on using this type, see
891 .Xr Pstack_iter 3PROC .
892 .Pp
893 .Sy proc_fdinfo_f
894 .Pp
895 The
896 .Sy proc_fdinfo_f
897 is a function pointer type that is used with the
898 .Fn Pfdinfo_iter
899 function. It is defined as
900 .Sy typedef
901 .Ft int
902 .Fo proc_fdinfo_f
903 .Fa "void *"
904 .Fa "prfdinfo_t *"
905 .Fc .
906 The first argument is a pointer to an argument that the user specifies.
907 The second argument contains information about an open file descriptor.
908 The members of the
909 .Sy prfdinfo_t
910 are defined earlier in this section. For additional information on using
911 this type, see
912 .Xr Pfdinfo_iter 3PROC .
913 .Sh PROGRAMMING NOTES
914 When working with live processes, whether from the
915 .Xr Pgrab

```

```

916 or
917 .Xr Pcreate
918 family of functions, there are some additional considerations.
919 Importantly, if a process calls any of the
920 .Xr exec 2
921 suite of functions, much of the state information that is obtained,
922 particularly that about mappings in the process will be invalid. Callers
923 must ensure that they call
924 .Xr Preset_maps 3PROC
925 when they hold a process handle across an exec. In addition, users of
926 the library should familiarize themselves with the
927 .Sy PROGRAMMING NOTES
928 section of the
929 .Xr proc 4
930 manual page, which discusses issues of privileges and security.
931 .Sh DEBUGGING
932 The library provides a means for obtaining additional debugging
933 information. The output itself is not part of the
934 .Nm
935 library's stable interface. Setting the environment variable
936 .Ev LIBPROC_DEBUG
937 to some value will print information to standard error. For example,
938 .Ev LIBPROC_DEBUG Ns = Ns Em please .
939 .Sh LOCKING
940 Most functions operate on a handle to a process in the form of a
941 .Vt "struct ps_prochandle *" .
942 Unless otherwise indicated, the library does not provide any
943 synchronization for different routines that are operating on the
944 .Sy same
945 .Nm
946 library handle. It is up to the caller to ensure that only a single
947 thread is using a handle at any given time. Multiple threads may call
948 .Nm
949 library routines at the same time as long as each thread is using a
950 different handle.
951 .Pp
952 Each individual function notes its
953 .Sy MT-Level
954 section. The MT-Level of a routine that matches the above description
955 will refer to this manual page. If it does not, then it refers to the
956 standard attributes in
957 .Xr attributes 5 .
958 .Sh INTERFACE STABILITY
959 .Sy Uncommitted
960 .Pp
961 While the library is considered an uncommitted interface, and is still
962 evolving, changes that break compatibility have been uncommon and this
963 trend is expected to continue. It is documented to allow consumers,
964 whether part of illumos or outside of it, to understand the library and
965 make use of it with the understanding that changes may occur which break
966 both source and binary compatibility.
967 .Sh SEE ALSO
968 .Xr gcore 1 ,
969 .Xr mdb 1 ,
970 .Xr proc 1 ,
971 .Xr ps 1 ,
972 .Xr coreadm 1M ,
973 .Xr exec 2 ,
974 .Xr fcntl 2 ,
975 .Xr stat 2 ,
976 .Xr Intro 3 ,
977 .Xr dlopen 3C ,
978 .Xr elf 3ELF ,
979 .Xr ctf 4 ,
980 .Xr proc 4 ,
981 .Xr attributes 5 ,

```

```

982 .Xr environ 5 ,
983 .Xr privileges 5
984 .Pp
985 .Rs
986 .%T Linkers and Libraries Guide
987 .Re
988 .Pp
989 .Xr Lfree 3PROC ,
990 .Xr Lgrab 3PROC ,
991 .Xr Lgrab_error 3PROC ,
992 .Xr Pcreate 3PROC ,
993 .Xr Pcreate_agent 3PROC ,
994 .Xr Pcreate_callback 3PROC ,
995 .Xr Pcreate_error 3PROC ,
996 .Xr Pdestroy_agent 3PROC ,
997 .Xr Pfggrab_core 3PROC ,
998 .Xr Pfree 3PROC ,
999 .Xr Pgrab 3PROC ,
1000 .Xr Pgrab_core 3PROC ,
1001 .Xr Pgrab_error 3PROC ,
1002 .Xr Pgrab_file 3PROC ,
1003 .Xr Pgrab_ops 3PROC ,
1004 .Xr Prelease 3PROC ,
1005 .Xr Preopen 3PROC ,
1006 .Xr Pxcreate 3PROC
1007 .Pp
1008 .Xr Paddr_to_ctf 3PROC ,
1009 .Xr Paddr_to_loadobj 3PROC ,
1010 .Xr Paddr_to_map 3PROC ,
1011 .Xr Paddr_to_text_map 3PROC ,
1012 .Xr Pasfd 3PROC ,
1013 .Xr Pclearfault 3PROC ,
1014 .Xr Pclearsig 3PROC ,
1015 .Xr Pcontent 3PROC ,
1016 .Xr Pcred 3PROC ,
1017 .Xr Pctlfd 3PROC ,
1018 .Xr Pdelbkpt 3PROC ,
1019 .Xr Pdelwapt 3PROC ,
1020 .Xr Pdstop 3PROC ,
1021 .Xr Pexecname 3PROC ,
1022 .Xr Pfault 3PROC ,
1023 .Xr Pfgcore 3PROC ,
1024 .Xr Pgcure 3PROC ,
1025 .Xr Pgetareg 3PROC ,
1026 .Xr Pgetauxval 3PROC ,
1027 .Xr Pgetauxvec 3PROC ,
1028 .Xr Pgetenv 3PROC ,
1029 .Xr Pissprocdir 3PROC ,
1030 .Xr Pissyscall_prev 3PROC ,
1031 .Xr Plmid 3PROC ,
1032 .Xr Plmid_to_loadobj 3PROC ,
1033 .Xr Plmid_to_map 3PROC ,
1034 .Xr Plookup_by_addr 3PROC ,
1035 .Xr Plookup_by_name 3PROC ,
1036 .Xr Plwp_alt_stack 3PROC ,
1037 .Xr Plwp_getfpregs 3PROC ,
1038 .Xr Plwp_getpsinfo 3PROC ,
1039 .Xr Plwp_getregs 3PROC ,
1040 .Xr Plwp_getspymaster 3PROC ,
1041 .Xr Plwp_main_stack 3PROC ,
1042 .Xr Plwp_setfpregs 3PROC ,
1043 .Xr Plwp_setregs 3PROC ,
1044 .Xr Plwp_stack 3PROC ,
1045 .Xr Pname_to_ctf 3PROC ,
1046 .Xr Pname_to_loadobj 3PROC ,
1047 .Xr Pname_to_map 3PROC ,

```

```

1048 .Xr Pobjname 3PROC ,
1049 .Xr Pobjname_resolved 3PROC ,
1050 .Xr Pplatform 3PROC ,
1051 .Xr Ppltdest 3PROC ,
1052 .Xr Ppriv 3PROC ,
1053 .Xr Ppsinfo 3PROC ,
1054 .Xr Pputareg 3PROC ,
1055 .Xr Prd_agent 3PROC ,
1056 .Xr Pread 3PROC ,
1057 .Xr Pread_string 3PROC ,
1058 .Xr Preset_maps 3PROC ,
1059 .Xr Psecflags 3PROC ,
1060 #endif /* ! codereview */
1061 .Xr Psetbkpt 3PROC ,
1062 .Xr Psetcred 3PROC ,
1063 .Xr Psetfault 3PROC ,
1064 .Xr Psetflags 3PROC ,
1065 .Xr Psetpriv 3PROC ,
1066 .Xr Psetrun 3PROC ,
1067 .Xr Psetsignal 3PROC ,
1068 .Xr Psetsysentry 3PROC ,
1069 .Xr Psetsysexit 3PROC ,
1070 .Xr Psetwapt 3PROC ,
1071 .Xr Psetzoneid 3PROC ,
1072 .Xr Psignal 3PROC ,
1073 .Xr Pstate 3PROC ,
1074 .Xr Pstatus 3PROC ,
1075 .Xr Pstop 3PROC ,
1076 .Xr Pstopstatus 3PROC ,
1077 .Xr Psync 3PROC ,
1078 .Xr Psyseentry 3PROC ,
1079 .Xr Psysexit 3PROC ,
1080 .Xr Puname 3PROC ,
1081 .Xr Punsetflags 3PROC ,
1082 .Xr Pupdate_maps 3PROC ,
1083 .Xr Pupdate_syms 3PROC ,
1084 .Xr Pwait 3PROC ,
1085 .Xr Pwrite 3PROC ,
1086 .Xr Pxecbkpt 3PROC ,
1087 .Xr Pxecwapt 3PROC ,
1088 .Xr Pxlookup_by_addr 3PROC ,
1089 .Xr Pxlookup_by_addr_resolved 3PROC ,
1090 .Xr Pxlookup_by_name 3PROC ,
1091 .Xr Pzonename 3PROC ,
1092 .Xr Pzonepath 3PROC ,
1093 .Xr Pzoneroot 3PROC
1094 .Pp
1095 .Xr Lalt_stack 3PROC ,
1096 .Xr Lclearfault 3PROC ,
1097 .Xr Lclearsig 3PROC ,
1098 .Xr Lctlfd 3PROC ,
1099 .Xr Ldstop 3PROC ,
1100 .Xr Lgetareg 3PROC ,
1101 .Xr Lmain_stack 3PROC ,
1102 .Xr Lprochandle 3PROC ,
1103 .Xr Lpsinfo 3PROC ,
1104 .Xr Lputareg 3PROC ,
1105 .Xr Lsetrun 3PROC ,
1106 .Xr Lstack 3PROC ,
1107 .Xr Lstate 3PROC ,
1108 .Xr Lstatus 3PROC ,
1109 .Xr Lstop 3PROC ,
1110 .Xr Lsync 3PROC ,
1111 .Xr Lwait 3PROC ,
1112 .Xr Lxecbkpt 3PROC ,
1113 .Xr Lxecwapt 3PROC

```

```

1114 .Pp
1115 .Xr pr_access 3PROC ,
1116 .Xr pr_close 3PROC ,
1117 .Xr pr_creat 3PROC ,
1118 .Xr pr_door_info 3PROC ,
1119 .Xr pr_exit 3PROC ,
1120 .Xr pr_fcntl 3PROC ,
1121 .Xr pr_fstat 3PROC ,
1122 .Xr pr_fstat64 3PROC ,
1123 .Xr pr_fstatvfs 3PROC ,
1124 .Xr pr_getitimer 3PROC ,
1125 .Xr pr_getpeername 3PROC ,
1126 .Xr pr_getpeerucred 3PROC ,
1127 .Xr pr_getprojid 3PROC ,
1128 .Xr pr_getrctl 3PROC ,
1129 .Xr pr_getrlimit 3PROC ,
1130 .Xr pr_getrlimit64 3PROC ,
1131 .Xr pr_getsockname 3PROC ,
1132 .Xr pr_getsockopt 3PROC ,
1133 .Xr pr_gettaskid 3PROC ,
1134 .Xr pr_getzoneid 3PROC ,
1135 .Xr pr_ioctl 3PROC ,
1136 .Xr pr_link 3PROC ,
1137 .Xr pr_llseek 3PROC ,
1138 .Xr pr_lseek 3PROC ,
1139 .Xr pr_lstat 3PROC ,
1140 .Xr pr_lstat64 3PROC ,
1141 .Xr pr_memcntl 3PROC ,
1142 .Xr pr_meminfo 3PROC ,
1143 .Xr pr_mmap 3PROC ,
1144 .Xr pr_munmap 3PROC ,
1145 .Xr pr_open 3PROC ,
1146 .Xr pr_processor_bind 3PROC ,
1147 .Xr pr_rename 3PROC ,
1148 .Xr pr_setitimer 3PROC ,
1149 .Xr pr_setrctl 3PROC ,
1150 .Xr pr_setrlimit 3PROC ,
1151 .Xr pr_setrlimit64 3PROC ,
1152 .Xr pr_settaskid 3PROC ,
1153 .Xr pr_sigaction 3PROC ,
1154 .Xr pr_stat 3PROC ,
1155 .Xr pr_stat64 3PROC ,
1156 .Xr pr_statvfs 3PROC ,
1157 .Xr pr_unlink 3PROC ,
1158 .Xr pr_waitid 3PROC ,
1159 .Pp
1160 .Xr Penv_iter 3PROC ,
1161 .Xr Plwp_iter 3PROC ,
1162 .Xr Plwp_iter_all 3PROC ,
1163 .Xr Pmapping_iter 3PROC ,
1164 .Xr Pmapping_iter_resolved 3PROC ,
1165 .Xr Pobject_iter 3PROC ,
1166 .Xr Pobject_iter_resolved 3PROC ,
1167 .Xr Pstack_iter 3PROC ,
1168 .Xr Psymbol_iter 3PROC ,
1169 .Xr Psymbol_iter_by_addr 3PROC ,
1170 .Xr Psymbol_iter_by_lmid 3PROC ,
1171 .Xr Psymbol_iter_by_name 3PROC ,
1172 .Xr Pxsymbol_iter 3PROC ,
1173 .Xr Pfdinfo_iter 3PROC
1174 .Pp
1175 .Xr Perror_printf 3PROC ,
1176 .Xr proc_arg_grab 3PROC ,
1177 .Xr proc_arg_psinfo 3PROC ,
1178 .Xr proc_arg_xgrab 3PROC ,
1179 .Xr proc_arg_xpsinfo 3PROC ,

```

```

1180 .Xr proc_content2str 3PROC ,
1181 .Xr proc_finistdio 3PROC ,
1182 .Xr procfltname 3PROC ,
1183 .Xr procfltset2str 3PROC ,
1184 .Xr proc_flushstdio 3PROC ,
1185 .Xr proc_get_auxv 3PROC ,
1186 .Xr proc_get_cred 3PROC ,
1187 .Xr proc_get_priv 3PROC ,
1188 .Xr proc_get_psinfo 3PROC ,
1189 .Xr proc_get_status 3PROC ,
1190 .Xr proc_initstdio 3PROC ,
1191 .Xr proc_lwp_in_set 3PROC ,
1192 .Xr proc_lwp_range_valid 3PROC ,
1193 .Xr proc_signame 3PROC ,
1194 .Xr proc_sigset2str 3PROC ,
1195 .Xr proc_str2content 3PROC ,
1196 .Xr proc_str2flt 3PROC ,
1197 .Xr proc_str2fltset 3PROC ,
1198 .Xr proc_str2sig 3PROC ,
1199 .Xr proc_str2sigset 3PROC ,
1200 .Xr proc_str2sys 3PROC ,
1201 .Xr proc_str2sysset 3PROC ,
1202 .Xr proc_sysname 3PROC ,
1203 .Xr proc_sysset2str 3PROC ,
1204 .Xr proc_unctrl_psinfo 3PROC ,
1205 .Xr proc_walk 3PROC
1206 .Pp
1207 .Xr Pldt 3PROC ,
1208 .Xr proc_get_ldt 3PROC ,
1209 .Pp
1210 .Xr Plwp_getgwindows 3PROC ,
1211 .Xr Plwp_getxregs 3PROC ,
1212 .Xr Plwp_setxregs 3PROC ,
1213 .Pp
1214 .Xr Plwp_getasrs 3PROC ,
1215 .Xr Plwp_setasrs 3PROC

```

```

*****
11425 Thu Jun 30 22:02:44 2016
new/usr/src/man/man3proc/Makefile
Code review comments from pmoooney (sundry), and igork (screwups in zonecfg refac)
*****
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
9 # at http://www.illumos.org/license/CDDL.
10 #
11 #
12 #
13 # Copyright 2011, Richard Lowe
14 # Copyright 2013 Nexenta Systems, Inc. All rights reserved.
15 # Copyright 2015 Joyent, Inc.
16 #
17 #
18 include $(SRC)/Makefile.master
19 #
20 MANSECT= 3proc
21 #
22 MANFILES=
23 Lctld.3proc
24 Lfree.3proc
25 Lgrab_error.3proc
26 Lgrab.3proc
27 Lprochandle.3proc
28 Lpsinfo.3proc
29 Lstate.3proc
30 Lstatus.3proc
31 Paddr_to_ctf.3proc
32 Paddr_to_loadobj.3proc
33 Paddr_to_map.3proc
34 Pasfd.3proc
35 Pclearfault.3proc
36 Pclearsig.3proc
37 Pcontent.3proc
38 Pcreate_agent.3proc
39 Pcreate_error.3proc
40 Pcreate.3proc
41 Pcred.3proc
42 Pctld.3proc
43 Pdelbkpt.3proc
44 Pdelwapt.3proc
45 Pdestroy_agent.3proc
46 Penv_iter.3proc
47 Perror_printf.3proc
48 Pexecname.3proc
49 Pfault.3proc
50 Pfdinfo_iter.3proc
51 Pgcov.3proc
52 Pgetareg.3proc
53 Pgetauxval.3proc
54 Pgetauxvec.3proc
55 Pgetenv.3proc
56 Pgrab_core.3proc
57 Pgrab_error.3proc
58 Pgrab_file.3proc
59 Pgrab.3proc
60 Pisprocd.3proc
61 Pissyscall.3proc

```

```

62 Pldt.3proc
63 Plmid.3proc
64 Plookup_by_addr.3proc
65 Plwp_getasrs.3proc
66 Plwp_getgwindows.3proc
67 Plwp_getpsinfo.3proc
68 Plwp_getregs.3proc
69 Plwp_getspymaster.3proc
70 Plwp_getxregs.3proc
71 Plwp_iter.3proc
72 Plwp_stack.3proc
73 Pmapping_iter.3proc
74 Pobjname.3proc
75 Pplatform.3proc
76 Ppltd.3proc
77 Ppriv.3proc
78 Ppsinfo.3proc
79 pr_access.3proc
80 pr_close.3proc
81 pr_creat.3proc
82 pr_door_info.3proc
83 pr_exit.3proc
84 pr_fcntl.3proc
85 pr_fstatvfs.3proc
86 pr_getitimer.3proc
87 pr_getpeername.3proc
88 pr_getpeerucred.3proc
89 pr_getprojid.3proc
90 pr_getrctl.3proc
91 pr_getrlimit.3proc
92 pr_getsockname.3proc
93 pr_getsockopt.3proc
94 pr_gettaskid.3proc
95 pr_getzoneid.3proc
96 pr_ioctl.3proc
97 pr_link.3proc
98 pr_llseek.3proc
99 pr_lseek.3proc
100 pr_memcntl.3proc
101 pr_meminfo.3proc
102 pr_mmap.3proc
103 pr_munmap.3proc
104 pr_open.3proc
105 pr_processor_bind.3proc
106 pr_rename.3proc
107 pr_setitimer.3proc
108 pr_setrctl.3proc
109 pr_setrlimit.3proc
110 pr_settaskid.3proc
111 pr_sigaction.3proc
112 pr_stat.3proc
113 pr_statvfs.3proc
114 pr_unlink.3proc
115 pr_waitid.3proc
116 Prd_agent.3proc
117 Pread.3proc
118 Prelease.3proc
119 Preopen.3proc
120 Preset_maps.3proc
121 proc_arg_grab.3proc
122 proc_arg_psinfo.3proc
123 proc_content2str.3proc
124 proc_fltname.3proc
125 proc_fltset2str.3proc
126 proc_get_auxv.3proc
127 proc_get_cred.3proc

```

```

128     proc_get_priv.3proc      \
129     proc_get_psinfo.3proc   \
130     proc_get_status.3proc   \
131     proc_initstdio.3proc    \
132     proc_lwp_in_set.3proc   \
133     proc_service.3proc      \
134     proc_str2flt.3proc      \
135     proc_str2fltset.3proc   \
136     proc_unctrl_psinfo.3proc \
137     proc_walk.3proc         \
138     Psecflags.3proc         \
139 #endif /* ! codereview */
140     Psetbkpt.3proc          \
141     Psetcred.3proc          \
142     Psetfault.3proc         \
143     Psetflags.3proc         \
144     Psetpriv.3proc          \
145     Psetrun.3proc           \
146     Psetsignal.3proc        \
147     Psetsysentry.3proc      \
148     Psetwapt.3proc          \
149     Psetzoneid.3proc        \
150     Psignal.3proc           \
151     Pstack_iter.3proc       \
152     Pstate.3proc            \
153     Pstatus.3proc           \
154     Pstopstatus.3proc       \
155     Psymbol_iter.3proc      \
156     Psync.3proc             \
157     Psysentry.3proc         \
158     Puname.3proc            \
159     Pupdate_maps.3proc      \
160     Pupdate_syms.3proc     \
161     Pwrite.3proc            \
162     Pxecbkpt.3proc          \
163     Pzonename.3proc         \
164     ps_lgetregs.3proc       \
165     ps_pglobal_lookup.3proc \
166     ps_pread.3proc          \
167     ps_pstop.3proc          \
\
170 MANLINKS= \
171     Lalt_stack.3proc        \
172     Lclearfault.3proc      \
173     Lclearsig.3proc        \
174     Ldstop.3proc           \
175     Lgetareg.3proc         \
176     Lmain_stack.3proc      \
177     Lputareg.3proc         \
178     Lsetrun.3proc          \
179     Lstack.3proc           \
180     Lstop.3proc            \
181     Lsync.3proc            \
182     Lwait.3proc            \
183     Lxecbkpt.3proc         \
184     Lxecwapt.3proc         \
185     Paddr_to_text_map.3proc \
186     Pcreate_callback.3proc \
187     Pdstop.3proc           \
188     Pfgcore.3proc          \
189     Pfggrab_core.3proc     \
190     Pfree.3proc            \
191     Pissyscall_prev.3proc   \
192     Plmid_to_ctf.3proc     \
193     Plmid_to_loadobj.3proc \

```

```

194     Plmid_to_map.3proc     \
195     Plookup_by_name.3proc  \
196     Plwp_alt_stack.3proc   \
197     Plwp_getfpregs.3proc   \
198     Plwp_iter_all.3proc    \
199     Plwp_main_stack.3proc  \
200     Plwp_setasrs.3proc     \
201     Plwp_setfpregs.3proc   \
202     Plwp_setregs.3proc     \
203     Plwp_setxregs.3proc    \
204     Pmapping_iter_resolved.3proc \
205     Pname_to_ctf.3proc     \
206     Pname_to_loadobj.3proc \
207     Pname_to_map.3proc     \
208     Pobject_iter_resolved.3proc \
209     Pobject_iter.3proc     \
210     Pobjname_resolved.3proc \
211     Ppriv_free.3proc       \
212     Pputareg.3proc         \
213     pr_fstat.3proc         \
214     pr_fstat64.3proc       \
215     pr_getrlimit64.3proc   \
216     pr_lstat.3proc         \
217     pr_lstat64.3proc       \
218     pr_setrlimit64.3proc   \
219     pr_stat64.3proc        \
220     Pread_string.3proc     \
221     proc_arg_xgrab.3proc   \
222     proc_arg_xpsinfo.3proc \
223     proc_finistdio.3proc   \
224     proc_flushstdio.3proc  \
225     proc_free_priv.3proc   \
226     proc_get_ldt.3proc     \
227     proc_lwp_range_valid.3proc \
228     proc_signame.3proc     \
229     proc_sigset2str.3proc  \
230     proc_str2content.3proc \
231     proc_str2sig.3proc     \
232     proc_str2sigset.3proc  \
233     proc_str2sys.3proc     \
234     proc_str2sysset.3proc  \
235     proc_sysname.3proc     \
236     proc_sysset2str.3proc  \
237     ps_kill.3proc          \
238     ps_lcontinue.3proc     \
239     ps_lgetfpregs.3proc    \
240     ps_lgetxregs.3proc     \
241     ps_lgetxregsize.3proc  \
242     ps_lrolltoaddr.3proc   \
243     ps_lsetfpregs.3proc    \
244     ps_lsetregs.3proc     \
245     ps_lsetxregs.3proc     \
246     ps_lstop.3proc         \
247     ps_pcontinue.3proc     \
248     ps_phread.3proc        \
249     ps_pwrite.3proc        \
250     ps_pglobal_sym.3proc   \
251     ps_ptread.3proc        \
252     ps_ptwrite.3proc       \
253     ps_pwrite.3proc        \
254     Psetsysexit.3proc     \
255     Pstop.3proc           \
256     Psymbol_iter_by_addr.3proc \
257     Psymbol_iter_by_lmid.3proc \
258     Psymbol_iter_by_name.3proc \
259     Psysexit.3proc         \

```

```

260         Punsetflags.3proc      \
261         Pwait.3proc            \
262         Pxcreate.3proc         \
263         Pxecwapt.3proc        \
264         Pxlookup_by_addr_resolved.3proc \
265         Pxlookup_by_addr.3proc \
266         Pxlookup_by_name.3proc \
267         Pxsymbol_iter.3proc   \
268         Pzonepath.3proc       \
269         Pzoneroot.3proc

272 ps_lgetfpregs.3proc      := LINKSRC = ps_lgetregs.3proc
273 ps_lgetxregs.3proc      := LINKSRC = ps_lgetregs.3proc
274 ps_lgetxregsize.3proc   := LINKSRC = ps_lgetregs.3proc
275 ps_lsetfpregs.3proc     := LINKSRC = ps_lgetregs.3proc
276 ps_lsetregs.3proc       := LINKSRC = ps_lgetregs.3proc
277 ps_lsetxregs.3proc      := LINKSRC = ps_lgetregs.3proc

279 ps_pglobal_sym.3proc    := LINKSRC = ps_pglobal_lookup.3proc

281 ps_pread.3proc          := LINKSRC = ps_pread.3proc
282 ps_pwrite.3proc         := LINKSRC = ps_pread.3proc
283 ps_ptread.3proc         := LINKSRC = ps_pread.3proc
284 ps_ptwrite.3proc        := LINKSRC = ps_pread.3proc
285 ps_pwrite.3proc         := LINKSRC = ps_pread.3proc

287 ps_kill.3proc           := LINKSRC = ps_pstop.3proc
288 ps_lcontinue.3proc      := LINKSRC = ps_pstop.3proc
289 ps_lrolltoaddr.3proc   := LINKSRC = ps_pstop.3proc
290 ps_lstop.3proc          := LINKSRC = ps_pstop.3proc
291 ps_pcontinue.3proc      := LINKSRC = ps_pstop.3proc

294 Pxcreate.3proc          := LINKSRC = Pcreate.3proc
295 Pcreate_callback.3proc := LINKSRC = Pcreate.3proc

297 Pfgrab_core.3proc      := LINKSRC = Pgrab_core.3proc

299 Pfree.3proc             := LINKSRC = Prelease.3proc

301 Plwp_iter_all.3proc     := LINKSRC = Plwp_iter.3proc

303 Pmapping_iter_resolved.3proc := LINKSRC = Pmapping_iter.3proc
304 Pobject_iter.3proc      := LINKSRC = Pmapping_iter.3proc
305 Pobject_iter_resolved.3proc := LINKSRC = Pmapping_iter.3proc

307 Psymbol_iter_by_addr.3proc := LINKSRC = Psymbol_iter.3proc
308 Psymbol_iter_by_lmid.3proc := LINKSRC = Psymbol_iter.3proc
309 Psymbol_iter_by_name.3proc := LINKSRC = Psymbol_iter.3proc
310 Pxsymbol_iter.3proc      := LINKSRC = Psymbol_iter.3proc

312 Plmid_to_ctf.3proc      := LINKSRC = Paddr_to_ctf.3proc
313 Pname_to_ctf.3proc      := LINKSRC = Paddr_to_ctf.3proc

315 Plmid_to_loadobj.3proc  := LINKSRC = Paddr_to_loadobj.3proc
316 Pname_to_loadobj.3proc := LINKSRC = Paddr_to_loadobj.3proc

318 Paddr_to_text_map.3proc := LINKSRC = Paddr_to_map.3proc
319 Plmid_to_map.3proc      := LINKSRC = Paddr_to_map.3proc
320 Pname_to_map.3proc      := LINKSRC = Paddr_to_map.3proc

322 Pdstop.3proc            := LINKSRC = Pstopstatus.3proc
323 Pstop.3proc             := LINKSRC = Pstopstatus.3proc
324 Pwait.3proc             := LINKSRC = Pstopstatus.3proc
325 Ldstop.3proc            := LINKSRC = Pstopstatus.3proc

```

```

326 Lstop.3proc             := LINKSRC = Pstopstatus.3proc
327 Lwait.3proc             := LINKSRC = Pstopstatus.3proc

329 Lsync.3proc             := LINKSRC = Psync.3proc

331 Pfgcore.3proc           := LINKSRC = Pgc core.3proc

333 Pputareg.3proc          := LINKSRC = Pgetareg.3proc
334 Lgetareg.3proc         := LINKSRC = Pgetareg.3proc
335 Lputareg.3proc         := LINKSRC = Pgetareg.3proc

337 Pissyscall_prev.3proc   := LINKSRC = Pissyscall.3proc

339 Pxlookup_by_addr.3proc  := LINKSRC = Plookup_by_addr.3proc
340 Pxlookup_by_addr_resolved.3proc := LINKSRC = Plookup_by_addr.3proc
341 Plookup_by_name.3proc   := LINKSRC = Plookup_by_addr.3proc
342 Pxlookup_by_name.3proc  := LINKSRC = Plookup_by_addr.3proc

344 Plwp_setregs.3proc      := LINKSRC = Plwp_getregs.3proc
345 Plwp_getfpregs.3proc   := LINKSRC = Plwp_getregs.3proc
346 Plwp_setfpregs.3proc   := LINKSRC = Plwp_getregs.3proc

348 Plwp_alt_stack.3proc    := LINKSRC = Plwp_stack.3proc
349 Plwp_main_stack.3proc   := LINKSRC = Plwp_stack.3proc
350 Lalt_stack.3proc        := LINKSRC = Plwp_stack.3proc
351 Lmain_stack.3proc       := LINKSRC = Plwp_stack.3proc
352 Lstack.3proc            := LINKSRC = Plwp_stack.3proc

354 Pobjname_resolved.3proc := LINKSRC = Pobjname.3proc

356 Ppriv_free.3proc        := LINKSRC = Ppriv.3proc

358 Pread_string.3proc      := LINKSRC = Pread.3proc

360 Punsetflags.3proc       := LINKSRC = Psetflags.3proc

362 Psetsysexit.3proc       := LINKSRC = Psetsysentry.3proc

364 Psysexit.3proc          := LINKSRC = Psysentry.3proc

366 Pxecwapt.3proc          := LINKSRC = Pxecbkpt.3proc
367 Lxecbkpt.3proc         := LINKSRC = Pxecbkpt.3proc
368 Lxecwapt.3proc         := LINKSRC = Pxecbkpt.3proc

370 Lclearfault.3proc       := LINKSRC = Pclearfault.3proc

372 Lclearsig.3proc         := LINKSRC = Pclearsig.3proc

374 Lsetrun.3proc           := LINKSRC = Psetrun.3proc

376 Pzonepath.3proc         := LINKSRC = Pzonename.3proc
377 Pzoneroot.3proc         := LINKSRC = Pzonename.3proc

379 pr_fstat.3proc          := LINKSRC = pr_stat.3proc
380 pr_fstat64.3proc        := LINKSRC = pr_stat.3proc
381 pr_lstat.3proc          := LINKSRC = pr_stat.3proc
382 pr_lstat64.3proc        := LINKSRC = pr_stat.3proc
383 pr_stat64.3proc         := LINKSRC = pr_stat.3proc

385 pr_getrlimit64.3proc   := LINKSRC = pr_getrlimit.3proc

387 pr_setrlimit64.3proc   := LINKSRC = pr_setrlimit.3proc

389 proc_arg_xgrab.3proc    := LINKSRC = proc_arg_grab.3proc

391 proc_arg_xpsinfo.3proc  := LINKSRC = proc_arg_psinfo.3proc

```

```
393 proc_str2content.3proc      := LINKSRC = proc_content2str.3proc
395 proc_flushstdio.3proc      := LINKSRC = proc_initstdio.3proc
396 proc_finistdio.3proc       := LINKSRC = proc_initstdio.3proc
398 proc_signame.3proc         := LINKSRC = proc_fltname.3proc
399 proc_sysname.3proc         := LINKSRC = proc_fltname.3proc
401 proc_sigset2str.3proc       := LINKSRC = proc_fltset2str.3proc
402 proc_sysset2str.3proc       := LINKSRC = proc_fltset2str.3proc
404 proc_free_priv.3proc        := LINKSRC = proc_get_priv.3proc
406 proc_lwp_range_valid.3proc := LINKSRC = proc_lwp_in_set.3proc
408 proc_str2sig.3proc          := LINKSRC = proc_str2flt.3proc
409 proc_str2sys.3proc          := LINKSRC = proc_str2flt.3proc
411 proc_str2sigset.3proc       := LINKSRC = proc_str2fltset.3proc
412 proc_str2sysset.3proc       := LINKSRC = proc_str2fltset.3proc
414 proc_get_ldt.3proc          := LINKSRC = Pldt.3proc
416 Plwp_setxregs.3proc        := LINKSRC = Plwp_getxregs.3proc
418 Plwp_setasrs.3proc         := LINKSRC = Plwp_getasrs.3proc
420 .KEEP_STATE:
422 include      $(SRC)/man/Makefile.man
424 install:     $(ROOTMANFILES) $(ROOTMANLINKS)
```



```

*****
1637 Thu Jun 30 22:02:45 2016
new/usr/src/man/man3proc/Psecflags.3proc
Code review comments from jeffpc
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
1 .\"
2 .\" This file and its contents are supplied under the terms of the
3 .\" Common Development and Distribution License ("CDDL"), version 1.0.
4 .\" You may only use this file in accordance with the terms of version
5 .\" 1.0 of the CDDL.
6 .\"
7 .\" A full copy of the text of the CDDL should have accompanied this
8 .\" source. A copy of the CDDL is also available via the Internet at
9 .\" http://www.illumos.org/license/CDDL.
10 .\"
11 .\"
12 .\" Copyright 2016, Richard Lowe.
13 .\"
14 .Dd June 06, 2016
15 .Dt PSECFLAGS 3PROC
16 .Os
17 .Sh NAME
18 .Nm Psecflags ,
19 .Nm Psecflags_free
20 .Nd get and free process security flags
21 .Sh SYNOPSIS
22 .Lb libproc
23 .In libproc.h
24 .Ft int
25 .Fo Psecflags
26 .Fa "struct ps_prochandle *p"
27 .Fa "prsecflags_t **psf"
28 .Fc
29 .Ft void
30 .Fo Psecflags_free
31 .Fa "struct ps_prochandle *p"
32 .Fa "prsecflags_t *psf"
33 .Fc
34 .Sh DESCRIPTION
35 The
36 .Fn Psecflags
37 function obtains the security flags of the process handle
38 .Fa P .
39 The security flags structure will be dynamically allocated and a pointer to it
40 will be placed in
41 .Fa psf .
42 It must be released with a call to
43 .Fn Psecflags_free .
44 The definition of the
45 .Sy prsecflags_t
46 structure is documented in
47 .Xr proc 4 .
48 .Pp
49 The
50 .Fn Psecflags_free
51 function releases the storage in
52 .Fa psf
53 that was allocated as a result of calling
54 .Fn Psecflags .
55 .Sh RETURN VALUES
56 Upon successful completion, the
57 .Fn Psecflags

```

```

58 function returns
59 .Sy 0
60 and
61 .Fa psf
62 is updated with a pointer to the allocated security flags. Otherwise,
63 .Sy -1
64 is returned and
65 .Fa psf
66 is not updated.
67 .Sh INTERFACE STABILITY
68 .Sy Uncommitted
69 .Sh MT-LEVEL
70 See
71 .Sy LOCKING
72 in
73 .Xr libproc 3LIB .
74 .Sh SEE ALSO
75 .Xr libproc 3LIB ,
76 .Xr proc 4 ,
77 .Xr security-flags 5
78 #endif /* !codereview */

```

```

*****
16868 Thu Jun 30 22:02:46 2016
new/usr/src/man/man4/core.4
Code review comments from jeffpc
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
1 \" te
2 .\\ Copyright (C) 2008, Sun Microsystems, Inc. All Rights Reserved.
3 .\\ Copyright 2012 DEY Storage Systems, Inc. All rights reserved.
4 .\\ Copyright (c) 2013, Joyent, Inc. All rights reserved.
5 .\\ Copyright 1989 AT&T
6 .\\ The contents of this file are subject to the terms of the Common Development
7 .\\ You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE or http:
8 .\\ When distributing Covered Code, include this CDDL HEADER in each file and in
9 .TH CORE 4 \"Jun 6, 2016\"
9 .TH CORE 4 \"Mar 31, 2013\"
10 .SH NAME
11 core \\- process core file
12 .SH DESCRIPTION
13 .sp
13 .LP
14 The operating system writes out a core file for a process when the process is
15 terminated due to receiving certain signals. A core file is a disk copy of the
16 contents of the process address space at the time the process received the
17 signal, along with additional information about the state of the process. This
18 information can be consumed by a debugger. Core files can also be generated by
19 applying the \\fbgcore\\fR(1) utility to a running process.
20 .sp
21 .LP
22 Typically, core files are produced following abnormal termination of a process
23 resulting from a bug in the corresponding application. Whatever the cause, the
24 core file itself provides invaluable information to the programmer or support
25 engineer to aid in diagnosing the problem. The core file can be inspected using
26 a debugger such as \\fbdbx\\fR(1) or \\fbmdb\\fR(1) or by applying one of the
27 \\fbproc\\fR(1) tools.
28 .sp
29 .LP
30 The operating system attempts to create up to two core files for each
31 abnormally terminating process, using a global core file name pattern and a
32 per-process core file name pattern. These patterns are expanded to determine
33 the pathname of the resulting core files, and can be configured by
34 \\fBcoreadm\\fR(1M). By default, the global core file pattern is disabled and not
35 used, and the per-process core file pattern is set to \\fBcore\\fR. Therefore, by
36 default, the operating system attempts to create a core file named \\fBcore\\fR
37 in the process's current working directory.
38 .sp
39 .LP
40 A process terminates and produces a core file whenever it receives one of the
41 signals whose default disposition is to cause a core dump. The list of signals
42 that result in generating a core file is shown in \\fBsignal.h\\fR(3HEAD).
43 Therefore, a process might not produce a core file if it has blocked or
44 modified the behavior of the corresponding signal. Additionally, no core dump
45 can be created under the following conditions:
46 .RS +4
47 .TP
48 .ie t \\(bu
49 .el o
50 If normal file and directory access permissions prevent the creation or
51 modification of the per-process core file pathname by the current process user
52 and group ID. This test does not apply to the global core file pathname
53 because, regardless of the UID of the process dumping core, the attempt to
54 write the global core file is made as the superuser.
55 .RE

```

```

56 .RS +4
57 .TP
58 .ie t \\(bu
59 .el o
60 Core files owned by the user \\fBnobody\\fR will not be produced. For example,
61 core files generated for the superuser on an NFS directory are owned by
62 \\fBnobody\\fR and are, therefore, not written.
63 .RE
64 .RS +4
65 .TP
66 .ie t \\(bu
67 .el o
68 If the core file pattern expands to a pathname that contains intermediate
69 directory components that do not exist. For example, if the global pattern is
70 set to \\fB/var/core/%n/core.%p\\fR, and no directory \\fB/var/core/'uname -n'\\fR
71 has been created, no global core files are produced.
72 .RE
73 .RS +4
74 .TP
75 .ie t \\(bu
76 .el o
77 If the destination directory is part of a filesystem that is mounted read-only.
78 .RE
79 .RS +4
80 .TP
81 .ie t \\(bu
82 .el o
83 If the resource limit \\fBRLIMIT_CORE\\fR has been set to \\fB0\\fR for the
84 process, no per-process core file is produced. Refer to \\fBsetrlimit\\fR(2) and
85 \\fBulimit\\fR(1) for more information on resource limits.
86 .RE
87 .RS +4
88 .TP
89 .ie t \\(bu
90 .el o
91 If the core file name already exists in the destination directory and is not a
92 regular file (that is, is a symlink, block or character special-file, and so
93 forth).
94 .RE
95 .RS +4
96 .TP
97 .ie t \\(bu
98 .el o
99 If the kernel cannot open the destination file \\fBEXCL\\fR, which can occur if
100 same file is being created by another process simultaneously.
101 .RE
102 .RS +4
103 .TP
104 .ie t \\(bu
105 .el o
106 If the process's effective user ID is different from its real user ID or if its
107 effective group ID is different from its real group ID. Similarly, set-user-ID
108 and set-group-ID programs do not produce core files as this could potentially
109 compromise system security. These processes can be explicitly granted
110 permission to produce core files using \\fBcoreadm\\fR(1M), at the risk of
111 exposing secure information.
112 .RE
113 .sp
114 .LP
115 The core file contains all the process information pertinent to debugging:
116 contents of hardware registers, process status, and process data. The format of
117 a core file is object file specific.
118 .sp
119 .LP
120 For ELF executable programs (see \\fBa.out\\fR(4)), the core file generated is
121 also an ELF file, containing ELF program and file headers. The \\fBe_type\\fR

```

122 field in the file header has type `\fBET_CORE`. The program header contains an
 123 entry for every segment that was part of the process address space, including
 124 shared library segments. The contents of the mappings specified by
 125 `\fBcoreadm` are also part of the core image. Each program header has its
 126 `\fBp_memsz` field set to the size of the mapping. The program headers that
 127 represent mappings whose data is included in the core file have their
 128 `\fBp_filesz` field set the same as `\fBp_memsz`, otherwise `\fBp_filesz` is
 129 `\fBzero`.
 130 .sp
 131 .LP
 132 A mapping's data can be excluded due to the core file content settings (see
 133 `\fBcoreadm`), due to a failure, or due to a signal received after
 134 core dump initiation but before its completion. If the data is excluded
 135 because of a failure, the program header entry will have the
 136 `\fBPF_SUNW_FAILURE` flag
 137 set in its `\fBp_flags` field; if the data is excluded because of a signal,
 138 the segment's `\fBp_flags` field will have the `\fBPF_SUNW_KILLED`
 139 flag set.
 140 .sp
 141 .LP
 142 The program headers of an `\fBELF` core file also contain entries for two
 143 `\fBNOTE` segments, each containing several note entries as described below.
 144 The note entry header and core file note type (`\fBn_type`) definitions are
 145 contained in `<\fBsys/elf.h>`. The first `\fBNOTE` segment exists for binary
 146 compatibility with old programs that deal with core files. It contains
 147 structures defined in `<\fBsys/old_procfs.h>`. New programs should recognize
 148 and skip this `\fBNOTE` segment, advancing instead to the new `\fBNOTE`
 149 segment. The old `\fBNOTE` segment is deleted from core files in a future
 150 release.
 151 .sp
 152 .LP
 153 The old `\fBNOTE` segment contains the following entries. Each has entry name
 154 `\fB"CORE"` and presents the contents of a system structure:
 155 .sp
 156 .ne 2
 157 .na
 158 `\fB\fBprpsinfo_t`
 159 .ad
 160 .RS 16n
 161 `\fBn_type`: `\fBNT_PRPSINFO`. This entry contains information of interest to
 162 the `\fBps` command, such as process status, `\fBCPU` usage, `\fBnice`
 163 value, controlling terminal, user-ID, process-ID, the name of the executable,
 164 and so forth. The `\fBprpsinfo_t` structure is defined in
 165 `<\fBsys/old_procfs.h>`.
 166 .RE
 168 .sp
 169 .ne 2
 170 .na
 171 `\fB\fBbchar` array
 172 .ad
 173 .RS 16n
 174 `\fBn_type`: `\fBNT_PLATFORM`. This entry contains a string describing the
 175 specific model of the hardware platform on which this core file was created.
 176 This information is the same as provided by `\fBsysinfo` when invoked with
 177 the command `\fBBI_PLATFORM`.
 178 .RE
 180 .sp
 181 .ne 2
 182 .na
 183 `\fB\fBauxv_t` array
 184 .ad
 185 .RS 16n
 186 `\fBn_type`: `\fBNT_AUXV`. This entry contains the array of `\fBauxv_t`
 187 structures that was passed by the operating system as startup information to

188 the dynamic linker. Auxiliary vector information is defined in
 189 `<\fBsys/auxv.h>`.
 190 .RE
 192 .sp
 193 .LP
 194 Following these entries, for each active (non-zombie) light-weight process
 195 (LWP) in the process, the old `\fBNOTE` segment contains an entry with a
 196 `\fBprstatus_t` structure, plus other optionally-present entries describing
 197 the LWP, as follows:
 198 .sp
 199 .ne 2
 200 .na
 201 `\fB\fBprstatus_t`
 202 .ad
 203 .RS 16n
 204 `\fBn_type`: `\fBNT_PRSTATUS`. This structure contains things of interest to
 205 a debugger from the operating system, such as the general registers, signal
 206 dispositions, state, reason for stopping, process-ID, and so forth. The
 207 `\fBprstatus_t` structure is defined in `<\fBsys/old_procfs.h>`.
 208 .RE
 210 .sp
 211 .ne 2
 212 .na
 213 `\fB\fBprfpregset_t`
 214 .ad
 215 .RS 16n
 216 `\fBn_type`: `\fBNT_PRFPREG`. This entry is present only if the `\fBLWP`
 217 used the floating-point hardware. It contains the floating-point registers. The
 218 `\fBprfpregset_t` structure is defined in `<\fBsys/procfs_isa.h>`.
 219 .RE
 221 .sp
 222 .ne 2
 223 .na
 224 `\fB\fBgwindows_t`
 225 .ad
 226 .RS 16n
 227 `\fBn_type`: `\fBNT_GWINDOWS`. This entry is present only on a SPARC machine
 228 and only if the system was unable to flush all of the register windows to the
 229 stack. It contains all of the unspilled register windows. The `\fBgwindows_t`
 230 structure is defined in `<\fBsys/regset.h>`.
 231 .RE
 233 .sp
 234 .ne 2
 235 .na
 236 `\fB\fBprxregset_t`
 237 .ad
 238 .RS 16n
 239 `\fBn_type`: `\fBNT_PRXREG`. This entry is present only if the machine has
 240 extra register state associated with it. It contains the extra register state.
 241 The `\fBprxregset_t` structure is defined in `<\fBsys/procfs_isa.h>`.
 242 .RE
 244 .sp
 245 .LP
 246 The new `\fBNOTE` segment contains the following entries. Each has entry name
 247 `"\fBCORE"` and presents the contents of a system structure:
 248 .sp
 249 .ne 2
 250 .na
 251 `\fB\fBpsinfo_t`
 252 .ad
 253 .RS 20n

```

254 \fBn_type\fR: \fBNT_PSINFO\fR. This structure contains information of interest
255 to the \fBps\fR(1) command, such as process status, \fBCPU\fR usage, \fBnice\fR
256 value, controlling terminal, user-ID, process-ID, the name of the executable,
257 and so forth. The \fBpsinfo_t\fR structure is defined in <\fBsys/procfs.h\fR>.
258 .RE

260 .sp
261 .ne 2
262 .na
263 \fB\fBpstatus_t\fR\fR
264 .ad
265 .RS 20n
266 \fBn_type\fR: \fBNT_PSTATUS\fR. This structure contains things of interest to a
267 debugger from the operating system, such as pending signals, state, process-ID,
268 and so forth. The \fBpstatus_t\fR structure is defined in <\fBsys/procfs.h\fR>.
269 .RE

271 .sp
272 .ne 2
273 .na
274 \fB\fBbchar\fR array\fR
275 .ad
276 .RS 20n
277 \fBn_type\fR: \fBNT_PLATFORM\fR. This entry contains a string describing the
278 specific model of the hardware platform on which this core file was created.
279 This information is the same as provided by \fBsysinfo\fR(2) when invoked with
280 the command \fBFSI_PLATFORM\fR.
281 .RE

283 .sp
284 .ne 2
285 .na
286 \fB\fBbauxv_t\fR array\fR
287 .ad
288 .RS 20n
289 \fBn_type\fR: \fBNT_AUXV\fR. This entry contains the array of \fBbauxv_t\fR
290 structures that was passed by the operating system as startup information to
291 the dynamic linker. Auxiliary vector information is defined in
292 <\fBsys/auxv.h\fR>.
293 .RE

295 .sp
296 .ne 2
297 .na
298 \fB\fBstruct utsname\fR\fR
299 .ad
300 .RS 20n
301 \fBn_type\fR: \fBNT_UTSNAME\fR. This structure contains the system information
302 that would have been returned to the process if it had performed a
303 \fBuname\fR(2) system call prior to dumping core. The \fButsname\fR structure
304 is defined in <\fBsys/utsname.h\fR>.
305 .RE

307 .sp
308 .ne 2
309 .na
310 \fB\fBprcred_t\fR\fR
311 .ad
312 .RS 20n
313 \fBn_type\fR: \fBNT_PRCRED\fR. This structure contains the process credentials,
314 including the real, saved, and effective user and group IDs. The \fBprcred_t\fR
315 structure is defined in <\fBsys/procfs.h\fR>. Following the structure is an
316 optional array of supplementary group IDs. The total number of supplementary
317 group IDs is given by the \fBpr_ngroups\fR member of the \fBprcred_t\fR
318 structure, and the structure includes space for one supplementary group. If
319 \fBpr_ngroups\fR is greater than 1, there is \fBpr_ngroups - 1\fR \fBbgid_t\fR

```

```

320 items following the structure; otherwise, there is no additional data.
321 .RE

323 .sp
324 .ne 2
325 .na
326 \fB\fBbchar array\fR\fR
327 .ad
328 .RS 20n
329 \fBn_type\fR: \fBNT_ZONENAME\fR. This entry contains a string which describes
330 the name of the zone in which the process was running. See \fBzones\fR(5). The
331 information is the same as provided by \fBgetzonenamebyid\fR(3C) when invoked
332 with the numerical ID returned by \fBgetzoneid\fR(3C).
333 .RE

335 .sp
336 .ne 2
337 .na
338 \fB\fBprfdinfo_t\fR\fR
339 .ad
340 .RS 20n
341 \fBn_type\fR: \fBNT_FDINFO\fR. This structure contains information about
342 any open file descriptors, including the path, flags, and
343 \fBstat\fR(2) information. The \fBprfdinfo_t\fR structure is defined in
344 <\fBsys/procfs.h\fR>.
345 .RE

347 .sp
348 .ne 2
349 .na
350 \fB\fBstruct ssd\fR array\fR
351 .ad
352 .RS 20n
353 \fBn_type\fR: \fBNT_LDT\fR. This entry is present only on an 32-bit x86 machine
354 and only if the process has set up a Local Descriptor Table (LDT). It contains
355 an array of structures of type \fBstruct ssd\fR, each of which was typically
356 used to set up the \fB%gs\fR segment register to be used to fetch the address
357 of the current thread information structure in a multithreaded process. The
358 \fBssd\fR structure is defined in <\fBsys/sysi86.h\fR>.
359 .RE

361 .sp
362 .ne 2
363 .na
364 \fB\fBcore_content_t\fR\fR
365 .ad
366 .RS 20n
367 \fBn_type\fR: \fBNT_CONTENT\fR. This optional entry indicates which parts of
368 the process image are specified to be included in the core file. See
369 \fBcoreadm\fR(1M).
370 .RE

372 .sp
373 .LP
374 Following these entries, for each active and zombie \fBBLWP\fR in the process,
375 the new \fBNOTE\fR segment contains an entry with an \fBBlwpsinfo_t\fR structure
376 plus, for a non-zombie LWP, an entry with an \fBBlwpstatus_t\fR structure, plus
377 other optionally-present entries describing the LWP, as follows. A zombie LWP
378 is a non-detached LWP that has terminated but has not yet been reaped by
379 another LWP in the same process.
380 .sp
381 .ne 2
382 .na
383 \fB\fBlwpsinfo_t\fR\fR
384 .ad
385 .RS 15n

```

```

386 \fBn_type\fR: \fBNT_LWPSINFO\fR. This structure contains information of
387 interest to the \fBps\fR(1) command, such as \fBLWP\fR status, \fBCPU\fR usage,
388 \fBnice\fR value, \fBLWP-ID\fR, and so forth. The \fBlwpsinfo_t\fR structure is
389 defined in <\fBsys/procfs.h\fR>. This is the only entry present for a zombie
390 LWP.
391 .RE

393 .sp
394 .ne 2
395 .na
396 \fB\fBlwpstatus_t\fR\fR
397 .ad
398 .RS 15n
399 \fBn_type\fR: \fBNT_LWPSTATUS\fR. This structure contains things of interest to
400 a debugger from the operating system, such as the general registers, the
401 floating point registers, state, reason for stopping, \fBLWP-ID\fR, and so
402 forth. The \fBlwpstatus_t\fR structure is defined in <\fBsys/procfs.h\fR>.
403 .RE

405 .sp
406 .ne 2
407 .na
408 \fB\fBgwindows_t\fR\fR
409 .ad
410 .RS 15n
411 \fBn_type\fR: \fBNT_GWINDOWS\fR. This entry is present only on a SPARC machine
412 and only if the system was unable to flush all of the register windows to the
413 stack. It contains all of the unspilled register windows. The \fBgwindows_t\fR
414 structure is defined in \fB<sys/regset.h>\fR&.
415 .RE

417 .sp
418 .ne 2
419 .na
420 \fB\fBprxregset_t\fR\fR
421 .ad
422 .RS 15n
423 \fBn_type\fR: \fBNT_PRXREG\fR. This entry is present only if the machine has
424 extra register state associated with it. It contains the extra register state.
425 The \fBprxregset_t\fR structure is defined in \fB<sys/procfs_isa.h>\fR&.
426 .RE

428 .sp
429 .ne 2
430 .na
431 \fB\fBasrset_t\fR\fR
432 .ad
433 .RS 15n
434 \fBn_type\fR: \fBNT_ASRS\fR. This entry is present only on a SPARC V9 machine
435 and only if the process is a 64-bit process. It contains the ancillary state
436 registers for the \fBLWP\fR. The \fBasrset_t\fR structure is defined in
437 \fB<sys/regset.h>\fR&.
438 .RE

440 .sp
441 .ne 2
442 .na
443 \fB\fBpsinfo_t\fR\fR
444 .ad
445 .RS 15n
446 \fBn_type\fR: \fBNT_SPYMASTER\fR. This entry is present only for an agent
447 LWP and contains the \fBpsinfo_t\fR of the process that created the agent
448 LWP. See the \fBproc\fR(4) description of the \fBspymaster\fR entry for
449 more details.
450 .RE

```

```

452 .sp
453 .ne 2
454 .na
455 \fB\fBprsecflags_t\fR\fR
456 .ad
457 .RS 15n
458 \fBn_type\fR: \fBNT_SECFLAGS\fR. This entry contains the process
459 security-flags, see \fBsecurity-flags\fR(5), \fBproc\fR(4), and
460 \fBpsecflags\fR(1M) for more information.
461 .RE

463 .sp
464 #endif /* ! codereview */
465 .LP
466 Depending on the \fBcoreadm\fR(1M) settings, the section header of an ELF core
467 file can contain entries for CTF, symbol table, and string table sections. The
468 \fBsh_addr\fR fields are set to the base address of the first mapping of the
469 load object that they came from to. This can be used to match those sections
470 with the corresponding load object.
471 .sp
472 .LP
473 The size of the core file created by a process can be controlled by the user
474 (see \fBgetrlimit\fR(2)).
475 .SH SEE ALSO
476 .sp
477 \fBelfdump\fR(1), \fBgcore\fR(1), \fBmdb\fR(1), \fBproc\fR(1), \fBps\fR(1),
478 \fBcoreadm\fR(1M), \fBgetrlimit\fR(2), \fBsetrlimit\fR(2), \fBsetuid\fR(2),
479 \fBsysinfo\fR(2), \fBuname\fR(2), \fBgetzonebyname\fR(3C),
480 \fBgetzoneid\fR(3C), \fBelf\fR(3ELF), \fBsignal.h\fR(3HEAD), \fBba.out\fR(4),
481 \fBproc\fR(4), \fBzones\fR(5), \fBsecurity-flags\fR(5)
482 .sp
483 .LP
484 \fBIANSI C Programmer's Guide\fR

```

```

*****
99687 Thu Jun 30 22:02:47 2016
new/usr/src/man/man4/proc.4
Code review comments from jeffpc
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
1 \" te
2.\" Copyright 1989 AT&T
3.\" Copyright (c) 2006, Sun Microsystems, Inc. All Rights Reserved.
4.\" Copyright (c) 2013, Joyent, Inc. All rights reserved.
5.\" The contents of this file are subject to the terms of the Common Development
6.\" You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE or http:
7.\" When distributing Covered Code, include this CDDL HEADER in each file and in
8.TH PROC 4 "Jun 6, 2016"
9.TH PROC 4 "Mar 31, 2013"
10 .SH NAME
11 .SH DESCRIPTION
12 .sp
13 .LP
14 \fB/proc\fR is a file system that provides access to the state of each process
15 and light-weight process (lwp) in the system. The name of each entry in the
16 \fB/proc\fR directory is a decimal number corresponding to a process-ID. These
17 entries are themselves subdirectories. Access to process state is provided by
18 additional files contained within each subdirectory; the hierarchy is described
19 more completely below. In this document, ``\fB/proc\fR file'' refers to a
20 non-directory file within the hierarchy rooted at \fB/proc\fR. The owner of
21 each \fB/proc\fR file and subdirectory is determined by the user-ID of the
22 process.
23 .sp
24 .LP
25 \fB/proc\fR can be mounted on any mount point, in addition to the standard
26 \fB/proc\fR mount point, and can be mounted several places at once. Such
27 additional mounts are allowed in order to facilitate the confinement of
28 processes to subtrees of the file system via \fBchroot\fR(1M) and yet allow
29 such processes access to commands like \fBps\fR(1).
30 .sp
31 .LP
32 Standard system calls are used to access \fB/proc\fR files: \fBopen\fR(2),
33 \fBclose\fR(2), \fBread\fR(2), and \fBwrite\fR(2) (including \fBreadv\fR(2),
34 \fBwritev\fR(2), \fBpread\fR(2), and \fBpwrite\fR(2)). Most files describe
35 process state and can only be opened for reading. \fBctl\fR and \fBlwpctl\fR
36 (control) files permit manipulation of process state and can only be opened for
37 writing. \fBas\fR (address space) files contain the image of the running
38 process and can be opened for both reading and writing. An open for writing
39 allows process control; a read-only open allows inspection but not control. In
40 this document, we refer to the process as open for reading or writing if any of
41 its associated \fB/proc\fR files is open for reading or writing.
42 .sp
43 .LP
44 In general, more than one process can open the same \fB/proc\fR file at the
45 same time. \fBexclusive\fR \fBopen\fR is an advisory mechanism provided to
46 allow controlling processes to avoid collisions with each other. A process can
47 obtain exclusive control of a target process, with respect to other cooperating
48 processes, if it successfully opens any \fB/proc\fR file in the target process
49 for writing (the \fBas\fR or \fBctl\fR files, or the \fBlwpctl\fR file of any
50 lwp) while specifying \fB_EXCL\fR in the \fBopen\fR(2). Such an open will fail
51 if the target process is already open for writing (that is, if an \fBas\fR,
52 \fBctl\fR, or \fBlwpctl\fR file is already open for writing). There can be any
53 number of concurrent read-only opens; \fB_EXCL\fR is ignored on opens for
54 reading. It is recommended that the first open for writing by a controlling
55 process use the \fB_EXCL\fR flag; multiple controlling processes usually
56 result in chaos.

```

```

56 .sp
57 .LP
58 If a process opens one of its own \fB/proc\fR files for writing, the open
59 succeeds regardless of \fB_EXCL\fR and regardless of whether some other
60 process has the process open for writing. Self-opens do not count when another
61 process attempts an exclusive open. (A process cannot exclude a debugger by
62 opening itself for writing and the application of a debugger cannot prevent a
63 process from opening itself.) All self-opens for writing are forced to be
64 close-on-exec (see the \fB_SETFD\fR operation of \fBfcntl\fR(2)).
65 .sp
66 .LP
67 Data may be transferred from or to any locations in the address space of the
68 traced process by applying \fBlseek\fR(2) to position the \fBas\fR file at the
69 virtual address of interest followed by \fBread\fR(2) or \fBwrite\fR(2) (or by
70 using \fBpread\fR(2) or \fBpwrite\fR(2) for the combined operation). The
71 address-map files \fB/proc/\fR\fIpid\fR/\fBmap\fR and
72 \fB/proc/\fR\fIpid\fR/\fBxmap\fR can be read to determine the accessible areas
73 (mappings) of the address space. \fBIO\fR transfers may span contiguous
74 mappings. An \fBIO\fR request extending into an unmapped area is truncated at
75 the boundary. A write request beginning at an unmapped virtual address fails
76 with \fBEBIO\fR; a read request beginning at an unmapped virtual address returns
77 zero (an end-of-file indication).
78 .sp
79 .LP
80 Information and control operations are provided through additional files.
81 \fBprocfs.h\fR contains definitions of data structures and message formats
82 used with these files. Some of these definitions involve the use of sets of
83 flags. The set types \fBsigset_t\fR, \fBftset_t\fR, and \fBsysset_t\fR
84 correspond, respectively, to signal, fault, and system call enumerations
85 defined in \fB<sys/signal.h>\fR, \fB<sys/fault.h>\fR, and
86 \fB<sys/syscall.h>\fR. Each set type is large enough to hold flags for its
87 own enumeration. Although they are of different sizes, they have a common
88 structure and can be manipulated by these macros:
89 .sp
90 .in +2
91 .nf
92 prfillset(&set); /* turn on all flags in set */
93 premtypset(&set); /* turn off all flags in set */
94 praddset(&set, flag); /* turn on the specified flag */
95 prdelset(&set, flag); /* turn off the specified flag */
96 r = prismember(&set, flag); /* != 0 iff flag is turned on */
97 .fi
98 .in -2
100 .sp
101 .LP
102 One of \fBprfillset()\fR or \fBpremtypset()\fR must be used to initialize
103 \fBset\fR before it is used in any other operation. \fBflag\fR must be a member
104 of the enumeration corresponding to \fBset\fR.
105 .sp
106 .LP
107 Every process contains at least one \fBlight-weight process\fR, or \fBilwp\fR.
108 Each lwp represents a flow of execution that is independently scheduled by the
109 operating system. All lwps in a process share its address space as well as many
110 other attributes. Through the use of \fBlwpctl\fR and \fBctl\fR files as
111 described below, it is possible to affect individual lwps in a process or to
112 affect all of them at once, depending on the operation.
113 .sp
114 .LP
115 When the process has more than one lwp, a representative lwp is chosen by the
116 system for certain process status files and control operations. The
117 representative lwp is a stopped lwp only if all of the process's lwps are
118 stopped; is stopped on an event of interest only if all of the lwps are so
119 stopped (excluding \fBPR_SUSPENDED\fR lwps); is in a \fBPR_REQUESTED\fR stop
120 only if there are no other events of interest to be found; or, failing
121 everything else, is in a \fBPR_SUSPENDED\fR stop (implying that the process is

```

122 deadlocked). See the description of the `\fBstatus\fR` file for definitions of
 123 stopped states. See the `\fBPCSTOP\fR` control operation for the definition of
 124 'event of interest'.
 125 .sp
 126 .LP
 127 The representative lwp remains fixed (it will be chosen again on the next
 128 operation) as long as all of the lwps are stopped on events of interest or are
 129 in a `\fBPR_SUSPENDED\fR` stop and the `\fBPCRUN\fR` control operation is not
 130 applied to any of them.
 131 .sp
 132 .LP
 133 When applied to the process control file, every `\fB/proc\fR` control operation
 134 that must act on an lwp uses the same algorithm to choose which lwp to act
 135 upon. Together with synchronous stopping (see `\fBPCSET\fR`), this enables a
 136 debugger to control a multiple-lwp process using only the process-level status
 137 and control files if it so chooses. More fine-grained control can be achieved
 138 using the lwp-specific files.
 139 .sp
 140 .LP
 141 The system supports two process data models, the traditional 32-bit data model
 142 in which ints, longs and pointers are all 32 bits wide (the ILP32 data model),
 143 and on some platforms the 64-bit data model in which longs and pointers, but
 144 not ints, are 64 bits in width (the LP64 data model). In the LP64 data model
 145 some system data types, notably `\fBsize_t\fR`, `\fBoff_t\fR`, `\fBtime_t\fR` and
 146 `\fBdev_t\fR`, grow from 32 bits to 64 bits as well.
 147 .sp
 148 .LP
 149 The `\fB/proc\fR` interfaces described here are available to both 32-bit and
 150 64-bit controlling processes. However, many operations attempted by a 32-bit
 151 controlling process on a 64-bit target process will fail with `\fBEOVERFLOW\fR`
 152 because the address space range of a 32-bit process cannot encompass a 64-bit
 153 process or because the data in some 64-bit system data type cannot be
 154 compressed to fit into the corresponding 32-bit type without loss of
 155 information. Operations that fail in this circumstance include reading and
 156 writing the address space, reading the address-map files, and setting the
 157 target process's registers. There is no restriction on operations applied by a
 158 64-bit process to either a 32-bit or a 64-bit target processes.
 159 .sp
 160 .LP
 161 The format of the contents of any `\fB/proc\fR` file depends on the data model of
 162 the observer (the controlling process), not on the data model of the target
 163 process. A 64-bit debugger does not have to translate the information it reads
 164 from a `\fB/proc\fR` file for a 32-bit process from 32-bit format to 64-bit
 165 format. However, it usually has to be aware of the data model of the target
 166 process. The `\fBpr_dmodel\fR` field of the `\fBstatus\fR` files indicates the
 167 target process's data model.
 168 .sp
 169 .LP
 170 To help deal with system data structures that are read from 32-bit processes, a
 171 64-bit controlling program can be compiled with the C preprocessor symbol
 172 `\fB_SYSCALL32\fR` defined before system header files are included. This makes
 173 explicit 32-bit fixed-width data structures (like `\fBcstruct stat32\fR`) visible
 174 to the 64-bit program. See `\fBtypes32.h\fR(3HEAD)`.
 175 .SH DIRECTORY STRUCTURE
 176 .sp
 176 .LP
 177 At the top level, the directory `\fB/proc\fR` contains entries each of which
 178 names an existing process in the system. These entries are themselves
 179 directories. Except where otherwise noted, the files described below can be
 180 opened for reading only. In addition, if a process becomes a `\fBzombie\fR` (one
 181 that has exited but whose parent has not yet performed a `\fBwait\fR(3C)` upon
 182 it), most of its associated `\fB/proc\fR` files disappear from the hierarchy;
 183 subsequent attempts to open them, or to read or write files opened before the
 184 process exited, will elicit the error `\fBENOENT\fR`.
 185 .sp
 186 .LP

187 Although process state and consequently the contents of `\fB/proc\fR` files can
 188 change from instant to instant, a single `\fBread\fR(2)` of a `\fB/proc\fR` file is
 189 guaranteed to return a sane representation of state; that is, the read will be
 190 atomic with respect to the state of the process. No such guarantee applies to
 191 successive reads applied to a `\fB/proc\fR` file for a running process. In
 192 addition, atomicity is not guaranteed for `\fBI/O\fR` applied to the `\fBas\fR`
 193 (address-space) file for a running process or for a process whose address space
 194 contains memory shared by another running process.
 195 .sp
 196 .LP
 197 A number of structure definitions are used to describe the files. These
 198 structures may grow by the addition of elements at the end in future releases
 199 of the system and it is not legitimate for a program to assume that they will
 200 not.
 201 .SH STRUCTURE OF `\fB/proc\fR`
 202 .sp
 202 .LP
 203 A given directory `\fB/proc/\fR` contains the following entries. A
 204 process can use the invisible alias `\fB/proc/self\fR` if it wishes to open one
 205 of its own `\fB/proc\fR` files (invisible in the sense that the name 'self'
 206 does not appear in a directory listing of `\fB/proc\fR` obtained from
 207 `\fBls\fR(1)`, `\fBgetdents\fR(2)`, or `\fBreaddir\fR(3C)`).
 208 .SS "contracts"
 209 .sp
 210 .LP
 210 A directory containing references to the contracts held by the process. Each
 211 entry is a symlink to the contract's directory under `\fB/system/contract\fR`.
 212 See `\fBcontract\fR(4)`.
 213 .SS "as"
 214 .sp
 214 .LP
 215 Contains the address-space image of the process; it can be opened for both
 216 reading and writing. `\fBlseek\fR(2)` is used to position the file at the virtual
 217 address of interest and then the address space can be examined or changed
 218 through `\fBread\fR(2)` or `\fBwrite\fR(2)` (or by using `\fBpread\fR(2)` or
 219 `\fBpwrite\fR(2)` for the combined operation).
 220 .SS "ctl"
 221 .sp
 221 .LP
 222 A write-only file to which structured messages are written directing the system
 223 to change some aspect of the process's state or control its behavior in some
 224 way. The seek offset is not relevant when writing to this file. Individual lwps
 225 also have associated `\fBlwpcctl\fR` files in the lwp subdirectories. A control
 226 message may be written either to the process's `\fBctl\fR` file or to a specific
 227 `\fBlwpcctl\fR` file with operation-specific effects. The effect of a control
 228 message is immediately reflected in the state of the process visible through
 229 appropriate status and information files. The types of control messages are
 230 described in detail later. See `\fBCONTROL MESSAGES\fR`.
 231 .SS "status"
 232 .sp
 232 .LP
 233 Contains state information about the process and the representative lwp. The
 234 file contains a `\fBpstatus\fR` structure which contains an embedded
 235 `\fBlwpstatus\fR` structure for the representative lwp, as follows:
 236 .sp
 237 .in +2
 238 .nf
 239 typedef struct pstatus {
 240 int pr_flags; /* flags (see below) */
 241 int pr_nlwp; /* number of active lwps in the process */
 242 int pr_nzomb; /* number of zombie lwps in the process */
 243 pid_t pr_pid; /* process id */
 244 pid_t pr_ppid; /* parent process id */
 245 pid_t pr_pgid; /* process group id */
 246 pid_t pr_sid; /* session id */
 247 id_t pr_aslwpid; /* obsolete */

```

248     id_t pr_agentid;      /* lwp-id of the agent lwp, if any */
249     sigset_t pr_sigpend; /* set of process pending signals */
250     uintptr_t pr_brkbase; /* virtual address of the process heap */
251     size_t pr_brksize;    /* size of the process heap, in bytes */
252     uintptr_t pr_stkbase; /* virtual address of the process stack */
253     size_t pr_stksize;    /* size of the process stack, in bytes */
254     timestruc_t pr_utime; /* process user cpu time */
255     timestruc_t pr_stime; /* process system cpu time */
256     timestruc_t pr_cutime; /* sum of children's user times */
257     timestruc_t pr_cstime; /* sum of children's system times */
258     sigset_t pr_sigtrace; /* set of traced signals */
259     fltset_t pr_fltrace;  /* set of traced faults */
260     sysset_t pr_sysentry; /* set of system calls traced on entry */
261     sysset_t pr_sysexit; /* set of system calls traced on exit */
262     char pr_dmodel;      /* data model of the process */
263     taskid_t pr_taskid; /* task id */
264     projid_t pr_projid; /* project id */
265     zoneid_t pr_zoneid; /* zone id */
266     lwpstatus_t pr_lwp; /* status of the representative lwp */
267 } pstatus_t;

```

unchanged portion omitted

```

493 .fi
494 .in -2

496 .sp
497 .LP
498 \fBpr_flags\fR is a bit-mask holding the following lwp flags. For convenience,
499 it also contains the process flags, described previously.
500 .sp
501 .ne 2
502 .na
503 \fB\fbpr_stopped\fR
504 .ad
505 .RS 14n
506 The lwp is stopped.
507 .RE

509 .sp
510 .ne 2
511 .na
512 \fB\fbpr_istop\fR
513 .ad
514 .RS 14n
515 The lwp is stopped on an event of interest (see \fBpcstop\fR).
516 .RE

518 .sp
519 .ne 2
520 .na
521 \fB\fbpr_dstop\fR
522 .ad
523 .RS 14n
524 The lwp has a stop directive in effect (see \fBpcstop\fR).
525 .RE

527 .sp
528 .ne 2
529 .na
530 \fB\fbpr_step\fR
531 .ad
532 .RS 14n
533 The lwp has a single-step directive in effect (see \fBpcrun\fR).
534 .RE

536 .sp
537 .ne 2

```

```

538 .na
539 \fB\fbpr_asleep\fR
540 .ad
541 .RS 14n
542 The lwp is in an interruptible sleep within a system call.
543 .RE

545 .sp
546 .ne 2
547 .na
548 \fB\fbpr_pcINVAL\fR
549 .ad
550 .RS 14n
551 The lwp's current instruction (\fBpr_instr\fR) is undefined.
552 .RE

554 .sp
555 .ne 2
556 .na
557 \fB\fbpr_detach\fR
558 .ad
559 .RS 14n
560 This is a detached lwp (see \fBpthread_create\fR(3C) and
561 \fBpthread_join\fR(3C)).
562 .RE

564 .sp
565 .ne 2
566 .na
567 \fB\fbpr_daemon\fR
568 .ad
569 .RS 14n
570 This is a daemon lwp (see \fBpthread_create\fR(3C)).
571 .RE

573 .sp
574 .ne 2
575 .na
576 \fB\fbpr_aslwp\fR
577 .ad
578 .RS 14n
579 This flag is obsolete and is never set.
580 .RE

582 .sp
583 .ne 2
584 .na
585 \fB\fbpr_agent\fR
586 .ad
587 .RS 14n
588 This is the \fB/proc\fR agent lwp for the process.
589 .RE

591 .sp
592 .LP
593 \fBpr_lwpid\fR names the specific lwp.
594 .sp
595 .LP
596 \fBpr_why\fR and \fBpr_what\fR together describe, for a stopped lwp, the reason
597 for the stop. Possible values of \fBpr_why\fR and the associated \fBpr_what\fR
598 are:
599 .sp
600 .ne 2
601 .na
602 \fB\fbpr_requested\fR
603 .ad

```



```

604 .RS 17n
605 indicates that the stop occurred in response to a stop directive, normally
606 because \fBPCSTOP\fR was applied or because another lwp stopped on an event of
607 interest and the asynchronous-stop flag (see \fBPCSET\fR) was not set for the
608 process. \fBpr_what\fR is unused in this case.
609 .RE

611 .sp
612 .ne 2
613 .na
614 \fB\fBPR_SIGNALED\fR\fR
615 .ad
616 .RS 17n
617 indicates that the lwp stopped on receipt of a signal (see \fBPCSTRACE\fR);
618 \fBpr_what\fR holds the signal number that caused the stop (for a newly-stopped
619 lwp, the same value is in \fBpr_cursig\fR).
620 .RE

622 .sp
623 .ne 2
624 .na
625 \fB\fBPR_FAULTED\fR\fR
626 .ad
627 .RS 17n
628 indicates that the lwp stopped on incurring a hardware fault (see
629 \fBPCSFault\fR); \fBpr_what\fR holds the fault number that caused the stop.
630 .RE

632 .sp
633 .ne 2
634 .na
635 \fB\fBPR_SYSENTRY\fR\fR
636 .ad
637 .br
638 .na
639 \fB\fBPR_SYSEXIT\fR\fR
640 .ad
641 .RS 17n
642 indicate a stop on entry to or exit from a system call (see \fBPCSENTRY\fR and
643 \fBPCSEXIT\fR); \fBpr_what\fR holds the system call number.
644 .RE

646 .sp
647 .ne 2
648 .na
649 \fB\fBPR_JOBCONTROL\fR\fR
650 .ad
651 .RS 17n
652 indicates that the lwp stopped due to the default action of a job control stop
653 signal (see \fBsigaction\fR(2)); \fBpr_what\fR holds the stopping signal
654 number.
655 .RE

657 .sp
658 .ne 2
659 .na
660 \fB\fBPR_SUSPENDED\fR\fR
661 .ad
662 .RS 17n
663 indicates that the lwp stopped due to internal synchronization of lwps within
664 the process. \fBpr_what\fR is unused in this case.
665 .RE

667 .sp
668 .LP
669 \fBpr_cursig\fR names the current signal, that is, the next signal to be

```

```

670 delivered to the lwp, if any. \fBpr_info\fR, when the lwp is in a
671 \fBPR_SIGNALED\fR or \fBPR_FAULTED\fR stop, contains additional information
672 pertinent to the particular signal or fault (see \fB<sys/signinfo.h>\fR).
673 .sp
674 .LP
675 \fBpr_lwppend\fR identifies any synchronous or directed signals pending for the
676 lwp. \fBpr_lwphold\fR identifies those signals whose delivery is being blocked
677 by the lwp (the signal mask).
678 .sp
679 .LP
680 \fBpr_action\fR contains the signal action information pertaining to the
681 current signal (see \fBsigaction\fR(2)); it is undefined if \fBpr_cursig\fR is
682 zero. \fBpr_altstack\fR contains the alternate signal stack information for the
683 lwp (see \fBsigaltstack\fR(2)).
684 .sp
685 .LP
686 \fBpr_oldcontext\fR, if not zero, contains the address on the lwp stack of a
687 \fBpr_ucontext\fR structure describing the previous user-level context (see
688 \fBpr_ucontext.h\fR(3HEAD)). It is non-zero only if the lwp is executing in the
689 context of a signal handler.
690 .sp
691 .LP
692 \fBpr_syscall\fR is the number of the system call, if any, being executed by
693 the lwp; it is non-zero if and only if the lwp is stopped on \fBPR_SYSENTRY\fR
694 or \fBPR_SYSEXIT\fR, or is asleep within a system call ( \fBPR_ASLEEP\fR is
695 set). If \fBpr_syscall\fR is non-zero, \fBpr_nsysarg\fR is the number of
696 arguments to the system call and \fBpr_sysarg\fR contains the actual arguments.
697 .sp
698 .LP
699 \fBpr_rvall\fR, \fBpr_rval2\fR, and \fBpr_errno\fR are defined only if the lwp
700 is stopped on \fBPR_SYSEXIT\fR or if the \fBPR_VFORKP\fR flag is set. If
701 \fBpr_errno\fR is zero, \fBpr_rvall\fR and \fBpr_rval2\fR contain the return
702 values from the system call. Otherwise, \fBpr_errno\fR contains the error
703 number for the failing system call (see \fB<sys/errno.h>\fR).
704 .sp
705 .LP
706 \fBpr_clname\fR contains the name of the lwp's scheduling class.
707 .sp
708 .LP
709 \fBpr_tstamp\fR, if the lwp is stopped, contains a time stamp marking when the
710 lwp stopped, in real time seconds and nanoseconds since an arbitrary time in
711 the past.
712 .sp
713 .LP
714 \fBpr_utime\fR is the amount of user level CPU time used by this LWP.
715 .sp
716 .LP
717 \fBpr_stime\fR is the amount of system level CPU time used by this LWP.
718 .sp
719 .LP
720 \fBpr_ustack\fR is the virtual address of the \fBpr_stack_t\fR that contains the
721 stack boundaries for this LWP. See \fBpr_getustack\fR(2) and
722 \fBpr_stack_grow\fR(3C).
723 .sp
724 .LP
725 \fBpr_instr\fR contains the machine instruction to which the lwp's program
726 counter refers. The amount of data retrieved from the process is
727 machine-dependent. On SPARC based machines, it is a 32-bit word. On x86-based
728 machines, it is a single byte. In general, the size is that of the machine's
729 smallest instruction. If \fBPR_PCINVAL\fR is set, \fBpr_instr\fR is undefined;
730 this occurs whenever the lwp is not stopped or when the program counter refers
731 to an invalid virtual address.
732 .sp
733 .LP
734 \fBpr_reg\fR is an array holding the contents of a stopped lwp's general
735 registers.

```

```

736 .sp
737 .ne 2
738 .na
739 \fBSPARC\fR
740 .ad
741 .RS 2ln
742 On SPARC-based machines, the predefined constants \fBR_G0\fR ... \fBR_G7\fR,
743 \fBR_O0\fR ... \fBR_O7\fR, \fBR_L0\fR ... \fBR_L7\fR, \fBR_I0\fR ...
744 \fBR_I7\fR, \fBR_PC\fR, \fBR_nPC\fR, and \fBR_Y\fR can be used as indices to
745 refer to the corresponding registers; previous register windows can be read
746 from their overflow locations on the stack (however, see the \fBgwindows\fR
747 file in the \fB/proc/\fR\fIpid\fR/\fB/lwp/\fR\fIlwpid\fR subdirectory).
748 .RE

750 .sp
751 .ne 2
752 .na
753 \fBSPARC V8 (32-bit)\fR
754 .ad
755 .RS 2ln
756 For SPARC V8 (32-bit) controlling processes, the predefined constants
757 \fBR_PSR\fR, \fBR_WIM\fR, and \fBR_TBR\fR can be used as indices to refer to
758 the corresponding special registers. For SPARC V9 (64-bit) controlling
759 processes, the predefined constants \fBR_CCR\fR, \fBR_ASI\fR, and \fBR_FPRS\fR
760 can be used as indices to refer to the corresponding special registers.
761 .RE

763 .sp
764 .ne 2
765 .na
766 \fBx86 (32-bit)\fR
767 .ad
768 .RS 2ln
769 For 32-bit x86 processes, the predefined constants listed below can be used as
770 indices to refer to the corresponding registers.
771 .sp
772 .in +2
773 .nf
774 SS
775 UESP
776 EFL
777 CS
778 EIP
779 ERR
780 TRAPNO
781 EAX
782 ECX
783 EDX
784 EBX
785 ESP
786 EBP
787 ESI
788 EDI
789 DS
790 ES
791 GS
792 .fi
793 .in -2

795 The preceding constants are listed in \fB<sys/regset.h>\fR&.
796 .sp
797 Note that a 32-bit process can run on an x86 64-bit system, using the constants
798 listed above.
799 .RE

801 .sp

```

```

802 .ne 2
803 .na
804 \fBx86 (64-bit)\fR
805 .ad
806 .RS 2ln
807 To read the registers of a 32- \fBor\fR a 64-bit process, a 64-bit x86 process
808 should use the predefined constants listed below.
809 .sp
810 .in +2
811 .nf
812 REG_GSBASE
813 REG_FSBASE
814 REG_DS
815 REG_ES
816 REG_GS
817 REG_FS
818 REG_SS
819 REG_RSP
820 REG_RFL
821 REG_CS
822 REG_RIP
823 REG_ERR
824 REG_TRAPNO
825 REG_RAX
826 REG_RCX
827 REG_RDX
828 REG_RBX
829 REG_RBP
830 REG_RSI
831 REG_RDI
832 REG_R8
833 REG_R9
834 REG_R10
835 REG_R11
836 REG_R12
837 REG_R13
838 REG_R14
839 REG_R15
840 .fi
841 .in -2

843 The preceding constants are listed in \fB<sys/regset.h>\fR&.
844 .RE

846 .sp
847 .LP
848 \fBpr_fpreg\fR is a structure holding the contents of the floating-point
849 registers.
850 .sp
851 .LP
852 SPARC registers, both general and floating-point, as seen by a 64-bit
853 controlling process are the V9 versions of the registers, even if the target
854 process is a 32-bit (V8) process. V8 registers are a subset of the V9
855 registers.
856 .sp
857 .LP
858 If the lwp is not stopped, all register values are undefined.
859 .SS "psinfo"
860 .LP
861 Contains miscellaneous information about the process and the representative lwp
862 needed by the \fBps(1)\fR command. \fBpsinfo\fR remains accessible after a
863 process becomes a \fBzombie\fR. The file contains a \fBpsinfo\fR structure
864 which contains an embedded \fBlwpsinfo\fR structure for the representative lwp,
865 as follows:
866 .sp

```

```

867 .in +2
868 .nf
869 typedef struct psinfo {
870     int pr_flag; /* process flags (DEPRECATED: see below) */
871     int pr_nlwp; /* number of active lwps in the process */
872     int pr_nzomb; /* number of zombie lwps in the process */
873     pid_t pr_pid; /* process id */
874     pid_t pr_ppid; /* process id of parent */
875     pid_t pr_pgid; /* process id of process group leader */
876     pid_t pr_sid; /* session id */
877     uid_t pr_uid; /* real user id */
878     uid_t pr_euid; /* effective user id */
879     gid_t pr_gid; /* real group id */
880     gid_t pr_egid; /* effective group id */
881     uintptr_t pr_addr; /* address of process */
882     size_t pr_size; /* size of process image in Kbytes */
883     size_t pr_rssize; /* resident set size in Kbytes */
884     dev_t pr_ttydev; /* controlling tty device (or PRNODEV) */
885     ushort_t pr_pctcpu; /* % of recent cpu time used by all lwps */
886     ushort_t pr_pctmem; /* % of system memory used by process */
887     timestruc_t pr_start; /* process start time, from the epoch */
888     timestruc_t pr_time; /* cpu time for this process */
889     timestruc_t pr_ctime; /* cpu time for reaped children */
890     char pr_fname[PRFNSZ]; /* name of exec'ed file */
891     char pr_psargs[PRARGSZ]; /* initial characters of arg list */
892     int pr_wstat; /* if zombie, the wait() status */
893     int pr_argc; /* initial argument count */
894     uintptr_t pr_argv; /* address of initial argument vector */
895     uintptr_t pr_envp; /* address of initial environment vector */
896     char pr_dmodel; /* data model of the process */
897     lwpsinfo_t pr_lwp; /* information for representative lwp */
898     taskid_t pr_taskid; /* task id */
899     projid_t pr_projid; /* project id */
900     poolid_t pr_poolid; /* pool id */
901     zoneid_t pr_zoneid; /* zone id */
902     ctid_t pr_contract; /* process contract id */
903 } psinfo_t;

```

unchanged portion omitted

```

955 .fi
956 .in -2

958 .sp
959 .LP
960 Some of the entries in \fBlwpsinfo\fR, such as \fBpr_addr\fR, \fBpr_wchan\fR,
961 \fBpr_stype\fR, \fBpr_state\fR, and \fBpr_name\fR, refer to internal kernel
962 data structures and should not be expected to retain their meanings across
963 different versions of the operating system.
964 .sp
965 .LP
966 \fBlwpsinfo_t.pr_flag\fR is a deprecated interface that should no longer be
967 used.
968 .sp
969 .LP
970 \fBpr_pctcpu\fR is a 16-bit binary fraction, as described above. It represents
971 the \fBpr_pctcpu\fR time used by the specific lwp. On a multi-processor machine, the
972 maximum value is 1/N, where N is the number of \fBpr_cpu\fRs.
973 .sp
974 .LP
975 \fBpr_contract\fR is the id of the process contract of which the process is a
976 member. See \fBpr_contract\fR(4) and \fBpr_process\fR(4).
977 .SS "cred"
978 .sp
979 Contains a description of the credentials associated with the process:
980 .sp
981 .in +2

```

```

982 .nf
983 typedef struct prcred {
984     uid_t pr_euid; /* effective user id */
985     uid_t pr_ruid; /* real user id */
986     uid_t pr_suid; /* saved user id (from exec) */
987     gid_t pr_egid; /* effective group id */
988     gid_t pr_rgid; /* real group id */
989     gid_t pr_sgid; /* saved group id (from exec) */
990     int pr_ngroups; /* number of supplementary groups */
991     gid_t pr_groups[1]; /* array of supplementary groups */
992 } prcred_t;
993 .fi
994 .in -2
995 .sp

997 .sp
998 .LP
999 The array of associated supplementary groups in \fBpr_groups\fR is of variable
1000 length; the \fBpr_cred\fR file contains all of the supplementary groups.
1001 \fBpr_ngroups\fR indicates the number of supplementary groups. (See also the
1002 \fBpr_cred\fR and \fBpr_credx\fR control operations.)
1003 .SS "priv"
1004 .sp
1005 Contains a description of the privileges associated with the process:
1006 .sp
1007 .in +2
1008 .nf
1009 typedef struct prpriv {
1010     uint32_t pr_nsets; /* number of privilege set */
1011     uint32_t pr_setsize; /* size of privilege set */
1012     uint32_t pr_infosize; /* size of supplementary data */
1013     priv_chunk_t pr_sets[1]; /* array of sets */
1014 } prpriv_t;
1015 .fi
1016 .in -2

1018 .sp
1019 .LP
1020 The actual dimension of the \fBpr_sets\fR[] field is
1021 .sp
1022 .in +2
1023 .nf
1024 pr_sets[pr_nsets][pr_setsize]
1025 .fi
1026 .in -2

1028 .sp
1029 .LP
1030 which is followed by additional information about the process state
1031 \fBpr_infosize\fR bytes in size.
1032 .sp
1033 .LP
1034 The full size of the structure can be computed using
1035 \fBPRIV_PRIV_SIZE\fR(\fBprpriv_t *\fR).
1036 .SS "secflags"
1037 .LP
1038 This file contains the security-flags of the process. It contains a
1039 description of the security flags associated with the process.
1040 .sp
1041 .in +2
1042 .nf
1043 typedef struct prsecflags {
1044     uint32_t pr_version; /* ABI Versioning of this structure */
1045     secflagset_t pr_effective; /* Effective flags */
1046     secflagset_t pr_inherit; /* Inheritable flags */

```

```

1047     secflagset_t pr_lower;      /* Lower flags */
1048     secflagset_t pr_upper;      /* Upper flags */
1049 } prsecflags_t;
1050 .in -2

1052 .sp
1053 .LP
1054 The \fBpr_version\fR field is a version number for the structure, currently
1055 \fBPRSECFLAGS_VERSION_1\fR.
1056 #endif /* ! codereview */
1057 .SS "sigact"
1058 .sp
1059 Contains an array of \fBsigaction\fR structures describing the current
1060 dispositions of all signals associated with the traced process (see
1061 \fBsigaction\fR(2)). Signal numbers are displaced by 1 from array indices, so
1062 that the action for signal number \fIn\fR appears in position \fIn-1 of the
1063 array.
1064 .SS "auxv"
1065 .sp
1066 Contains the initial values of the process's aux vector in an array of
1067 \fBauxv_t\fR structures (see \fB<sys/auxv.h>\fR). The values are those that
1068 were passed by the operating system as startup information to the dynamic
1069 linker.
1070 .SS "ldt"
1071 .sp
1072 This file exists only on x86-based machines. It is non-empty only if the
1073 process has established a local descriptor table (\fBLDT\fR). If non-empty, the
1074 file contains the array of currently active \fBLDT\fR entries in an array of
1075 elements of type \fBstruct ssd\fR, defined in \fB<sys/sysi86.h>\fR, one element
1076 for each active \fBLDT\fR entry.
1077 .SS "map, xmap"
1078 .sp
1079 Contain information about the virtual address map of the process. The map file
1080 contains an array of \fBprmap\fR structures while the xmap file contains an
1081 array of \fBprxmap\fR structures. Each structure describes a contiguous virtual
1082 address region in the address space of the traced process:
1083 .sp
1084 .in +2
1085 .nf
1086 typedef struct prmap {
1087     uintptr_t pr_vaddr;      /* virtual address of mapping */
1088     size_t pr_size;         /* size of mapping in bytes */
1089     char pr_mapname[PRMAPSZ]; /* name in /proc/pid/object */
1090     off_t pr_offset;        /* offset into mapped object, if any */
1091     int pr_mflags;          /* protection and attribute flags */
1092     int pr_pagesize;        /* pagesize for this mapping in bytes */
1093     int pr_shmid;           /* SysV shared memory identifier */
1094 } prmap_t;
1095 unchanged portion omitted
1117 .fi
1118 .in -2
1119 .sp

1121 .sp
1122 .LP
1123 \fBpr_vaddr\fR is the virtual address of the mapping within the traced process
1124 and \fBpr_size\fR is its size in bytes. \fBpr_mapname\fR, if it does not
1125 contain a null string, contains the name of a file in the \fBobject\fR
1126 directory (see below) that can be opened read-only to obtain a file descriptor
1127 for the mapped file associated with the mapping. This enables a debugger to
1128 find object file symbol tables without having to know the real path names of
1129 the executable file and shared libraries of the process. \fBpr_offset\fR is the

```

```

1130 64-bit offset within the mapped file (if any) to which the virtual address is
1131 mapped.
1132 .sp
1133 .LP
1134 \fBpr_mflags\fR is a bit-mask of protection and attribute flags:
1135 .sp
1136 .ne 2
1137 .na
1138 \fBFBMA_READ\fR
1139 .ad
1140 .RS 17n
1141 mapping is readable by the traced process.
1142 .RE

1144 .sp
1145 .ne 2
1146 .na
1147 \fBFBMA_WRITE\fR
1148 .ad
1149 .RS 17n
1150 mapping is writable by the traced process.
1151 .RE

1153 .sp
1154 .ne 2
1155 .na
1156 \fBFBMA_EXEC\fR
1157 .ad
1158 .RS 17n
1159 mapping is executable by the traced process.
1160 .RE

1162 .sp
1163 .ne 2
1164 .na
1165 \fBFBMA_SHARED\fR
1166 .ad
1167 .RS 17n
1168 mapping changes are shared by the mapped object.
1169 .RE

1171 .sp
1172 .ne 2
1173 .na
1174 \fBFBMA_ISM\fR
1175 .ad
1176 .RS 17n
1177 mapping is intimate shared memory (shared MMU resources)
1178 .RE

1180 .sp
1181 .ne 2
1182 .na
1183 \fBFBMAP_NORESERVE\fR
1184 .ad
1185 .RS 17n
1186 mapping does not have swap space reserved (mapped with MAP_NORESERVE)
1187 .RE

1189 .sp
1190 .ne 2
1191 .na
1192 \fBFBMA_SHM\fR
1193 .ad
1194 .RS 17n
1195 mapping System V shared memory

```

1196 .RE

1198 .sp
 1199 .LP
 1200 A contiguous area of the address space having the same underlying mapped object
 1201 may appear as multiple mappings due to varying read, write, and execute
 1202 attributes. The underlying mapped object does not change over the range of a
 1203 single mapping. An `\fBI/O\fR` operation to a mapping marked `\fBMA_SHARED\fR`
 1204 fails if applied at a virtual address not corresponding to a valid page in the
 1205 underlying mapped object. A write to a `\fBMA_SHARED\fR` mapping that is not
 1206 marked `\fBMA_WRITE\fR` fails. Reads and writes to private mappings always
 1207 succeed. Reads and writes to unmapped addresses fail.

1208 .sp
 1209 .LP
 1210 `\fBpr_pagesize\fR` is the page size for the mapping, currently always the system
 1211 `pagesize`.

1212 .sp
 1213 .LP
 1214 `\fBpr_shmid\fR` is the shared memory identifier, if any, for the mapping. Its
 1215 value is `\fB\mi1\fR` if the mapping is not System V shared memory. See
 1216 `\fBshmget\fR(2)`.

1217 .sp
 1218 .LP
 1219 `\fBpr_dev\fR` is the device of the mapped object, if any, for the mapping. Its
 1220 value is `\fBPRNODEV\fR (-1)` if the mapping does not have a device.

1221 .sp
 1222 .LP
 1223 `\fBpr_ino\fR` is the inode of the mapped object, if any, for the mapping. Its
 1224 contents are only valid if `\fBpr_dev\fR` is not `\fBPRNODEV`.

1225 .sp
 1226 .LP
 1227 `\fBpr_rss\fR` is the number of resident pages of memory for the mapping. The
 1228 number of resident bytes for the mapping may be determined by multiplying
 1229 `\fBpr_rss\fR` by the page size given by `\fBpr_pagesize`.

1230 .sp
 1231 .LP
 1232 `\fBpr_anon\fR` is the number of resident anonymous memory pages (pages which are
 1233 private to this process) for the mapping.

1234 .sp
 1235 .LP
 1236 `\fBpr_locked\fR` is the number of locked pages for the mapping. Pages which are
 1237 locked are always resident in memory.

1238 .sp
 1239 .LP
 1240 `\fBpr_hatpagesize\fR` is the size, in bytes, of the `\fBHAT` (`\fBMMU`)
 1241 translation for the mapping. `\fBpr_hatpagesize` may be different than
 1242 `\fBpr_pagesize`. The possible values are hardware architecture specific, and
 1243 may change over a mapping's lifetime.

1244 .SS "rmap"
 1237 .sp
 1245 .LP
 1246 Contains information about the reserved address ranges of the process. The file
 1247 contains an array of `\fBprmap` structures, as defined above for the `\fBmap`
 1248 file. Each structure describes a contiguous virtual address region in the
 1249 address space of the traced process that is reserved by the system in the sense
 1250 that an `\fBmmap(2)` system call that does not specify `\fBMAP_FIXED` will
 1251 not use any part of it for the new mapping. Examples of such reservations
 1252 include the address ranges reserved for the process stack and the individual
 1253 thread stacks of a multi-threaded process.

1254 .SS "cwd"
 1248 .sp
 1255 .LP
 1256 A symbolic link to the process's current working directory. See `\fBchdir`(2).
 1257 A `\fBreadlink`(2) of `\fB/proc/\fIpid\fR/cwd` yields a null string. However,
 1258 it can be opened, listed, and searched as a directory, and can be the target of
 1259 `\fBchdir`(2).

1260 .SS "root"
 1255 .sp
 1261 .LP
 1262 A symbolic link to the process's root directory.
 1263 `\fB/proc/\fIpid\fR/\fB/root` can differ from the system root directory if
 1264 the process or one of its ancestors executed `\fBchroot`(2) as super user. It
 1265 has the same semantics as `\fB/proc/\fR/\fIpid\fR/\fB/cwd`.

1266 .SS "fd"
 1262 .sp
 1267 .LP
 1268 A directory containing references to the open files of the process. Each entry
 1269 is a decimal number corresponding to an open file descriptor in the process.

1270 .sp
 1271 .LP
 1272 If an entry refers to a regular file, it can be opened with normal file system
 1273 semantics but, to ensure that the controlling process cannot gain greater
 1274 access than the controlled process, with no file access modes other than its
 1275 read/write open modes in the controlled process. If an entry refers to a
 1276 directory, it can be accessed with the same semantics as
 1277 `\fB/proc/\fIpid\fR/cwd`. An attempt to open any other type of entry fails

1278 with `\fBEACCESS`.

1279 .SS "object"
 1276 .sp
 1280 .LP
 1281 A directory containing read-only files with names corresponding to the
 1282 `\fBpr_mapname` entries in the `\fBmap` and `\fBpagedata` files. Opening
 1283 such a file yields a file descriptor for the underlying mapped file associated
 1284 with an address-space mapping in the process. The file name `\fBa.out` appears
 1285 in the directory as an alias for the process's executable file.

1286 .sp
 1287 .LP
 1288 The `\fBobject` directory makes it possible for a controlling process to gain
 1289 access to the object file and any shared libraries (and consequently the symbol
 1290 tables) without having to know the actual path names of the executable files.

1291 .SS "path"
 1289 .sp
 1292 .LP
 1293 A directory containing symbolic links to files opened by the process. The
 1294 directory includes one entry for `\fBcwd` and `\fBroot`. The directory also
 1295 contains a numerical entry for each file descriptor in the `\fBfd` directory,
 1296 and entries matching those in the `\fBobject` directory. If this information
 1297 is not available, any attempt to read the contents of the symbolic link will
 1298 fail. This is most common for files that do not exist in the filesystem
 1299 namespace (such as `\fBFIFO`s and sockets), but can also happen for regular
 1300 files. For the file descriptor entries, the path may be different from the one
 1301 used by the process to open the file.

1302 .SS "pagedata"
 1301 .sp
 1303 .LP
 1304 Opening the page data file enables tracking of address space references and
 1305 modifications on a per-page basis.

1306 .sp
 1307 .LP
 1308 A `\fBread`(2) of the page data file descriptor returns structured page data
 1309 and atomically clears the page data maintained for the file by the system. That
 1310 is to say, each read returns data collected since the last read; the first read
 1311 returns data collected since the file was opened. When the call completes, the
 1312 read buffer contains the following structure as its header and thereafter
 1313 contains a number of section header structures and associated byte arrays that
 1314 must be accessed by walking linearly through the buffer.

1315 .sp
 1316 .in +2
 1317 .nf
 1318 typedef struct prpageheader {
 1319 timestruc_t pr_tstamp; /* real time stamp, time of read() */
 1320 ulong_t pr_nmap; /* number of address space mappings */

```

1321     ulong_t pr_npage;      /* total number of pages */
1322 } prpageheader_t;
unchanged portion omitted
1342 .fi
1343 .in -2

1345 .sp
1346 .LP
1347 Each section header is followed by \fBpr_npage\fR bytes, one byte for each page
1348 in the mapping, plus 0-7 null bytes at the end so that the next \fBprasmmap\fR
1349 structure begins on an eight-byte aligned boundary. Each data byte may contain
1350 these flags:
1351 .sp
1352 .ne 2
1353 .na
1354 \fB\FBPG_REFERENCED\fR
1355 .ad
1356 .RS 17n
1357 page has been referenced.
1358 .RE

1360 .sp
1361 .ne 2
1362 .na
1363 \fB\FBPG_MODIFIED\fR
1364 .ad
1365 .RS 17n
1366 page has been modified.
1367 .RE

1369 .sp
1370 .LP
1371 If the read buffer is not large enough to contain all of the page data, the
1372 read fails with \fBE2BIG\fR and the page data is not cleared. The required size
1373 of the read buffer can be determined through \fBfstat(2)\fR. Application of
1374 \fBlseek(2)\fR to the page data file descriptor is ineffective; every read
1375 starts from the beginning of the file. Closing the page data file descriptor
1376 terminates the system overhead associated with collecting the data.
1377 .sp
1378 .LP
1379 More than one page data file descriptor for the same process can be opened, up
1380 to a system-imposed limit per traced process. A read of one does not affect the
1381 data being collected by the system for the others. An open of the page data
1382 file will fail with \fBENOMEM\fR if the system-imposed limit would be exceeded.
1383 .SS "watch"
1384 .sp
1385 .LP
1385 Contains an array of \fBprwatch\fR structures, one for each watched area
1386 established by the \fBPCWATCH\fR control operation. See \fBPCWATCH\fR for
1387 details.
1388 .SS "usage"
1389 .sp
1390 .LP
1390 Contains process usage information described by a \fBprusage\fR structure which
1391 contains at least the following fields:
1392 .sp
1393 .in +2
1394 .nf
1395 typedef struct prusage {
1396     id_t pr_lwpid;          /* lwp id. 0: process or defunct */
1397     int pr_count;          /* number of contributing lwps */
1398     timestruc_t pr_tstamp; /* real time stamp, time of read() */
1399     timestruc_t pr_create; /* process/lwp creation time stamp */
1400     timestruc_t pr_term;   /* process/lwp termination time stamp */
1401     timestruc_t pr_rtime;  /* total lwp real (elapsed) time */
1402     timestruc_t pr_utime;  /* user level CPU time */

```

```

1403     timestruc_t pr_stime; /* system call CPU time */
1404     timestruc_t pr_ttime; /* other system trap CPU time */
1405     timestruc_t pr_tftime; /* text page fault sleep time */
1406     timestruc_t pr_dftime; /* data page fault sleep time */
1407     timestruc_t pr_kftime; /* kernel page fault sleep time */
1408     timestruc_t pr_ltime; /* user lock wait sleep time */
1409     timestruc_t pr_slptime; /* all other sleep time */
1410     timestruc_t pr_wtime; /* wait-cpu (latency) time */
1411     timestruc_t pr_stoptime; /* stopped time */
1412     ulong_t pr_minf; /* minor page faults */
1413     ulong_t pr_majf; /* major page faults */
1414     ulong_t pr_nswap; /* swaps */
1415     ulong_t pr_inblk; /* input blocks */
1416     ulong_t pr_oublk; /* output blocks */
1417     ulong_t pr_msnd; /* messages sent */
1418     ulong_t pr_mrcv; /* messages received */
1419     ulong_t pr_sigs; /* signals received */
1420     ulong_t pr_vctx; /* voluntary context switches */
1421     ulong_t pr_ictx; /* involuntary context switches */
1422     ulong_t pr_sysc; /* system calls */
1423     ulong_t pr_ioch; /* chars read and written */
1424 } prusage_t;
1425 .fi
1426 .in -2

1428 .sp
1429 .LP
1430 Microstate accounting is now continuously enabled. While this information was
1431 previously an estimate, if microstate accounting were not enabled, the current
1432 information is now never an estimate represents time the process has spent in
1433 various states.
1434 .SS "lstatus"
1436 .sp
1437 .LP
1436 Contains a \fBprheader\fR structure followed by an array of \fBBlwpstatus\fR
1437 structures, one for each active lwp in the process (see also
1438 \fB/proc/\fR\fR\fR\fR\fR\fR\fR\fR\fR\fR\fR\fR\fR\fR\fR\fR\fR\fR\fR\fR\fR\fR\fR\fR\fR\fR\fR\fR\fR\fR\fR\fR\fR\fR\fR\fR\fR\fR\fR\fR\fR, below). The
1439 \fBprheader\fR structure describes the number and size of the array entries
1440 that follow.
1441 .sp
1442 .in +2
1443 .nf
1444 typedef struct prheader {
1445     long pr_nent; /* number of entries */
1446     size_t pr_entsize; /* size of each entry, in bytes */
1447 } prheader_t;
1448 .fi
1449 .in -2

1451 .sp
1452 .LP
1453 The \fBBlwpstatus\fR structure may grow by the addition of elements at the end
1454 in future releases of the system. Programs must use \fBpr_entsize\fR in the
1455 file header to index through the array. These comments apply to all \fB/proc/\fR
1456 files that include a \fBprheader\fR structure (\fBBlpinfo\fR and \fBBlusage\fR,
1457 below).
1458 .SS "lpinfo"
1461 .sp
1462 .LP
1460 Contains a \fBprheader\fR structure followed by an array of \fBBlpinfo\fR
1461 structures, one for each active and zombie lwp in the process. See also
1462 \fB/proc/\fR\fR\fR\fR\fR\fR\fR\fR\fR\fR\fR\fR\fR\fR\fR\fR\fR\fR\fR\fR\fR\fR\fR\fR\fR\fR\fR\fR\fR\fR\fR\fR, below.
1463 .SS "lusage"
1467 .sp
1468 .LP
1465 Contains a \fBprheader\fR structure followed by an array of \fBprusage\fR

```

1466 structures, one for each active lwp in the process, plus an additional element
 1467 at the beginning that contains the summation over all defunct lwps (lwps that
 1468 once existed but no longer exist in the process). Excluding the `\fbpr_lwpid\fr`,
 1469 `\fbpr_tstamp\fr`, `\fbpr_create\fr`, and `\fbpr_term\fr` entries, the entry-by-entry
 1470 summation over all these structures is the definition of the process usage
 1471 information obtained from the `\fbusage\fr` file. (See also
 1472 `\fb/proc/\fr\lwpid\fr\fb/lwp/\fr\lwpid\fr/\fbusage\fr`, below.)
 1473 .SS "lwp"
 1474 .sp
 1474 .LP
 1475 A directory containing entries each of which names an active or zombie lwp
 1476 within the process. These entries are themselves directories containing
 1477 additional files as described below. Only the `\fblwpsinfo\fr` file exists in the
 1478 directory of a zombie lwp.
 1479 .SH STRUCTURE OF `\fb/proc/\fr\lwpid\fr\fb/lwp/\fr\lwpid\fr`
 1485 .sp
 1480 .LP
 1481 A given directory `\fb/proc/\fr\lwpid\fr\fb/lwp/\fr\lwpid\fr` contains the
 1482 following entries:
 1483 .SS "lwpctl"
 1490 .sp
 1484 .LP
 1485 Write-only control file. The messages written to this file affect the specific
 1486 lwp rather than the representative lwp, as is the case for the process's
 1487 `\fbctl\fr` file.
 1488 .SS "lwpstatus"
 1496 .sp
 1489 .LP
 1490 lwp-specific state information. This file contains the `\fbwlpstatus\fr`
 1491 structure for the specific lwp as described above for the representative lwp in
 1492 the process's `\fbstatus\fr` file.
 1493 .SS "lwpsinfo"
 1502 .sp
 1494 .LP
 1495 lwp-specific `\fbps\fr(1)` information. This file contains the `\fblwpsinfo\fr`
 1496 structure for the specific lwp as described above for the representative lwp in
 1497 the process's `\fbpsinfo\fr` file. The `\fblwpsinfo\fr` file remains accessible
 1498 after an lwp becomes a zombie.
 1499 .SS "lwpusage"
 1509 .sp
 1500 .LP
 1501 This file contains the `\fbprusage\fr` structure for the specific lwp as
 1502 described above for the process's `\fbusage\fr` file.
 1503 .SS "gwindows"
 1514 .sp
 1504 .LP
 1505 This file exists only on SPARC based machines. If it is non-empty, it contains
 1506 a `\fbgwindows_t\fr` structure, defined in `\fb<sys/regset.h>\fr`, with the values
 1507 of those SPARC register windows that could not be stored on the stack when the
 1508 lwp stopped. Conditions under which register windows are not stored on the
 1509 stack are: the stack pointer refers to nonexistent process memory or the stack
 1510 pointer is improperly aligned. If the lwp is not stopped or if there are no
 1511 register windows that could not be stored on the stack, the file is empty (the
 1512 usual case).
 1513 .SS "xregs"
 1525 .sp
 1514 .LP
 1515 Extra state registers. The extra state register set is architecture dependent;
 1516 this file is empty if the system does not support extra state registers. If the
 1517 file is non-empty, it contains an architecture dependent structure of type
 1518 `\fbprxregset_t\fr`, defined in `\fb<procfs.h>\fr`, with the values of the lwp's
 1519 extra state registers. If the lwp is not stopped, all register values are
 1520 undefined. See also the `\fbPCSXREG\fr` control operation, below.
 1521 .SS "asrs"
 1534 .sp
 1522 .LP

1523 This file exists only for 64-bit SPARC V9 processes. It contains an
 1524 `\fbasrset_t\fr` structure, defined in `<\fbsys/regset.h\fr>`, containing the
 1525 values of the lwp's platform-dependent ancillary state registers. If the lwp is
 1526 not stopped, all register values are undefined. See also the `\fbPCASRS\fr`
 1527 control operation, below.
 1528 .SS "spymaster"
 1542 .sp
 1529 .LP
 1530 For an agent lwp (see `\fbPCAGENT\fr`), this file contains a `\fbpsinfo_t\fr`
 1531 structure that corresponds to the process that created the agent lwp at the
 1532 time the agent was created. This structure is identical to that retrieved via
 1533 the `\fbpsinfo\fr` file, with one modification: the `\fbpr_time\fr` field does not
 1534 correspond to the CPU time for the process, but rather to the creation time of
 1535 the agent lwp.
 1536 .SS "templates"
 1551 .sp
 1537 .LP
 1538 A directory which contains references to the active templates for the lwp,
 1539 named by the contract type. Changes made to an active template descriptor do
 1540 not affect the original template which was activated, though they do affect the
 1541 active template. It is not possible to activate an active template descriptor.
 1542 See `\fbcontract\fr(4)`.
 1543 .SH CONTROL MESSAGES
 1559 .sp
 1544 .LP
 1545 Process state changes are effected through messages written to a process's
 1546 `\fbctl\fr` file or to an individual lwp's `\fbwlpctl\fr` file. All control
 1547 messages consist of a `\fblong\fr` that names the specific operation followed by
 1548 additional data containing the operand, if any.
 1549 .sp
 1550 .LP
 1551 Multiple control messages may be combined in a single `\fbwrite\fr(2)` (or
 1552 `\fbwritev\fr(2)`) to a control file, but no partial writes are permitted. That
 1553 is, each control message, operation code plus operand, if any, must be
 1554 presented in its entirety to the `\fbwrite\fr(2)` and not in pieces over several
 1555 system calls. If a control operation fails, no subsequent operations contained
 1556 in the same `\fbwrite\fr(2)` are attempted.
 1557 .sp
 1558 .LP
 1559 Descriptions of the allowable control messages follow. In all cases, writing a
 1560 message to a control file for a process or lwp that has terminated elicits the
 1561 error `\fbENOENT\fr`.
 1562 .SS "PCSTOP PCDSTOP PCWSTOP PCTWSTOP"
 1579 .sp
 1563 .LP
 1564 When applied to the process control file, `\fbPCSTOP\fr` directs all lwps to stop
 1565 and waits for them to stop, `\fbPCDSTOP\fr` directs all lwps to stop without
 1566 waiting for them to stop, and `\fbPCWSTOP\fr` simply waits for all lwps to stop.
 1567 When applied to an lwp control file, `\fbPCSTOP\fr` directs the specific lwp to
 1568 stop and waits until it has stopped, `\fbPCDSTOP\fr` directs the specific lwp to
 1569 stop without waiting for it to stop, and `\fbPCWSTOP\fr` simply waits for the
 1570 specific lwp to stop. When applied to an lwp control file, `\fbPCSTOP\fr` and
 1571 `\fbPCWSTOP\fr` complete when the lwp stops on an event of interest, immediately
 1572 if already so stopped; when applied to the process control file, they complete
 1573 when every lwp has stopped either on an event of interest or on a
 1574 `\fbPR_SUSPENDE\fr` stop.
 1575 .sp
 1576 .LP
 1577 `\fbPCTWSTOP\fr` is identical to `\fbPCWSTOP\fr` except that it enables the
 1578 operation to time out, to avoid waiting forever for a process or lwp that may
 1579 never stop on an event of interest. `\fbPCTWSTOP\fr` takes a `\fblong\fr` operand
 1580 specifying a number of milliseconds; the wait will terminate successfully after
 1581 the specified number of milliseconds even if the process or lwp has not
 1582 stopped; a timeout value of zero makes the operation identical to
 1583 `\fbPCWSTOP\fr`.
 1584 .sp

1585 .LP
 1586 An ``event of interest'' is either a \fBPR_REQUESTED\fR stop or a stop that has
 1587 been specified in the process's tracing flags (set by \fBPCSTRACE\fR,
 1588 \fBPCSFault\fR, \fBPCSENTRY\fR, and \fBPCSEXIT\fR). \fBPR_JOBCONTROL\fR and
 1589 \fBPR_SUSPENDED\fR stops are specifically not events of interest. (An lwp may
 1590 stop twice due to a stop signal, first showing \fBPR_SIGNALED\fR if the signal
 1591 is traced and again showing \fBPR_JOBCONTROL\fR if the lwp is set running
 1592 without clearing the signal.) If \fBPCSTOP\fR or \fBPCDSTOP\fR is applied to an
 1593 lwp that is stopped, but not on an event of interest, the stop directive takes
 1594 effect when the lwp is restarted by the competing mechanism. At that time, the
 1595 lwp enters a \fBPR_REQUESTED\fR stop before executing any user-level code.
 1596 .sp
 1597 .LP
 1598 A write of a control message that blocks is interruptible by a signal so that,
 1599 for example, an \fBalarm\fR(2) can be set to avoid waiting forever for a
 1600 process or lwp that may never stop on an event of interest. If \fBPCSTOP\fR is
 1601 interrupted, the lwp stop directives remain in effect even though the
 1602 \fBwrite\fR(2) returns an error. (Use of \fBPCWSTOP\fR with a non-zero timeout
 1603 is recommended over \fBPCWSTOP\fR with an \fBalarm\fR(2).)
 1604 .sp
 1605 .LP
 1606 A system process (indicated by the \fBPR_ISSYS\fR flag) never executes at user
 1607 level, has no user-level address space visible through \fB/proc\fR, and cannot
 1608 be stopped. Applying one of these operations to a system process or any of its
 1609 lwps elicits the error \fBBEBUSY\fR.
 1610 .SS "PCRUN"
 1628 .sp
 1611 .LP
 1612 Make an lwp runnable again after a stop. This operation takes a \fBlong\fR
 1613 operand containing zero or more of the following flags:
 1614 .sp
 1615 .ne 2
 1616 .na
 1617 \fB\fBPRCSIG\fR
 1618 .ad
 1619 .RS 12n
 1620 clears the current signal, if any (see \fBPCCSIG\fR).
 1621 .RE
 1623 .sp
 1624 .ne 2
 1625 .na
 1626 \fB\fBPRCFault\fR
 1627 .ad
 1628 .RS 12n
 1629 clears the current fault, if any (see \fBPCCFault\fR).
 1630 .RE
 1632 .sp
 1633 .ne 2
 1634 .na
 1635 \fB\fBPRSTEP\fR
 1636 .ad
 1637 .RS 12n
 1638 directs the lwp to execute a single machine instruction. On completion of the
 1639 instruction, a trace trap occurs. If \fBFLTRACE\fR is being traced, the lwp
 1640 stops; otherwise, it is sent \fBSIGTRAP\fR. If \fBSIGTRAP\fR is being traced
 1641 and is not blocked, the lwp stops. When the lwp stops on an event of interest,
 1642 the single-step directive is cancelled, even if the stop occurs before the
 1643 instruction is executed. This operation requires hardware and operating system
 1644 support and may not be implemented on all processors. It is implemented on
 1645 SPARC and x86-based machines.
 1646 .RE
 1648 .sp
 1649 .ne 2

1650 .na
 1651 \fB\fBPRSAORT\fR
 1652 .ad
 1653 .RS 12n
 1654 is meaningful only if the lwp is in a \fBPR_SYSENTRY\fR stop or is marked
 1655 \fBPR_ASLEEP\fR; it instructs the lwp to abort execution of the system call
 1656 (see \fBPCSENTRY\fR and \fBPCSEXIT\fR).
 1657 .RE
 1659 .sp
 1660 .ne 2
 1661 .na
 1662 \fB\fBPRSTOP\fR
 1663 .ad
 1664 .RS 12n
 1665 directs the lwp to stop again as soon as possible after resuming execution (see
 1666 \fBPCDSTOP\fR). In particular, if the lwp is stopped on \fBPR_SIGNALED\fR or
 1667 \fBPR_FAULTED\fR, the next stop will show \fBPR_REQUESTED\fR, no other stop
 1668 will have intervened, and the lwp will not have executed any user-level code.
 1669 .RE
 1671 .sp
 1672 .LP
 1673 When applied to an lwp control file, \fBPCRUN\fR clears any outstanding
 1674 directed-stop request and makes the specific lwp runnable. The operation fails
 1675 with \fBBEBUSY\fR if the specific lwp is not stopped on an event of interest or
 1676 has not been directed to stop or if the agent lwp exists and this is not the
 1677 agent lwp (see \fBPCAGENT\fR).
 1678 .sp
 1679 .LP
 1680 When applied to the process control file, a representative lwp is chosen for
 1681 the operation as described for \fB/proc\fR\fR\fR\fR\fR\fR\fR/status\fR. The
 1682 operation fails with \fBBEBUSY\fR if the representative lwp is not stopped on an
 1683 event of interest or has not been directed to stop or if the agent lwp exists.
 1684 If \fBPRSTEP\fR or \fBPRSTOP\fR was requested, the representative lwp is made
 1685 runnable and its outstanding directed-stop request is cleared; otherwise all
 1686 outstanding directed-stop requests are cleared and, if it was stopped on an
 1687 event of interest, the representative lwp is marked \fBPR_REQUESTED\fR. If, as
 1688 a consequence, all lwps are in the \fBPR_REQUESTED\fR or \fBPR_SUSPENDED\fR
 1689 stop state, all lwps showing \fBPR_REQUESTED\fR are made runnable.
 1690 .SS "PCSTRACE"
 1709 .sp
 1691 .LP
 1692 Define a set of signals to be traced in the process. The receipt of one of
 1693 these signals by an lwp causes the lwp to stop. The set of signals is defined
 1694 using an operand \fBsigset_t\fR contained in the control message. Receipt of
 1695 \fBSIGKILL\fR cannot be traced; if specified, it is silently ignored.
 1696 .sp
 1697 .LP
 1698 If a signal that is included in an lwp's held signal set (the signal mask) is
 1699 sent to the lwp, the signal is not received and does not cause a stop until it
 1700 is removed from the held signal set, either by the lwp itself or by setting the
 1701 held signal set with \fBPCSHOLD\fR.
 1702 .SS "PCCSIG"
 1722 .sp
 1703 .LP
 1704 The current signal, if any, is cleared from the specific or representative lwp.
 1705 .SS "PCCSIG"
 1726 .sp
 1706 .LP
 1707 The current signal and its associated signal information for the specific or
 1708 representative lwp are set according to the contents of the operand
 1709 \fBsiginfo_t\fR structure (see \fB<sys/signinfo.h>\fR). If the specified signal
 1710 number is zero, the current signal is cleared. The semantics of this operation
 1711 are different from those of \fBkill\fR(2) in that the signal is delivered to
 1712 the lwp immediately after execution is resumed (even if it is being blocked)


```

1713 and an additional \fBPR_SIGNALED\fR stop does not intervene even if the signal
1714 is traced. Setting the current signal to \fBSIGKILL\fR terminates the process
1715 immediately.
1716 .SS "PCKILL"
1738 .sp
1717 .LP
1718 If applied to the process control file, a signal is sent to the process with
1719 semantics identical to those of \fBkill\fR(2). If applied to an lwp control
1720 file, a directed signal is sent to the specific lwp. The signal is named in a
1721 \fBlong\fR operand contained in the message. Sending \fBSIGKILL\fR terminates
1722 the process immediately.
1723 .SS "PCUNKILL"
1746 .sp
1724 .LP
1725 A signal is deleted, that is, it is removed from the set of pending signals. If
1726 applied to the process control file, the signal is deleted from the process's
1727 pending signals. If applied to an lwp control file, the signal is deleted from
1728 the lwp's pending signals. The current signal (if any) is unaffected. The
1729 signal is named in a \fBlong\fR operand in the control message. It is an error
1730 (\fBEINVAL\fR) to attempt to delete \fBSIGKILL\fR.
1731 .SS "PCSHOLD"
1755 .sp
1732 .LP
1733 Set the set of held signals for the specific or representative lwp (signals
1734 whose delivery will be blocked if sent to the lwp). The set of signals is
1735 specified with a \fBsigset_t\fR operand. \fBSIGKILL\fR and \fBSIGSTOP\fR cannot
1736 be held; if specified, they are silently ignored.
1737 .SS "PCSAULT"
1762 .sp
1738 .LP
1739 Define a set of hardware faults to be traced in the process. On incurring one
1740 of these faults, an lwp stops. The set is defined via the operand
1741 \fBfltset_t\fR structure. Fault names are defined in \fB<sys/fault.h>\fR and
1742 include the following. Some of these may not occur on all processors; there may
1743 be processor-specific faults in addition to these.
1744 .sp
1745 .ne 2
1746 .na
1747 \fB\FBFLTILL\fR
1748 .ad
1749 .RS 13n
1750 illegal instruction
1751 .RE

1753 .sp
1754 .ne 2
1755 .na
1756 \fB\FBFLTPRIV\fR
1757 .ad
1758 .RS 13n
1759 privileged instruction
1760 .RE

1762 .sp
1763 .ne 2
1764 .na
1765 \fB\FBFLTBPT\fR
1766 .ad
1767 .RS 13n
1768 breakpoint trap
1769 .RE

1771 .sp
1772 .ne 2
1773 .na
1774 \fB\FBFLTTRACE\fR

```

```

1775 .ad
1776 .RS 13n
1777 trace trap (single-step)
1778 .RE

1780 .sp
1781 .ne 2
1782 .na
1783 \fB\FBFLTWATCH\fR
1784 .ad
1785 .RS 13n
1786 watchpoint trap
1787 .RE

1789 .sp
1790 .ne 2
1791 .na
1792 \fB\FBFLTACCESS\fR
1793 .ad
1794 .RS 13n
1795 memory access fault (bus error)
1796 .RE

1798 .sp
1799 .ne 2
1800 .na
1801 \fB\FBFLTBOUNDS\fR
1802 .ad
1803 .RS 13n
1804 memory bounds violation
1805 .RE

1807 .sp
1808 .ne 2
1809 .na
1810 \fB\FBFLTIOVF\fR
1811 .ad
1812 .RS 13n
1813 integer overflow
1814 .RE

1816 .sp
1817 .ne 2
1818 .na
1819 \fB\FBFLTIZDIV\fR
1820 .ad
1821 .RS 13n
1822 integer zero divide
1823 .RE

1825 .sp
1826 .ne 2
1827 .na
1828 \fB\FBFLTTFPE\fR
1829 .ad
1830 .RS 13n
1831 floating-point exception
1832 .RE

1834 .sp
1835 .ne 2
1836 .na
1837 \fB\FBFLTSTACK\fR
1838 .ad
1839 .RS 13n
1840 unrecoverable stack fault

```

```

1841 .RE
1843 .sp
1844 .ne 2
1845 .na
1846 \fB\fBFLTPAGE\fR\fR
1847 .ad
1848 .RS 13n
1849 recoverable page fault
1850 .RE

1852 .sp
1853 .LP
1854 When not traced, a fault normally results in the posting of a signal to the lwp
1855 that incurred the fault. If an lwp stops on a fault, the signal is posted to
1856 the lwp when execution is resumed unless the fault is cleared by \fBPCCFault\fR
1857 or by the \fBPRCFault\fR option of \fBPCRUN\fR. \fBFLTPAGE\fR is an exception;
1858 no signal is posted. The \fBpr_info\fR field in the \fBBlwpstatus\fR structure
1859 identifies the signal to be sent and contains machine-specific information
1860 about the fault.
1861 .SS "PCCFAULT"
1862 .LP
1863 The current fault, if any, is cleared; the associated signal will not be sent
1864 to the specific or representative lwp.
1865 .SS "PCSENTRY PCSEXIT"
1866 .LP
1867 These control operations instruct the process's lwps to stop on entry to or
1868 exit from specified system calls. The set of system calls to be traced is
1869 defined via an operand \fBsysset_t\fR structure.
1870 .sp
1871 .LP
1872 When entry to a system call is being traced, an lwp stops after having begun
1873 the call to the system but before the system call arguments have been fetched
1874 from the lwp. When exit from a system call is being traced, an lwp stops on
1875 completion of the system call just prior to checking for signals and returning
1876 to user level. At this point, all return values have been stored into the lwp's
1877 registers.
1878 .sp
1879 .LP
1880 If an lwp is stopped on entry to a system call (\fBPR_SysENTRY\fR) or when
1881 sleeping in an interruptible system call (\fBPR_ASLEEP\fR is set), it may be
1882 instructed to go directly to system call exit by specifying the \fBPR_ABORT\fR
1883 flag in a \fBPCRUN\fR control message. Unless exit from the system call is
1884 being traced, the lwp returns to user level showing \fBEINTR\fR.
1885 .SS "PCWATCH"
1886 .LP
1887 Set or clear a watched area in the controlled process from a \fBprwatch\fR
1888 structure operand:
1889 .sp
1890 .in +2
1891 .nf
1892 typedef struct prwatch {
1893     uintptr_t pr_vaddr; /* virtual address of watched area */
1894     size_t pr_size; /* size of watched area in bytes */
1895     int pr_wflags; /* watch type flags */
1896 } prwatch_t;
1897 .fi
1898 .in -2

1900 .sp
1901 .LP
1902 \fBpr_vaddr\fR specifies the virtual address of an area of memory to be watched
1903 in the controlled process. \fBpr_size\fR specifies the size of the area, in

```

```

1904 bytes. \fBpr_wflags\fR specifies the type of memory access to be monitored as a
1905 bit-mask of the following flags:
1906 .sp
1907 .ne 2
1908 .na
1909 \fBpr_BWA_READ\fR\fR
1910 .ad
1911 .RS 16n
1912 read access
1913 .RE

1915 .sp
1916 .ne 2
1917 .na
1918 \fBpr_BWA_WRITE\fR\fR
1919 .ad
1920 .RS 16n
1921 write access
1922 .RE

1924 .sp
1925 .ne 2
1926 .na
1927 \fBpr_BWA_EXEC\fR\fR
1928 .ad
1929 .RS 16n
1930 execution access
1931 .RE

1933 .sp
1934 .ne 2
1935 .na
1936 \fBpr_BWA_TRAPAFTR\fR\fR
1937 .ad
1938 .RS 16n
1939 trap after the instruction completes
1940 .RE

1942 .sp
1943 .LP
1944 If \fBpr_wflags\fR is non-empty, a watched area is established for the virtual
1945 address range specified by \fBpr_vaddr\fR and \fBpr_size\fR. If \fBpr_wflags\fR
1946 is empty, any previously-established watched area starting at the specified
1947 virtual address is cleared; \fBpr_size\fR is ignored.
1948 .sp
1949 .LP
1950 A watchpoint is triggered when an lwp in the traced process makes a memory
1951 reference that covers at least one byte of a watched area and the memory
1952 reference is as specified in \fBpr_wflags\fR. When an lwp triggers a
1953 watchpoint, it incurs a watchpoint trap. If \fBpr_BFLTWATCH\fR is being traced, the
1954 lwp stops; otherwise, it is sent a \fBBSIGTRAP\fR signal; if \fBBSIGTRAP\fR is
1955 being traced and is not blocked, the lwp stops.
1956 .sp
1957 .LP
1958 The watchpoint trap occurs before the instruction completes unless
1959 \fBpr_BWA_TRAPAFTR\fR was specified, in which case it occurs after the instruction
1960 completes. If it occurs before completion, the memory is not modified. If it
1961 occurs after completion, the memory is modified (if the access is a write
1962 access).
1963 .sp
1964 .LP
1965 Physical i/o is an exception for watchpoint traps. In this instance, there is
1966 no guarantee that memory before the watched area has already been modified (or
1967 in the case of \fBpr_BWA_TRAPAFTR\fR, that the memory following the watched area
1968 has not been modified) when the watchpoint trap occurs and the lwp stops.
1969 .sp

```

1970 .LP
 1971 \fBpr_info\fR in the \fBlwpstatus\fR structure contains information pertinent
 1972 to the watchpoint trap. In particular, the \fBsi_addr\fR field contains the
 1973 virtual address of the memory reference that triggered the watchpoint, and the
 1974 \fBsi_code\fR field contains one of \fBTRAP_RWATCH\fR, \fBTRAP_WWATCH\fR, or
 1975 \fBTRAP_XWATCH\fR, indicating read, write, or execute access, respectively. The
 1976 \fBsi_trapafter\fR field is zero unless \fBWA_TRAPAFTR\fR is in effect for
 1977 this watched area; non-zero indicates that the current instruction is not the
 1978 instruction that incurred the watchpoint trap. The \fBsi_pc\fR field contains
 1979 the virtual address of the instruction that incurred the trap.
 1980 .sp
 1981 .LP
 1982 A watchpoint trap may be triggered while executing a system call that makes
 1983 reference to the traced process's memory. The lwp that is executing the system
 1984 call incurs the watchpoint trap while still in the system call. If it stops as
 1985 a result, the \fBlwpstatus\fR structure contains the system call number and its
 1986 arguments. If the lwp does not stop, or if it is set running again without
 1987 clearing the signal or fault, the system call fails with \fBEFAULT\fR. If
 1988 \fBWA_TRAPAFTR\fR was specified, the memory reference will have completed and
 1989 the memory will have been modified (if the access was a write access) when the
 1990 watchpoint trap occurs.
 1991 .sp
 1992 .LP
 1993 If more than one of \fBWA_READ\fR, \fBWA_WRITE\fR, and \fBWA_EXEC\fR is
 1994 specified for a watched area, and a single instruction incurs more than one of
 1995 the specified types, only one is reported when the watchpoint trap occurs. The
 1996 precedence is \fBWA_EXEC\fR, \fBWA_READ\fR, \fBWA_WRITE\fR (\fBWA_EXEC\fR and
 1997 \fBWA_READ\fR take precedence over \fBWA_WRITE\fR), unless \fBWA_TRAPAFTR\fR
 1998 was specified, in which case it is \fBWA_WRITE\fR, \fBWA_READ\fR, \fBWA_EXEC\fR
 1999 (\fBWA_WRITE\fR takes precedence).
 2000 .sp
 2001 .LP
 2002 \fBPCWATCH\fR fails with \fBEINVAL\fR if an attempt is made to specify
 2003 overlapping watched areas or if \fBpr_wflags\fR contains flags other than those
 2004 specified above. It fails with \fBENOMEM\fR if an attempt is made to establish
 2005 more watched areas than the system can support (the system can support
 2006 thousands).
 2007 .sp
 2008 .LP
 2009 The child of a \fBvfork\fR(2) borrows the parent's address space. When a
 2010 \fBvfork\fR(2) is executed by a traced process, all watched areas established
 2011 for the parent are suspended until the child terminates or performs an
 2012 \fBexec\fR(2). Any watched areas established independently in the child are
 2013 cancelled when the parent resumes after the child's termination or
 2014 \fBexec\fR(2). \fBPCWATCH\fR fails with \fBEBUSY\fR if applied to the parent of
 2015 a \fBvfork\fR(2) before the child has terminated or performed an \fBexec\fR(2).
 2016 The \fBPR_VFORKP\fR flag is set in the \fBpstatus\fR structure for such a
 2017 parent process.
 2018 .sp
 2019 .LP
 2020 Certain accesses of the traced process's address space by the operating system
 2021 are immune to watchpoints. The initial construction of a signal stack frame
 2022 when a signal is delivered to an lwp will not trigger a watchpoint trap even if
 2023 the new frame covers watched areas of the stack. Once the signal handler is
 2024 entered, watchpoint traps occur normally. On SPARC based machines, register
 2025 window overflow and underflow will not trigger watchpoint traps, even if the
 2026 register window save areas cover watched areas of the stack.
 2027 .sp
 2028 .LP
 2029 Watched areas are not inherited by child processes, even if the traced
 2030 process's inherit-on-fork mode, \fBPR_FORK\fR, is set (see \fBPCSET\fR, below).
 2031 All watched areas are cancelled when the traced process performs a successful
 2032 \fBexec\fR(2).
 2033 .SS "PCSET PCUNSET"
 2062 .sp
 2034 .LP

2035 \fBPCSET\fR sets one or more modes of operation for the traced process.
 2036 \fBPCUNSET\fR unsets these modes. The modes to be set or unset are specified by
 2037 flags in an operand \fBlong\fR in the control message:
 2038 .sp
 2039 .ne 2
 2040 .na
 2041 \fBPR_FORK\fR
 2042 .ad
 2043 .RS 13n
 2044 (inherit-on-fork): When set, the process's tracing flags and its
 2045 inherit-on-fork mode are inherited by the child of a \fBfork\fR(2),
 2046 \fBfork1\fR(2), or \fBvfork\fR(2). When unset, child processes start with all
 2047 tracing flags cleared.
 2048 .RE
 2050 .sp
 2051 .ne 2
 2052 .na
 2053 \fBPR_RLC\fR
 2054 .ad
 2055 .RS 13n
 2056 (run-on-last-close): When set and the last writable \fB/proc\fR file descriptor
 2057 referring to the traced process or any of its lwps is closed, all of the
 2058 process's tracing flags and watched areas are cleared, any outstanding stop
 2059 directives are canceled, and if any lwps are stopped on events of interest,
 2060 they are set running as though \fBPCRUN\fR had been applied to them. When
 2061 unset, the process's tracing flags and watched areas are retained and lwps are
 2062 not set running on last close.
 2063 .RE
 2065 .sp
 2066 .ne 2
 2067 .na
 2068 \fBPR_KLC\fR
 2069 .ad
 2070 .RS 13n
 2071 (kill-on-last-close): When set and the last writable \fB/proc\fR file
 2072 descriptor referring to the traced process or any of its lwps is closed, the
 2073 process is terminated with \fBSIGKILL\fR.
 2074 .RE
 2076 .sp
 2077 .ne 2
 2078 .na
 2079 \fBPR_ASYNC\fR
 2080 .ad
 2081 .RS 13n
 2082 (asynchronous-stop): When set, a stop on an event of interest by one lwp does
 2083 not directly affect any other lwp in the process. When unset and an lwp stops
 2084 on an event of interest other than \fBPR_REQUESTED\fR, all other lwps in the
 2085 process are directed to stop.
 2086 .RE
 2088 .sp
 2089 .ne 2
 2090 .na
 2091 \fBPR_MSACCT\fR
 2092 .ad
 2093 .RS 13n
 2094 (microstate accounting): Microstate accounting is now continuously enabled.
 2095 This flag is deprecated and no longer has any effect upon microstate
 2096 accounting. Applications may toggle this flag; however, microstate accounting
 2097 will remain enabled regardless.
 2098 .RE
 2100 .sp

```

2101 .ne 2
2102 .na
2103 \fb\fbPR_MSFORK\fr\fr
2104 .ad
2105 .RS 13n
2106 (inherit microstate accounting): All processes now inherit microstate
2107 accounting, as it is continuously enabled. This flag has been deprecated and
2108 its use no longer has any effect upon the behavior of microstate accounting.
2109 .RE

2111 .sp
2112 .ne 2
2113 .na
2114 \fb\fbPR_BPTADJ\fr\fr
2115 .ad
2116 .RS 13n
2117 (breakpoint trap pc adjustment): On x86-based machines, a breakpoint trap
2118 leaves the program counter (the \fBEIP\fr) referring to the breakpointed
2119 instruction plus one byte. When \fbPR_BPTADJ\fr is set, the system will adjust
2120 the program counter back to the location of the breakpointed instruction when
2121 the lwp stops on a breakpoint. This flag has no effect on SPARC based machines,
2122 where breakpoint traps leave the program counter referring to the breakpointed
2123 instruction.
2124 .RE

2126 .sp
2127 .ne 2
2128 .na
2129 \fb\fbPR_PTRACE\fr\fr
2130 .ad
2131 .RS 13n
2132 (ptrace-compatibility): When set, a stop on an event of interest by the traced
2133 process is reported to the parent of the traced process by \fbwait\fr(3C),
2134 \fbSIGTRAP\fr is sent to the traced process when it executes a successful
2135 \fbexec\fr(2), setuid/setgid flags are not honored for execs performed by the
2136 traced process, any exec of an object file that the traced process cannot read
2137 fails, and the process dies when its parent dies. This mode is deprecated; it
2138 is provided only to allow \fbptrace\fr(3C) to be implemented as a library
2139 function using \fb/proc\fr.
2140 .RE

2142 .sp
2143 .LP
2144 It is an error (\fBEINVAL\fr) to specify flags other than those described above
2145 or to apply these operations to a system process. The current modes are
2146 reported in the \fbpr_flags\fr field of \fb/proc/\fr\flpid\fr\fb/status\fr and
2147 \fb/proc/\fr\flpid\fr\fb/lwp/\fr\flwp\fr\fb/lwpstatus\fr.
2148 .SS "PCSREG"
2149 .LP
2150 Set the general registers for the specific or representative lwp according to
2151 the operand \fbprgregset_t\fr structure.
2152 .sp
2153 .LP
2154 On SPARC based systems, only the condition-code bits of the processor-status
2155 register (R_PSR) of SPARC V8 (32-bit) processes can be modified by
2156 \fbPCSREG\fr. Other privileged registers cannot be modified at all.
2157 .sp
2158 .LP
2159 On x86-based systems, only certain bits of the flags register (EFL) can be
2160 modified by \fbPCSREG\fr: these include the condition codes, direction-bit, and
2161 overflow-bit.
2162 .sp
2163 .LP
2164 \fbPCSREG\fr fails with \fbEBUSY\fr if the lwp is not stopped on an event of
2165 interest.
```

```

2166 .SS "PCSVADDR"
2167 .sp
2168 .LP
2169 Set the address at which execution will resume for the specific or
2170 representative lwp from the operand \fblong\fr. On SPARC based systems, both
2171 %pc and %npc are set, with %npc set to the instruction following the virtual
2172 address. On x86-based systems, only %eip is set. \fbPCSVADDR\fr fails with
2173 \fbEBUSY\fr if the lwp is not stopped on an event of interest.
2174 .SS "PCSFPREG"
2175 .sp
2176 .LP
2177 Set the floating-point registers for the specific or representative lwp
2178 according to the operand \fbprfpregsset_t\fr structure. An error (\fBEINVAL\fr)
2179 is returned if the system does not support floating-point operations (no
2180 floating-point hardware and the system does not emulate floating-point machine
2181 instructions). \fbPCSFPREG\fr fails with \fbEBUSY\fr if the lwp is not stopped
2182 on an event of interest.
2183 .SS "PCSXREG"
2184 .sp
2185 .LP
2186 Set the extra state registers for the specific or representative lwp according
2187 to the architecture-dependent operand \fbprxregsset_t\fr structure. An error
2188 (\fBEINVAL\fr) is returned if the system does not support extra state
2189 registers. \fbPCSXREG\fr fails with \fbEBUSY\fr if the lwp is not stopped on an
2190 event of interest.
2191 .SS "PCSASRS"
2192 .sp
2193 .LP
2194 Set the ancillary state registers for the specific or representative lwp
2195 according to the SPARC V9 platform-dependent operand \fbasrset_t\fr structure.
2196 An error (\fBEINVAL\fr) is returned if either the target process or the
2197 controlling process is not a 64-bit SPARC V9 process. Most of the ancillary
2198 state registers are privileged registers that cannot be modified. Only those
2199 that can be modified are set; all others are silently ignored. \fbPCSASRS\fr
2200 fails with \fbEBUSY\fr if the lwp is not stopped on an event of interest.
2201 .SS "PCAGENT"
2202 .sp
2203 .LP
2204 Create an agent lwp in the controlled process with register values from the
2205 operand \fbprgregset_t\fr structure (see \fbPCSREG\fr, above). The agent lwp is
2206 created in the stopped state showing \fbPR_REQUESTED\fr and with its held
2207 signal set (the signal mask) having all signals except \fbSIGKILL\fr and
2208 \fbSIGSTOP\fr blocked.
2209 .sp
2210 .LP
2211 The \fbPCAGENT\fr operation fails with \fbEBUSY\fr unless the process is fully
2212 stopped via \fb/proc\fr, that is, unless all of the lwps in the process are
2213 stopped either on events of interest or on \fbPR_SUSPENDED\fr, or are stopped
2214 on \fbPR_JOBCONTROL\fr and have been directed to stop via \fbPCDSTOP\fr. It
2215 fails with \fbEBUSY\fr if an agent lwp already exists. It fails with
2216 \fbBENOMEM\fr if system resources for creating new lwps have been exhausted.
2217 .sp
2218 .LP
2219 Any \fbPCRUN\fr operation applied to the process control file or to the control
2220 file of an lwp other than the agent lwp fails with \fbEBUSY\fr as long as the
2221 agent lwp exists. The agent lwp must be caused to terminate by executing the
2222 \fbSYS_lwp_exit\fr system call trap before the process can be restarted.
2223 .sp
2224 .LP
2225 Once the agent lwp is created, its lwp-ID can be found by reading the process
2226 status file. To facilitate opening the agent lwp's control and status files,
2227 the directory name \fb/proc/\fr\flpid\fr\fb/lwp/agent\fr is accepted for
2228 lookup operations as an invisible alias for
2229 \fb/proc/\fr\flpid\fr\fb/lwp/\fr\flwpid\fr \fr \flwpid\fr being the lwp-ID of
2230 the agent lwp (invisible in the sense that the name 'agent' does not appear
2231 in a directory listing of \fb/proc/\fr\flpid\fr\fb/lwp\fr obtained from
```

```

2227 \fBlis\fR(1), \fBgetdents\fR(2), or \fBreaddir\fR(3C)).
2228 .sp
2229 .LP
2230 The purpose of the agent lwp is to perform operations in the controlled process
2231 on behalf of the controlling process: to gather information not directly
2232 available via \fB/proc\fR files, or in general to make the process change state
2233 in ways not directly available via \fB/proc\fR control operations. To make use
2234 of an agent lwp, the controlling process must be capable of making it execute
2235 system calls (specifically, the \fBSYS_lwp_exit\fR system call trap). The
2236 register values given to the agent lwp on creation are typically the registers
2237 of the representative lwp, so that the agent lwp can use its stack.
2238 .sp
2239 .LP
2240 If the controlling process neglects to force the agent lwp to execute the
2241 \fBSYS_lwp_exit\fR system call (due to either logic error or fatal failure on
2242 the part of the controlling process), the agent lwp will remain in the target
2243 process. For purposes of being able to debug these otherwise rogue agents,
2244 information as to the creator of the agent lwp is reflected in that lwp's
2245 \fBspymaster\fR file in \fB/proc\fR. Should the target process generate a core
2246 dump with the agent lwp in place, this information will be available via the
2247 \fBNT_SPYMASTER\fR note in the core file (see \fBcore\fR(4)).
2248 .sp
2249 .LP
2250 The agent lwp is not allowed to execute any variation of the \fBSYS_fork\fR or
2251 \fBSYS_exec\fR system call traps. Attempts to do so yield \fBENOTSUP\fR to the
2252 agent lwp.
2253 .sp
2254 .LP
2255 Symbolic constants for system call trap numbers like \fBSYS_lwp_exit\fR and
2256 \fBSYS_lwp_create\fR can be found in the header file <\fBsys/syscall.h\fR>.
2257 .SS "PCREAD PCWRITE"
2258 .sp
2259 .LP
2259 Read or write the target process's address space via a \fBpriovec\fR structure
2260 operand:
2261 .sp
2262 .in +2
2263 .nf
2264 typedef struct priovec {
2265     void *pio_base;      /* buffer in controlling process */
2266     size_t pio_len;     /* size of read/write request in bytes */
2267     off_t pio_offset;   /* virtual address in target process */
2268 } priovec_t;
2269 .fi
2270 .in -2

```

```

2272 .sp
2273 .LP
2274 These operations have the same effect as \fBpread\fR(2) and \fBpwrite\fR(2),
2275 respectively, of the target process's address space file. The difference is
2276 that more than one \fBPCREAD\fR or \fBPCWRITE\fR control operation can be
2277 written to the control file at once, and they can be interspersed with other
2278 control operations in a single write to the control file. This is useful, for
2279 example, when planting many breakpoint instructions in the process's address
2280 space, or when stepping over a breakpointed instruction. Unlike \fBpread\fR(2)
2281 and \fBpwrite\fR(2), no provision is made for partial reads or writes; if the
2282 operation cannot be performed completely, it fails with \fBEIO\fR.
2283 .SS "PCNICE"
2284 .sp
2285 .LP
2285 The traced process's \fBnice\fR(2) value is incremented by the amount in the
2286 operand \fBlong\fR. Only a process with the {\fBPRIV_PROC_PRIOCNTL\fR}
2287 privilege asserted in its effective set can better a process's priority in this
2288 way, but any user may lower the priority. This operation is not meaningful for
2289 all scheduling classes.
2290 .SS "PCSCRED"

```

```

2328 .sp
2329 .LP
2329 Set the target process credentials to the values contained in the
2329 \fBprcred_t\fR structure operand (see \fB/proc/\fR\fR\fR\fR\fR\fR\fR). The
2329 effective, real, and saved user-IDs and group-IDs of the target process are
2329 set. The target process's supplementary groups are not changed; the
2329 \fBprngroups\fR and \fBprgroups\fR members of the structure operand are
2329 ignored. Only the privileged processes can perform this operation; for all
2329 others it fails with \fBPERM\fR.
2329 .SS "PCSCREDD"
2338 .sp
2339 .LP
2339 Operates like \fBPCSCRED\fR but also sets the supplementary groups; the length
2339 of the data written with this control operation should be "sizeof
2339 (\fBprcred_t\fR) + sizeof (\fBgid_t\fR) * (#groups - 1)".
2340 .SS "PCSPRIV"
2344 .sp
2345 .LP
2345 Set the target process privilege to the values contained in the \fBprpriv_t\fR
2345 operand (see \fB/proc/pid/priv\fR). The effective, permitted, inheritable, and
2345 limit sets are all changed. Privilege flags can also be set. The process is
2345 made privilege aware unless it can relinquish privilege awareness. See
2345 \fBprivileges\fR(5).
2345 .sp
2346 .LP
2346 The limit set of the target process cannot be grown. The other privilege sets
2346 must be subsets of the intersection of the effective set of the calling process
2346 with the new limit set of the target process or subsets of the original values
2346 of the sets in the target process.
2347 .sp
2348 .LP
2348 If any of the above restrictions are not met, \fBPERM\fR is returned. If the
2348 structure written is improperly formatted, \fBINVAL\fR is returned.
2349 .SH PROGRAMMING NOTES
2352 .sp
2353 .LP
2353 For security reasons, except for the \fBpsinfo\fR, \fBusage\fR, \fBlpsinfo\fR,
2353 \fBlusage\fR, \fBlwpsinfo\fR, and \fBlwusage\fR files, which are
2353 world-readable, and except for privileged processes, an open of a \fB/proc/\fR
2353 file fails unless both the user-ID and group-ID of the caller match those of
2353 the traced process and the process's object file is readable by the caller. The
2353 effective set of the caller is a superset of both the inheritable and the
2353 permitted set of the target process. The limit set of the caller is a superset
2353 of the limit set of the target process. Except for the world-readable files
2353 just mentioned, files corresponding to setuid and setgid processes can be
2353 opened only by the appropriately privileged process.
2353 .sp
2354 .LP
2354 A process that is missing the basic privilege {\fBPRIV_PROC_INFO\fR} cannot see
2354 any processes under \fB/proc/\fR that it cannot send a signal to.
2355 .sp
2356 .LP
2356 A process that has {\fBPRIV_PROC_OWNER\fR} asserted in its effective set can
2356 open any file for reading. To manipulate or control a process, the controlling
2356 process must have at least as many privileges in its effective set as the
2356 target process has in its effective, inheritable, and permitted sets. The limit
2356 set of the controlling process must be a superset of the limit set of the
2356 target process. Additional restrictions apply if any of the uids of the target
2356 process are 0. See \fBprivileges\fR(5).
2356 .sp
2357 .LP
2357 Even if held by a privileged process, an open process or lwp file descriptor
2357 (other than file descriptors for the world-readable files) becomes invalid if
2357 the traced process performs an \fBexec\fR(2) of a setuid/setgid object file or
2357 an object file that the traced process cannot read. Any operation performed on
2357 an invalid file descriptor, except \fBclose\fR(2), fails with \fBEAGAIN\fR. In

```

2353 this situation, if any tracing flags are set and the process or any lwp file
 2354 descriptor is open for writing, the process will have been directed to stop and
 2355 its run-on-last-close flag will have been set (see \fBPCSET\fR). This enables a
 2356 controlling process (if it has permission) to reopen the \fB/proc\fR files to
 2357 get new valid file descriptors, close the invalid file descriptors, unset the
 2358 run-on-last-close flag (if desired), and proceed. Just closing the invalid file
 2359 descriptors causes the traced process to resume execution with all tracing
 2360 flags cleared. Any process not currently open for writing via \fB/proc\fR, but
 2361 that has left-over tracing flags from a previous open, and that executes a
 2362 setuid/setgid or unreadable object file, will not be stopped but will have all
 2363 its tracing flags cleared.

2364 .sp
 2365 .LP
 2366 To wait for one or more of a set of processes or lwps to stop or terminate,
 2367 \fB/proc\fR file descriptors (other than those obtained by opening the
 2368 \fB/cwd\fR or \fB/root\fR directories or by opening files in the \fB/fd\fR or
 2369 \fB/object\fR directories) can be used in a \fBpoll\fR(2) system call. When
 2370 requested and returned, either of the polling events \fBPOLLPRI\fR or
 2371 \fBPOLLWRNORM\fR indicates that the process or lwp stopped on an event of
 2372 interest. Although they cannot be requested, the polling events \fBPOLLHUP\fR,
 2373 \fBPOLLERR\fR, and \fBPOLLNVAL\fR may be returned. \fBPOLLHUP\fR indicates that
 2374 the process or lwp has terminated. \fBPOLLERR\fR indicates that the file
 2375 descriptor has become invalid. \fBPOLLNVAL\fR is returned immediately if
 2376 \fBPOLLPRI\fR or \fBPOLLWRNORM\fR is requested on a file descriptor referring
 2377 to a system process (see \fBPCSTOP\fR). The requested events may be empty to
 2378 wait simply for termination.

2379 .SH FILES
 2421 .sp
 2380 .ne 2
 2381 .na
 2382 \fB/fB/proc\fR
 2383 .ad
 2384 .sp .6
 2385 .RS 4n
 2386 directory (list of processes)
 2387 .RE

2389 .sp
 2390 .ne 2
 2391 .na
 2392 \fB/fB/proc/\fR\fR
 2393 .ad
 2394 .sp .6
 2395 .RS 4n
 2396 specific process directory
 2397 .RE

2399 .sp
 2400 .ne 2
 2401 .na
 2402 \fB/fB/proc/self\fR
 2403 .ad
 2404 .sp .6
 2405 .RS 4n
 2406 alias for a process's own directory
 2407 .RE

2409 .sp
 2410 .ne 2
 2411 .na
 2412 \fB/fB/proc/\fR/as\fR
 2413 .ad
 2414 .sp .6
 2415 .RS 4n
 2416 address space file
 2417 .RE

2419 .sp
 2420 .ne 2
 2421 .na
 2422 \fB/fB/proc/\fR/ctl\fR
 2423 .ad
 2424 .sp .6
 2425 .RS 4n
 2426 process control file
 2427 .RE

2429 .sp
 2430 .ne 2
 2431 .na
 2432 \fB/fB/proc/\fR/status\fR
 2433 .ad
 2434 .sp .6
 2435 .RS 4n
 2436 process status
 2437 .RE

2439 .sp
 2440 .ne 2
 2441 .na
 2442 \fB/fB/proc/\fR/lstatus\fR
 2443 .ad
 2444 .sp .6
 2445 .RS 4n
 2446 array of lwp status structs
 2447 .RE

2449 .sp
 2450 .ne 2
 2451 .na
 2452 \fB/fB/proc/\fR/psinfo\fR
 2453 .ad
 2454 .sp .6
 2455 .RS 4n
 2456 process \fBps\fR(1) info
 2457 .RE

2459 .sp
 2460 .ne 2
 2461 .na
 2462 \fB/fB/proc/\fR/lpsinfo\fR
 2463 .ad
 2464 .sp .6
 2465 .RS 4n
 2466 array of lwp \fBps\fR(1) info structs
 2467 .RE

2469 .sp
 2470 .ne 2
 2471 .na
 2472 \fB/fB/proc/\fR/map\fR
 2473 .ad
 2474 .sp .6
 2475 .RS 4n
 2476 address space map
 2477 .RE

2479 .sp
 2480 .ne 2
 2481 .na
 2482 \fB/fB/proc/\fR/xmap\fR
 2483 .ad

```

2484 .sp .6
2485 .RS 4n
2486 extended address space map
2487 .RE

2489 .sp
2490 .ne 2
2491 .na
2492 \fB\fB/proc/\fIpid\fR/rmap\fR\fR
2493 .ad
2494 .sp .6
2495 .RS 4n
2496 reserved address map
2497 .RE

2499 .sp
2500 .ne 2
2501 .na
2502 \fB\fB/proc/\fIpid\fR/cred\fR\fR
2503 .ad
2504 .sp .6
2505 .RS 4n
2506 process credentials
2507 .RE

2509 .sp
2510 .ne 2
2511 .na
2512 \fB\fB/proc/\fIpid\fR/priv\fR\fR
2513 .ad
2514 .sp .6
2515 .RS 4n
2516 process privileges
2517 .RE

2519 .sp
2520 .ne 2
2521 .na
2522 \fB\fB/proc/\fIpid\fR/sigact\fR\fR
2523 .ad
2524 .sp .6
2525 .RS 4n
2526 process signal actions
2527 .RE

2529 .sp
2530 .ne 2
2531 .na
2532 \fB\fB/proc/\fIpid\fR/auxv\fR\fR
2533 .ad
2534 .sp .6
2535 .RS 4n
2536 process aux vector
2537 .RE

2539 .sp
2540 .ne 2
2541 .na
2542 \fB\fB/proc/\fIpid\fR/ldt\fR\fR
2543 .ad
2544 .sp .6
2545 .RS 4n
2546 process \fBBLDT\fR (x86 only)
2547 .RE

2549 .sp

```

```

2550 .ne 2
2551 .na
2552 \fB\fB/proc/\fIpid\fR/usage\fR\fR
2553 .ad
2554 .sp .6
2555 .RS 4n
2556 process usage
2557 .RE

2559 .sp
2560 .ne 2
2561 .na
2562 \fB\fB/proc/\fIpid\fR/lusage\fR\fR
2563 .ad
2564 .sp .6
2565 .RS 4n
2566 array of lwp usage structs
2567 .RE

2569 .sp
2570 .ne 2
2571 .na
2572 \fB\fB/proc/\fIpid\fR/path\fR\fR
2573 .ad
2574 .sp .6
2575 .RS 4n
2576 symbolic links to process open files
2577 .RE

2579 .sp
2580 .ne 2
2581 .na
2582 \fB\fB/proc/\fIpid\fR/pagedata\fR\fR
2583 .ad
2584 .sp .6
2585 .RS 4n
2586 process page data
2587 .RE

2589 .sp
2590 .ne 2
2591 .na
2592 \fB\fB/proc/\fIpid\fR/watch\fR\fR
2593 .ad
2594 .sp .6
2595 .RS 4n
2596 active watchpoints
2597 .RE

2599 .sp
2600 .ne 2
2601 .na
2602 \fB\fB/proc/\fIpid\fR/cwd\fR\fR
2603 .ad
2604 .sp .6
2605 .RS 4n
2606 alias for the current working directory
2607 .RE

2609 .sp
2610 .ne 2
2611 .na
2612 \fB\fB/proc/\fIpid\fR/root\fR\fR
2613 .ad
2614 .sp .6
2615 .RS 4n

```

```

2616 alias for the root directory
2617 .RE

2619 .sp
2620 .ne 2
2621 .na
2622 \fB\fB/proc/\fIpid\fR/fd\fR\fR
2623 .ad
2624 .sp .6
2625 .RS 4n
2626 directory (list of open files)
2627 .RE

2629 .sp
2630 .ne 2
2631 .na
2632 \fB\fB/proc/\fIpid\fR/fd/*\fR\fR
2633 .ad
2634 .sp .6
2635 .RS 4n
2636 aliases for process's open files
2637 .RE

2639 .sp
2640 .ne 2
2641 .na
2642 \fB\fB/proc/\fIpid\fR/object\fR\fR
2643 .ad
2644 .sp .6
2645 .RS 4n
2646 directory (list of mapped files)
2647 .RE

2649 .sp
2650 .ne 2
2651 .na
2652 \fB\fB/proc/\fIpid\fR/object/a.out\fR\fR
2653 .ad
2654 .sp .6
2655 .RS 4n
2656 alias for process's executable file
2657 .RE

2659 .sp
2660 .ne 2
2661 .na
2662 \fB\fB/proc/\fIpid\fR/object/*\fR\fR
2663 .ad
2664 .sp .6
2665 .RS 4n
2666 aliases for other mapped files
2667 .RE

2669 .sp
2670 .ne 2
2671 .na
2672 \fB\fB/proc/\fIpid\fR/lwp\fR\fR
2673 .ad
2674 .sp .6
2675 .RS 4n
2676 directory (list of lwps)
2677 .RE

2679 .sp
2680 .ne 2
2681 .na

```

```

2682 \fB\fB/proc/\fIpid\fR/lwp/\fIlwpid\fR\fR\fR
2683 .ad
2684 .sp .6
2685 .RS 4n
2686 specific lwp directory
2687 .RE

2689 .sp
2690 .ne 2
2691 .na
2692 \fB\fB/proc/\fIpid\fR/lwp/agent\fR\fR
2693 .ad
2694 .sp .6
2695 .RS 4n
2696 alias for the agent lwp directory
2697 .RE

2699 .sp
2700 .ne 2
2701 .na
2702 \fB\fB/proc/\fIpid\fR/lwp/\fIlwpid\fR/lwpctl\fR\fR
2703 .ad
2704 .sp .6
2705 .RS 4n
2706 lwp control file
2707 .RE

2709 .sp
2710 .ne 2
2711 .na
2712 \fB\fB/proc/\fIpid\fR/lwp/\fIlwpid\fR/lwpstatus\fR\fR
2713 .ad
2714 .sp .6
2715 .RS 4n
2716 lwp status
2717 .RE

2719 .sp
2720 .ne 2
2721 .na
2722 \fB\fB/proc/\fIpid\fR/lwp/\fIlwpid\fR/lwpsinfo\fR\fR
2723 .ad
2724 .sp .6
2725 .RS 4n
2726 lwp \fBps\fR(1) info
2727 .RE

2729 .sp
2730 .ne 2
2731 .na
2732 \fB\fB/proc/\fIpid\fR/lwp/\fIlwpid\fR/lwpusage\fR\fR
2733 .ad
2734 .sp .6
2735 .RS 4n
2736 lwp usage
2737 .RE

2739 .sp
2740 .ne 2
2741 .na
2742 \fB\fB/proc/\fIpid\fR/lwp/\fIlwpid\fR/gwindows\fR\fR
2743 .ad
2744 .sp .6
2745 .RS 4n
2746 register windows (SPARC only)
2747 .RE

```



```

2749 .sp
2750 .ne 2
2751 .na
2752 \fB\fB/proc/\fIpid\fR/lwp/\fIilwpid\fR/xregs\fR\fR
2753 .ad
2754 .sp .6
2755 .RS 4n
2756 extra state registers
2757 .RE

2759 .sp
2760 .ne 2
2761 .na
2762 \fB\fB/proc/\fIpid\fR/lwp/\fIilwpid\fR/asrs\fR\fR
2763 .ad
2764 .sp .6
2765 .RS 4n
2766 ancillary state registers (SPARC V9 only)
2767 .RE

2769 .sp
2770 .ne 2
2771 .na
2772 \fB\fB/proc/\fIpid\fR/lwp/\fIilwpid\fR/spymaster\fR\fR
2773 .ad
2774 .sp .6
2775 .RS 4n
2776 For an agent LWP, the controlling process
2777 .RE

2779 .SH SEE ALSO
2822 .sp
2780 .LP
2781 \fBls\fR(1), \fBps\fR(1), \fBchroot\fR(1M), \fBalarm\fR(2), \fBbrk\fR(2),
2782 \fBchdir\fR(2), \fBchroot\fR(2), \fBclose\fR(2), \fBcreat\fR(2), \fBdup\fR(2),
2783 \fBexec\fR(2), \fBfcntl\fR(2), \fBfork\fR(2), \fBfork1\fR(2), \fBfstat\fR(2),
2784 \fBgetdents\fR(2), \fBgetustack\fR(2), \fBkill\fR(2), \fBlseek\fR(2),
2785 \fBmmap\fR(2), \fBnice\fR(2), \fBopen\fR(2), \fBpoll\fR(2), \fBpread\fR(2),
2786 \fBptrace\fR(3C), \fBpwrite\fR(2), \fBread\fR(2), \fBreadlink\fR(2),
2787 \fBreadv\fR(2), \fBshmget\fR(2), \fBsigaction\fR(2), \fBsigaltstack\fR(2),
2788 \fBvfork\fR(2), \fBwrite\fR(2), \fBwritev\fR(2), \fB_stack_grow\fR(3C),
2789 \fBbreaddir\fR(3C), \fBpthread_create\fR(3C), \fBpthread_join\fR(3C),
2790 \fBsiginfo.h\fR(3HEAD), \fBsignal.h\fR(3HEAD), \fBthr_create\fR(3C),
2791 \fBthr_join\fR(3C), \fBtypes32.h\fR(3HEAD), \fBucontext.h\fR(3HEAD),
2792 \fBwait\fR(3C), \fBcontract\fR(4), \fBcore\fR(4), \fBprocess\fR(4),
2793 \fBblfcompile\fR(5), \fBprivileges\fR(5), \fBsecurity-flags\fR(5)
2836 \fBblfcompile\fR(5), \fBprivileges\fR(5)
2794 .SH DIAGNOSTICS
2838 .sp
2795 .LP
2796 Errors that can occur in addition to the errors normally associated with file
2797 system access:
2798 .sp
2799 .ne 2
2800 .na
2801 \fB\FBE2BIG\fR\fR
2802 .ad
2803 .RS 13n
2804 Data to be returned in a \fBread\fR(2) of the page data file exceeds the size
2805 of the read buffer provided by the caller.
2806 .RE

2808 .sp
2809 .ne 2
2810 .na

```

```

2811 \fB\FBEACCES\fR\fR
2812 .ad
2813 .RS 13n
2814 An attempt was made to examine a process that ran under a different uid than
2815 the controlling process and {\fBPRIV_PROC_OWNER\fR} was not asserted in the
2816 effective set.
2817 .RE

2819 .sp
2820 .ne 2
2821 .na
2822 \fB\FBEAGAIN\fR\fR
2823 .ad
2824 .RS 13n
2825 The traced process has performed a \fBexec\fR(2) of a setuid/setgid object
2826 file or of an object file that it cannot read; all further operations on the
2827 process or lwp file descriptor (except \fBclose\fR(2)) elicit this error.
2828 .RE

2830 .sp
2831 .ne 2
2832 .na
2833 \fB\FBEBUSY\fR\fR
2834 .ad
2835 .RS 13n
2836 \fBPCSTOP\fR, \fBPCDSTOP\fR, \fBPCWSTOP\fR, or \fBPCTWSTOP\fR was applied to a
2837 system process; an exclusive \fBopen\fR(2) was attempted on a \fB/proc\fR file
2838 for a process already open for writing; \fBPCRUN\fR, \fBPCREG\fR,
2839 \fBPCSVADDR\fR, \fBPCSFPPREG\fR, or \fBPCSXREG\fR was applied to a process or
2840 lwp not stopped on an event of interest; an attempt was made to mount
2841 \fB/proc\fR when it was already mounted; \fBPCAGENT\fR was applied to a process
2842 that was not fully stopped or that already had an agent lwp.
2843 .RE

2845 .sp
2846 .ne 2
2847 .na
2848 \fB\FBEINVAL\fR\fR
2849 .ad
2850 .RS 13n
2851 In general, this means that some invalid argument was supplied to a system
2852 call. A non-exhaustive list of conditions eliciting this error includes: a
2853 control message operation code is undefined; an out-of-range signal number was
2854 specified with \fBPCSSIG\fR, \fBPCCKILL\fR, or \fBPCUNKILL\fR; \fBSIGKILL\fR was
2855 specified with \fBPCUNKILL\fR; \fBPCSFPPREG\fR was applied on a system that does
2856 not support floating-point operations; \fBPCSXREG\fR was applied on a system
2857 that does not support extra state registers.
2858 .RE

2860 .sp
2861 .ne 2
2862 .na
2863 \fB\FBEINTR\fR\fR
2864 .ad
2865 .RS 13n
2866 A signal was received by the controlling process while waiting for the traced
2867 process or lwp to stop via \fBPCSTOP\fR, \fBPCWSTOP\fR, or \fBPCTWSTOP\fR.
2868 .RE

2870 .sp
2871 .ne 2
2872 .na
2873 \fB\FBEIO\fR\fR
2874 .ad
2875 .RS 13n
2876 A \fBwrite\fR(2) was attempted at an illegal address in the traced process.

```

2877 .RE

2879 .sp
 2880 .ne 2
 2881 .na
 2882 \fB\FBENOENT\fR\fR
 2883 .ad
 2884 .RS 13n
 2885 The traced process or lwp has terminated after being opened. The basic
 2886 privilege {\fBPRIV_PROC_INFO\fR} is not asserted in the effective set of the
 2887 calling process and the calling process cannot send a signal to the target
 2888 process.
 2889 .RE

2891 .sp
 2892 .ne 2
 2893 .na
 2894 \fB\FBENOMEM\fR\fR
 2895 .ad
 2896 .RS 13n
 2897 The system-imposed limit on the number of page data file descriptors was
 2898 reached on an open of \fB/proc/\fR\fIpid\fR\fB/pagedata\fR; an attempt was made
 2899 with \fBPCWATCH\fR to establish more watched areas than the system can support;
 2900 the \fBPCAGENT\fR operation was issued when the system was out of resources for
 2901 creating lwps.
 2902 .RE

2904 .sp
 2905 .ne 2
 2906 .na
 2907 \fB\FBENOSYS\fR\fR
 2908 .ad
 2909 .RS 13n
 2910 An attempt was made to perform an unsupported operation (such as
 2911 \fBcreat\fR(2), \fBlink\fR(2), or \fBunlink\fR(2)) on an entry in \fB/proc\fR.
 2912 .RE

2914 .sp
 2915 .ne 2
 2916 .na
 2917 \fB\FBEOVERFLOW\fR\fR
 2918 .ad
 2919 .RS 13n
 2920 A 32-bit controlling process attempted to read or write the \fBas\fR file or
 2921 attempted to read the \fBmap\fR, \fBbrmap\fR, or \fBpagedata\fR file of a 64-bit
 2922 target process. A 32-bit controlling process attempted to apply one of the
 2923 control operations \fBPCSRREG\fR, \fBPCSXREG\fR, \fBPCSVADDR\fR, \fBPCWATCH\fR,
 2924 \fBPCAGENT\fR, \fBPCREAD\fR, \fBPCWRITE\fR to a 64-bit target process.
 2925 .RE

2927 .sp
 2928 .ne 2
 2929 .na
 2930 \fB\FBEPERM\fR\fR
 2931 .ad
 2932 .RS 13n
 2933 The process that issued the \fBPCSCRED\fR or \fBPCSCREDX\fR operation did not
 2934 have the {\fBPRIV_PROC_SETID\fR} privilege asserted in its effective set, or
 2935 the process that issued the \fBPCNICE\fR operation did not have the
 2936 {\fBPRIV_PROC_PRIOCNL\fR} in its effective set.
 2937 .sp
 2938 An attempt was made to control a process of which the E, P, and I privilege
 2939 sets were not a subset of the effective set of the controlling process or the
 2940 limit set of the controlling process is not a superset of limit set of the
 2941 controlled process.
 2942 .sp

2943 Any of the uids of the target process are 0 or an attempt was made to change
 2944 any of the uids to 0 using PCSCRED and the security policy imposed additional
 2945 restrictions. See \fBprivileges\fR(5).
 2946 .RE

2948 .SH NOTES
 2993 .sp
 2949 .LP
 2950 Descriptions of structures in this document include only interesting structure
 2951 elements, not filler and padding fields, and may show elements out of order for
 2952 descriptive clarity. The actual structure definitions are contained in
 2953 \fB<procfs.h>\fR&.
 2954 .SH BUGS
 3000 .sp
 2955 .LP
 2956 Because the old \fBioctl\fR(2)-based version of \fB/proc\fR is currently
 2957 supported for binary compatibility with old applications, the top-level
 2958 directory for a process, \fB/proc/\fR\fIpid\fR, is not world-readable, but it
 2959 is world-searchable. Thus, anyone can open \fB/proc/\fR\fIpid\fR\fB/psinfo\fR
 2960 even though \fBls\fR(1) applied to \fB/proc/\fR\fIpid\fR will fail for anyone
 2961 but the owner or an appropriately privileged process. Support for the old
 2962 \fBioctl\fR(2)-based version of \fB/proc\fR will be dropped in a future
 2963 release, at which time the top-level directory for a process will be made
 2964 world-readable.
 2965 .sp
 2966 .LP
 2967 On SPARC based machines, the types \fBgregset_t\fR and \fBfpregset_t\fR defined
 2968 in <\fBsys/regset.h\fR> are similar to but not the same as the types
 2969 \fBprgregset_t\fR and \fBprfpregset_t\fR defined in <\fBprocfs.h\fR>.

```

*****
4213 Thu Jun 30 22:02:50 2016
new/usr/src/man/man5/Makefile
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
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
9 # at http://www.illumos.org/license/CDDL.
10 #
11 #
12 #
13 # Copyright 2011, Richard Lowe
14 # Copyright (c) 2012 by Delphix. All rights reserved.
15 # Copyright 2014 Nexenta Systems, Inc.
16 # Copyright 2014 Garrett D'Amore <garrett@damore.org>
17 # Copyright (c) 2015, Joyent, Inc. All rights reserved.
18 #
19 #
20 include      $(SRC)/Makefile.master
21 #
22 MANSECT=     5
23 #
24 MANFILES=    Intro.5           //
25              acl.5             //
26              ad.5              //
27              ascii.5          //
28              attributes.5     //
29              audit_binfile.5  //
30              audit_remote.5   //
31              audit_syslog.5   //
32              brands.5         //
33              byteorder.5      //
34              cancellation.5   //
35              charmap.5        //
36              condition.5      //
37              crypt_bsdbf.5    //
38              crypt_bsdmd5.5   //
39              crypt_sha256.5   //
40              crypt_sha512.5   //
41              crypt_sunmd5.5   //
42              crypt_unix.5     //
43              device_clean.5   //
44              dhcp.5           //
45              environ.5        //
46              epoll.5          //
47              eqn.5            //
48              eqnchar.5        //
49              eventfd.5        //
50              extendedFILE.5   //
51              filesystem.5     //
52              fnmatch.5        //
53              formats.5        //
54              fsattr.5         //
55              grub.5           //
56              gss_auth_rules.5 //
57              hal.5            //
58              iconv.5          //

```

```

59              iconv_unicode.5 //
60              ieee802.3.5     //
61              ieee802.11.5    //
62              ipfilter.5      //
63              isalist.5       //
64              kerberos.5      //
65              krb5_auth_rules.5 //
66              krb5envvar.5    //
67              largefile.5     //
68              lf64.5          //
69              lfcompile.5     //
70              lfcompile64.5   //
71              locale.5        //
72              man.5           //
73              mandoc_char.5   //
74              mandoc_roff.5   //
75              mdoc.5          //
76              me.5            //
77              mech_spnego.5   //
78              mm.5            //
79              ms.5            //
80              mutex.5         //
81              nfssec.5        //
82              pam_allow.5     //
83              pam_authtok_check.5 //
84              pam_authtok_get.5 //
85              pam_authtok_store.5 //
86              pam_deny.5      //
87              pam_dhkeys.5    //
88              pam_dial_auth.5  //
89              pam_krb5.5      //
90              pam_krb5_migrate.5 //
91              pam_ldap.5      //
92              pam_list.5      //
93              pam_passwd_auth.5 //
94              pam_rhosts_auth.5 //
95              pam_roles.5     //
96              pam_sample.5    //
97              pam_smb_passwd.5 //
98              pam_smbfs_login.5 //
99              pam_timestamp.5 //
100             pam_tsol_account.5 //
101             pam_unix_account.5 //
102             pam_unix_auth.5   //
103             pam_unix_cred.5   //
104             pam_unix_session.5 //
105             pkcs11_kernel.5   //
106             pkcs11_softtoken.5 //
107             pkcs11_tpm.5     //
108             privileges.5      //
109             prof.5           //
110             rbac.5           //
111             regex.5          //
112             regexp.5         //
113             resource_controls.5 //
114             security_flags.5 //
115             #endif /* ! codereview */
116             smf.5             //
117             smf_bootstrap.5   //
118             smf_method.5      //
119             smf_restarter.5   //
120             smf_security.5    //
121             smf_template.5    //
122             standards.5      //
123             sticky.5         //
124             tbl.5            //

```

```

125         tecla.5           \
126         term.5           \
127         threads.5       \
128         timerfd.5       \
129         trusted_extensions.5 \
130         vgrindefs.5     \
131         zones.5        \
132         zpool-features.5

134 MANLINKS=  ANSI.5       \
135             C++.5        \
136             C.5          \
137             CSI.5        \
138             ISO.5        \
139             MT-Level.5   \
140             POSIX.1.5    \
141             POSIX.2.5    \
142             POSIX.5      \
143             RBAC.5       \
144             SUS.5        \
145             SUSv2.5      \
146             SUSv3.5     \
147             SVID.5       \
148             SVID3.5      \
149             XNS.5        \
150             XNS4.5       \
151             XNS5.5       \
152             XPG.5        \
153             XPG3.5       \
154             XPG4.5       \
155             XPG4v2.5     \
156             advance.5    \
157             architecture.5 \
158             availability.5 \
159             compile.5    \
160             endian.5     \
161             intro.5      \
162             pthreads.5   \
163             stability.5  \
164             standard.5   \
165             step.5       \
166             teclarc.5

168 intro.5      := LINKSRC = Intro.5

170 CSI.5        := LINKSRC = attributes.5
171 MT-Level.5   := LINKSRC = attributes.5
172 architecture.5 := LINKSRC = attributes.5
173 availability.5 := LINKSRC = attributes.5
174 stability.5   := LINKSRC = attributes.5
175 standard.5   := LINKSRC = attributes.5

177 endian.5     := LINKSRC = byteorder.5

179 RBAC.5       := LINKSRC = rbac.5

181 advance.5    := LINKSRC = regexp.5
182 compile.5    := LINKSRC = regexp.5
183 step.5       := LINKSRC = regexp.5

185 ANSI.5       := LINKSRC = standards.5
186 C++.5        := LINKSRC = standards.5
187 C.5          := LINKSRC = standards.5
188 ISO.5        := LINKSRC = standards.5
189 POSIX.1.5    := LINKSRC = standards.5
190 POSIX.2.5    := LINKSRC = standards.5

```

```

191 POSIX.5      := LINKSRC = standards.5
192 SUS.5        := LINKSRC = standards.5
193 SUSv2.5      := LINKSRC = standards.5
194 SUSv3.5      := LINKSRC = standards.5
195 SVID.5       := LINKSRC = standards.5
196 SVID3.5     := LINKSRC = standards.5
197 XNS.5        := LINKSRC = standards.5
198 XNS4.5       := LINKSRC = standards.5
199 XNS5.5       := LINKSRC = standards.5
200 XPG.5        := LINKSRC = standards.5
201 XPG3.5       := LINKSRC = standards.5
202 XPG4.5       := LINKSRC = standards.5
203 XPG4v2.5    := LINKSRC = standards.5

205 teclarc.5   := LINKSRC = tecla.5

207 pthreads.5  := LINKSRC = threads.5

209 .KEEP_STATE:

211 include      $(SRC)/man/Makefile.man

213 install:    $(ROOTMANFILES) $(ROOTMANLINKS)

```

```

*****
34007 Thu Jun 30 22:02:50 2016
new/usr/src/man/man5/privileges.5
Code review comments from jeffpc
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
1 \" te
2 .\\ Copyright (c) 2009, Sun Microsystems, Inc. All Rights Reserved.
3 .\\ Copyright 2015, Joyent, Inc. All Rights Reserved.
4 .\\ The contents of this file are subject to the terms of the Common Development
5 .\\ See the License for the specific language governing permissions and limitat
6 .\\ the fields enclosed by brackets \"[]\" replaced with your own identifying info
7 .TH PRIVILEGES 5 \"Jun 6, 2016\"
8 .TH PRIVILEGES 5 \"Oct 30, 2015\"
9 .SH NAME
9 privileges \\- process privilege model
10 .SH DESCRIPTION
11 .LP
12 Solaris software implements a set of privileges that provide fine-grained
13 control over the actions of processes. The possession of a certain privilege
14 allows a process to perform a specific set of restricted operations.
15 .sp
16 .LP
17 The change to a primarily privilege-based security model in the Solaris
18 operating system gives developers an opportunity to restrict processes to those
19 privileged operations actually needed instead of all (super-user) or no
20 privileges (non-zero UIDs). Additionally, a set of previously unrestricted
21 operations now requires a privilege; these privileges are dubbed the \"basic\"
22 privileges and are by default given to all processes.
23 .sp
24 .LP
25 Taken together, all defined privileges with the exception of the \"basic\"
26 privileges compose the set of privileges that are traditionally associated with
27 the root user. The \"basic\" privileges are \"privileges\" unprivileged processes
28 were accustomed to having.
29 .sp
30 .LP
31 The defined privileges are:
32 .sp
33 .ne 2
34 .na
35 \\fb\\fbPRIV_CONTRACT_EVENT\\fr\\fr
36 .ad
37 .sp .6
38 .RS 4n
39 Allow a process to request reliable delivery of events to an event endpoint.
40 .sp
41 Allow a process to include events in the critical event set term of a template
42 which could be generated in volume by the user.
43 .RE

45 .sp
46 .ne 2
47 .na
48 \\fb\\fbPRIV_CONTRACT_IDENTITY\\fr\\fr
49 .ad
50 .sp .6
51 .RS 4n
52 Allows a process to set the service FMRI value of a process contract template.
53 .RE

55 .sp
56 .ne 2

```

```

57 .na
58 \\fb\\fbPRIV_CONTRACT_OBSERVER\\fr\\fr
59 .ad
60 .sp .6
61 .RS 4n
62 Allow a process to observe contract events generated by contracts created and
63 owned by users other than the process's effective user ID.
64 .sp
65 Allow a process to open contract event endpoints belonging to contracts created
66 and owned by users other than the process's effective user ID.
67 .RE

69 .sp
70 .ne 2
71 .na
72 \\fb\\fbPRIV_CPC_CPU\\fr\\fr
73 .ad
74 .sp .6
75 .RS 4n
76 Allow a process to access per-CPU hardware performance counters.
77 .RE

79 .sp
80 .ne 2
81 .na
82 \\fb\\fbPRIV_DTRACE_KERNEL\\fr\\fr
83 .ad
84 .sp .6
85 .RS 4n
86 Allow DTrace kernel-level tracing.
87 .RE

89 .sp
90 .ne 2
91 .na
92 \\fb\\fbPRIV_DTRACE_PROC\\fr\\fr
93 .ad
94 .sp .6
95 .RS 4n
96 Allow DTrace process-level tracing. Allow process-level tracing probes to be
97 placed and enabled in processes to which the user has permissions.
98 .RE

100 .sp
101 .ne 2
102 .na
103 \\fb\\fbPRIV_DTRACE_USER\\fr\\fr
104 .ad
105 .sp .6
106 .RS 4n
107 Allow DTrace user-level tracing. Allow use of the syscall and profile DTrace
108 providers to examine processes to which the user has permissions.
109 .RE

111 .sp
112 .ne 2
113 .na
114 \\fb\\fbPRIV_FILE_CHOWN\\fr\\fr
115 .ad
116 .sp .6
117 .RS 4n
118 Allow a process to change a file's owner user ID. Allow a process to change a
119 file's group ID to one other than the process's effective group ID or one of
120 the process's supplemental group IDs.
121 .RE

```

```

123 .sp
124 .ne 2
125 .na
126 \fb\fbPRIV_FILE_CHOWN_SELF\fr\fr
127 .ad
128 .sp .6
129 .RS 4n
130 Allow a process to give away its files. A process with this privilege runs as
131 if {\fb_POSIX_CHOWN_RESTRICTED\fr} is not in effect.
132 .RE

134 .sp
135 .ne 2
136 .na
137 \fb\fbPRIV_FILE_DAC_EXECUTE\fr\fr
138 .ad
139 .sp .6
140 .RS 4n
141 Allow a process to execute an executable file whose permission bits or ACL
142 would otherwise disallow the process execute permission.
143 .RE

145 .sp
146 .ne 2
147 .na
148 \fb\fbPRIV_FILE_DAC_READ\fr\fr
149 .ad
150 .sp .6
151 .RS 4n
152 Allow a process to read a file or directory whose permission bits or ACL would
153 otherwise disallow the process read permission.
154 .RE

156 .sp
157 .ne 2
158 .na
159 \fb\fbPRIV_FILE_DAC_SEARCH\fr\fr
160 .ad
161 .sp .6
162 .RS 4n
163 Allow a process to search a directory whose permission bits or ACL would not
164 otherwise allow the process search permission.
165 .RE

167 .sp
168 .ne 2
169 .na
170 \fb\fbPRIV_FILE_DAC_WRITE\fr\fr
171 .ad
172 .sp .6
173 .RS 4n
174 Allow a process to write a file or directory whose permission bits or ACL do
175 not allow the process write permission. All privileges are required to write
176 files owned by UID 0 in the absence of an effective UID of 0.
177 .RE

179 .sp
180 .ne 2
181 .na
182 \fb\fbPRIV_FILE_DOWNGRADE_SL\fr\fr
183 .ad
184 .sp .6
185 .RS 4n
186 Allow a process to set the sensitivity label of a file or directory to a
187 sensitivity label that does not dominate the existing sensitivity label.
188 .sp

```

```

189 This privilege is interpreted only if the system is configured with Trusted
190 Extensions.
191 .RE

193 .sp
194 .ne 2
195 .na
196 \fb\fbPRIV_FILE_FLAG_SET\fr\fr
197 .ad
198 .sp .6
199 .RS 4n
200 Allows a process to set immutable, nounlink or appendonly file attributes.
201 .RE

203 .sp
204 .ne 2
205 .na
206 \fb\fbPRIV_FILE_LINK_ANY\fr\fr
207 .ad
208 .sp .6
209 .RS 4n
210 Allow a process to create hardlinks to files owned by a UID different from the
211 process's effective UID.
212 .RE

214 .sp
215 .ne 2
216 .na
217 \fb\fbPRIV_FILE_OWNER\fr\fr
218 .ad
219 .sp .6
220 .RS 4n
221 Allow a process that is not the owner of a file to modify that file's access
222 and modification times. Allow a process that is not the owner of a directory to
223 modify that directory's access and modification times. Allow a process that is
224 not the owner of a file or directory to remove or rename a file or directory
225 whose parent directory has the "save text image after execution" (sticky) bit
226 set. Allow a process that is not the owner of a file to mount a \fbnamefs\fr
227 upon that file. Allow a process that is not the owner of a file or directory to
228 modify that file's or directory's permission bits or ACL.
229 .RE

231 .sp
232 .ne 2
233 .na
234 \fb\fbPRIV_FILE_READ\fr\fr
235 .ad
236 .sp .6
237 .RS 4n
238 Allow a process to open objects in the filesystem for reading. This
239 privilege is not necessary to read from an already open file which was opened
240 before dropping the \fbPRIV_FILE_READ\fr privilege.
241 .RE

243 .sp
244 .ne 2
245 .na
246 \fb\fbPRIV_FILE_SETID\fr\fr
247 .ad
248 .sp .6
249 .RS 4n
250 Allow a process to change the ownership of a file or write to a file without
251 the set-user-ID and set-group-ID bits being cleared. Allow a process to set the
252 set-group-ID bit on a file or directory whose group is not the process's
253 effective group or one of the process's supplemental groups. Allow a process to
254 set the set-user-ID bit on a file with different ownership in the presence of

```

```

255 \fbPRIV_FILE_OWNER\fr. Additional restrictions apply when creating or modifying
256 a setuid 0 file.
257 .RE

259 .sp
260 .ne 2
261 .na
262 \fb\fbPRIV_FILE_UPGRADE_SL\fr\fr
263 .ad
264 .sp .6
265 .RS 4n
266 Allow a process to set the sensitivity label of a file or directory to a
267 sensitivity label that dominates the existing sensitivity label.
268 .sp
269 This privilege is interpreted only if the system is configured with Trusted
270 Extensions.
271 .RE

273 .sp
274 .ne 2
275 .na
276 \fb\fbPRIV_FILE_WRITE\fr\fr
277 .ad
278 .sp .6
279 .RS 4n
280 Allow a process to open objects in the filesystem for writing, or otherwise
281 modify them. This privilege is not necessary to write to an already open file
282 which was opened before dropping the \fbPRIV_FILE_WRITE\fr privilege.
283 .RE

285 .sp
286 .ne 2
287 .na
288 \fb\fbPRIV_GRAPHICS_ACCESS\fr\fr
289 .ad
290 .sp .6
291 .RS 4n
292 Allow a process to make privileged ioctls to graphics devices. Typically only
293 an xserver process needs to have this privilege. A process with this privilege
294 is also allowed to perform privileged graphics device mappings.
295 .RE

297 .sp
298 .ne 2
299 .na
300 \fb\fbPRIV_GRAPHICS_MAP\fr\fr
301 .ad
302 .sp .6
303 .RS 4n
304 Allow a process to perform privileged mappings through a graphics device.
305 .RE

307 .sp
308 .ne 2
309 .na
310 \fb\fbPRIV_IPC_DAC_READ\fr\fr
311 .ad
312 .sp .6
313 .RS 4n
314 Allow a process to read a System V IPC Message Queue, Semaphore Set, or Shared
315 Memory Segment whose permission bits would not otherwise allow the process read
316 permission.
317 .RE

319 .sp
320 .ne 2

```

```

321 .na
322 \fb\fbPRIV_IPC_DAC_WRITE\fr\fr
323 .ad
324 .sp .6
325 .RS 4n
326 Allow a process to write a System V IPC Message Queue, Semaphore Set, or Shared
327 Memory Segment whose permission bits would not otherwise allow the process
328 write permission.
329 .RE

331 .sp
332 .ne 2
333 .na
334 \fb\fbPRIV_IPC_OWNER\fr\fr
335 .ad
336 .sp .6
337 .RS 4n
338 Allow a process that is not the owner of a System V IPC Message Queue,
339 Semaphore Set, or Shared Memory Segment to remove, change ownership of, or
340 change permission bits of the Message Queue, Semaphore Set, or Shared Memory
341 Segment.
342 .RE

344 .sp
345 .ne 2
346 .na
347 \fb\fbPRIV_NET_ACCESS\fr\fr
348 .ad
349 .sp .6
350 .RS 4n
351 Allow a process to open a TCP, UDP, SDP, or SCTP network endpoint. This
352 privilege is not necessary to communicate using an existing endpoint already
353 opened before dropping the \fbPRIV_NET_ACCESS\fr privilege.
354 .RE

356 .sp
357 .ne 2
358 .na
359 \fb\fbPRIV_NET_BINDMLP\fr\fr
360 .ad
361 .sp .6
362 .RS 4n
363 Allow a process to bind to a port that is configured as a multi-level port
364 (MLP) for the process's zone. This privilege applies to both shared address and
365 zone-specific address MLPs. See \fb\fbB4\fr from the Trusted
366 Extensions manual pages for information on configuring MLP ports.
367 .sp
368 This privilege is interpreted only if the system is configured with Trusted
369 Extensions.
370 .RE

372 .sp
373 .ne 2
374 .na
375 \fb\fbPRIV_NET_ICMPACCESS\fr\fr
376 .ad
377 .sp .6
378 .RS 4n
379 Allow a process to send and receive ICMP packets.
380 .RE

382 .sp
383 .ne 2
384 .na
385 \fb\fbPRIV_NET_MAC_AWARE\fr\fr
386 .ad

```

```

387 .sp .6
388 .RS 4n
389 Allow a process to set the \fBNET_MAC_AWARE\fR process flag by using
390 \fBsetpflags\fR(2). This privilege also allows a process to set the
391 \fBBSO_MAC_EXEMPT\fR socket option by using \fBsetsockopt\fR(3SOCKET). The
392 \fBNET_MAC_AWARE\fR process flag and the \fBBSO_MAC_EXEMPT\fR socket option both
393 allow a local process to communicate with an unlabeled peer if the local
394 process's label dominates the peer's default label, or if the local process
395 runs in the global zone.
396 .sp
397 This privilege is interpreted only if the system is configured with Trusted
398 Extensions.
399 .RE

401 .sp
402 .ne 2
403 .na
404 \fB\FBPRIV_NET_MAC_IMPLICIT\fR
405 .ad
406 .sp .6
407 .RS 4n
408 Allow a process to set \fBBSO_MAC_IMPLICIT\fR option by using
409 \fBsetsockopt\fR(3SOCKET). This allows a privileged process to transmit
410 implicitly-labeled packets to a peer.
411 .sp
412 This privilege is interpreted only if the system is configured with
413 Trusted Extensions.
414 .RE

416 .sp
417 .ne 2
418 .na
419 \fB\FBPRIV_NET_OBSERVABILITY\fR
420 .ad
421 .sp .6
422 .RS 4n
423 Allow a process to open a device for just receiving network traffic, sending
424 traffic is disallowed.
425 .RE

427 .sp
428 .ne 2
429 .na
430 \fB\FBPRIV_NET_PRIVADDR\fR
431 .ad
432 .sp .6
433 .RS 4n
434 Allow a process to bind to a privileged port number. The privilege port numbers
435 are 1-1023 (the traditional UNIX privileged ports) as well as those ports
436 marked as "\fBudp/tcp_extra_priv_ports\fR" with the exception of the ports
437 reserved for use by NFS and SMB.
438 .RE

440 .sp
441 .ne 2
442 .na
443 \fB\FBPRIV_NET_RAWACCESS\fR
444 .ad
445 .sp .6
446 .RS 4n
447 Allow a process to have direct access to the network layer.
448 .RE

450 .sp
451 .ne 2
452 .na

```

```

453 \fB\FBPRIV_PROC_AUDIT\fR
454 .ad
455 .sp .6
456 .RS 4n
457 Allow a process to generate audit records. Allow a process to get its own audit
458 pre-selection information.
459 .RE

461 .sp
462 .ne 2
463 .na
464 \fB\FBPRIV_PROC_CHROOT\fR
465 .ad
466 .sp .6
467 .RS 4n
468 Allow a process to change its root directory.
469 .RE

471 .sp
472 .ne 2
473 .na
474 \fB\FBPRIV_PROC_CLOCK_HIGHRES\fR
475 .ad
476 .sp .6
477 .RS 4n
478 Allow a process to use high resolution timers.
479 .RE

481 .sp
482 .ne 2
483 .na
484 \fB\FBPRIV_PROC_EXEC\fR
485 .ad
486 .sp .6
487 .RS 4n
488 Allow a process to call \fBexec\fR(2).
489 .RE

491 .sp
492 .ne 2
493 .na
494 \fB\FBPRIV_PROC_FORK\fR
495 .ad
496 .sp .6
497 .RS 4n
498 Allow a process to call \fBfork\fR(2), \fBfork1\fR(2), or \fBvfork\fR(2).
499 .RE

501 .sp
502 .ne 2
503 .na
504 \fB\FBPRIV_PROC_INFO\fR
505 .ad
506 .sp .6
507 .RS 4n
508 Allow a process to examine the status of processes other than those to which it
509 can send signals. Processes that cannot be examined cannot be seen in
510 \fB/proc\fR and appear not to exist.
511 .RE

513 .sp
514 .ne 2
515 .na
516 \fB\FBPRIV_PROC_LOCK_MEMORY\fR
517 .ad
518 .sp .6

```



```

519 .RS 4n
520 Allow a process to lock pages in physical memory.
521 .RE

523 .sp
524 .ne 2
525 .na
526 \fB\fBPRIV_PROC_MEMINFO\fR\fR
527 .ad
528 .sp .6
529 .RS 4n
530 Allow a process to access physical memory information.
531 .RE

533 .sp
534 .ne 2
535 .na
536 \fB\fBPRIV_PROC_OWNER\fR\fR
537 .ad
538 .sp .6
539 .RS 4n
540 Allow a process to send signals to other processes and inspect and modify the
541 process state in other processes, regardless of ownership. When modifying
542 another process, additional restrictions apply: the effective privilege set of
543 the attaching process must be a superset of the target process's effective,
544 permitted, and inheritable sets; the limit set must be a superset of the
545 target's limit set; if the target process has any UID set to 0 all privilege
546 must be asserted unless the effective UID is 0. Allow a process to bind
547 arbitrary processes to CPUs.
548 .RE

550 .sp
551 .ne 2
552 .na
553 \fB\fBPRIV_PROC_PRIOUP\fR\fR
554 .ad
555 .sp .6
556 .RS 4n
557 Allow a process to elevate its priority above its current level.
558 .RE

560 .sp
561 .ne 2
562 .na
563 \fB\fBPRIV_PROC_PRIOCNTL\fR\fR
564 .ad
565 .sp .6
566 .RS 4n
567 Allows all that PRIV_PROC_PRIOUP allows.
568 Allow a process to change its scheduling class to any scheduling class,
569 including the RT class.
570 .RE

572 .sp
573 .ne 2
574 .na
575 \fB\PRIV_PROC_SECFLAGS\fR
576 .ad
577 .sp .6
578 .RS 4n
579 Allow a process to manipulate the secflags of processes (subject to,
580 additionally, the ability to signal that process).
581 .RE

583 .sp
584 .ne 2

```

```

585 .na
586 #endif /* ! codereview */
587 \fB\fBPRIV_PROC_SESSION\fR\fR
588 .ad
589 .sp .6
590 .RS 4n
591 Allow a process to send signals or trace processes outside its session.
592 .RE

594 .sp
595 .ne 2
596 .na
597 \fB\fBPRIV_PROC_SETID\fR\fR
598 .ad
599 .sp .6
600 .RS 4n
601 Allow a process to set its UIDs at will, assuming UID 0 requires all privileges
602 to be asserted.
603 .RE

605 .sp
606 .ne 2
607 .na
608 \fB\fBPRIV_PROC_TASKID\fR\fR
609 .ad
610 .sp .6
611 .RS 4n
612 Allow a process to assign a new task ID to the calling process.
613 .RE

615 .sp
616 .ne 2
617 .na
618 \fB\fBPRIV_PROC_ZONE\fR\fR
619 .ad
620 .sp .6
621 .RS 4n
622 Allow a process to trace or send signals to processes in other zones. See
623 \fBzones\fR(5).
624 .RE

626 .sp
627 .ne 2
628 .na
629 \fB\fBPRIV_SYS_ACCT\fR\fR
630 .ad
631 .sp .6
632 .RS 4n
633 Allow a process to enable and disable and manage accounting through
634 \fBacct\fR(2).
635 .RE

637 .sp
638 .ne 2
639 .na
640 \fB\fBPRIV_SYS_ADMIN\fR\fR
641 .ad
642 .sp .6
643 .RS 4n
644 Allow a process to perform system administration tasks such as setting node and
645 domain name and specifying \fBcoreadm\fR(1M) and \fBnscd\fR(1M) settings
646 .RE

648 .sp
649 .ne 2
650 .na

```

```

651 \fb\fbPRIV_SYS_AUDIT\fr\fr
652 .ad
653 .sp .6
654 .RS 4n
655 Allow a process to start the (kernel) audit daemon. Allow a process to view and
656 set audit state (audit user ID, audit terminal ID, audit sessions ID, audit
657 pre-selection mask). Allow a process to turn off and on auditing. Allow a
658 process to configure the audit parameters (cache and queue sizes, event to
659 class mappings, and policy options).
660 .RE

662 .sp
663 .ne 2
664 .na
665 \fb\fbPRIV_SYS_CONFIG\fr\fr
666 .ad
667 .sp .6
668 .RS 4n
669 Allow a process to perform various system configuration tasks. Allow
670 filesystem-specific administrative procedures, such as filesystem configuration
671 ioctls, quota calls, creation and deletion of snapshots, and manipulating the
672 PCFS bootsector.
673 .RE

675 .sp
676 .ne 2
677 .na
678 \fb\fbPRIV_SYS_DEVICES\fr\fr
679 .ad
680 .sp .6
681 .RS 4n
682 Allow a process to create device special files. Allow a process to successfully
683 call a kernel module that calls the kernel \fbDrv_priv\fr(9F) function to check
684 for allowed access. Allow a process to open the real console device directly.
685 Allow a process to open devices that have been exclusively opened.
686 .RE

688 .sp
689 .ne 2
690 .na
691 \fb\fbPRIV_SYS_DL_CONFIG\fr\fr
692 .ad
693 .sp .6
694 .RS 4n
695 Allow a process to configure a system's datalink interfaces.
696 .RE

698 .sp
699 .ne 2
700 .na
701 \fb\fbPRIV_SYS_IP_CONFIG\fr\fr
702 .ad
703 .sp .6
704 .RS 4n
705 Allow a process to configure a system's IP interfaces and routes. Allow a
706 process to configure network parameters for \fbTCP/IP\fr using \fbNdd\fr. Allow
707 a process access to otherwise restricted \fbTCP/IP\fr information using
708 \fbNdd\fr. Allow a process to configure \fbIPsec\fr. Allow a process to pop
709 anchored \fbSTREAM\frs modules with matching \fbzoneid\fr.
710 .RE

712 .sp
713 .ne 2
714 .na
715 \fb\fbPRIV_SYS_IPC_CONFIG\fr\fr
716 .ad

```

```

717 .sp .6
718 .RS 4n
719 Allow a process to increase the size of a System V IPC Message Queue buffer.
720 .RE

722 .sp
723 .ne 2
724 .na
725 \fb\fbPRIV_SYS_IPTUN_CONFIG\fr\fr
726 .ad
727 .sp .6
728 .RS 4n
729 Allow a process to configure IP tunnel links.
730 .RE

732 .sp
733 .ne 2
734 .na
735 \fb\fbPRIV_SYS_LINKDIR\fr\fr
736 .ad
737 .sp .6
738 .RS 4n
739 Allow a process to unlink and link directories.
740 .RE

742 .sp
743 .ne 2
744 .na
745 \fb\fbPRIV_SYS_MOUNT\fr\fr
746 .ad
747 .sp .6
748 .RS 4n
749 Allow a process to mount and unmount filesystems that would otherwise be
750 restricted (that is, most filesystems except \fbNamefs\fr). Allow a process to
751 add and remove swap devices.
752 .RE

754 .sp
755 .ne 2
756 .na
757 \fb\fbPRIV_SYS_NET_CONFIG\fr\fr
758 .ad
759 .sp .6
760 .RS 4n
761 Allow a process to do all that \fbPRIV_SYS_IP_CONFIG\fr,
762 \fbPRIV_SYS_DL_CONFIG\fr, and \fbPRIV_SYS_PPP_CONFIG\fr allow, plus the
763 following: use the \fbRpmcmod\fr STREAMS module and insert/remove STREAMS
764 modules on locations other than the top of the module stack.
765 .RE

767 .sp
768 .ne 2
769 .na
770 \fb\fbPRIV_SYS_NFS\fr\fr
771 .ad
772 .sp .6
773 .RS 4n
774 Allow a process to provide NFS service: start NFS kernel threads, perform NFS
775 locking operations, bind to NFS reserved ports: ports 2049 (\fbNfs\fr) and port
776 4045 (\fbBlockd\fr).
777 .RE

779 .sp
780 .ne 2
781 .na
782 \fb\fbPRIV_SYS_PPP_CONFIG\fr\fr

```

```

783 .ad
784 .sp .6
785 .RS 4n
786 Allow a process to create, configure, and destroy PPP instances with pppd(1M)
787 \fBpppd\fR(1M) and control PPPoE plumbing with \fBspptun\fR(1M)sppptun(1M).
788 This privilege is granted by default to exclusive IP stack instance zones.
789 .RE

791 .sp
792 .ne 2
793 .na
794 \fB\FBPRIV_SYS_RES_BIND\fR\fR
795 .ad
796 .sp .6
797 .RS 4n
798 Allows a process to bind processes to processor sets.
799 .RE

801 .sp
802 .ne 2
803 .na
804 \fB\FBPRIV_SYS_RES_CONFIG\fR\fR
805 .ad
806 .sp .6
807 .RS 4n
808 Allows all that PRIV_SYS_RES_BIND allows.
809 Allow a process to create and delete processor sets, assign CPUs to processor
810 sets and override the \fBSET_NOESCAPE\fR property. Allow a process to change
811 the operational status of CPUs in the system using \fBp_online\fR(2). Allow a
812 process to configure filesystem quotas. Allow a process to configure resource
813 pools and bind processes to pools.
814 .RE

816 .sp
817 .ne 2
818 .na
819 \fB\FBPRIV_SYS_RESOURCE\fR\fR
820 .ad
821 .sp .6
822 .RS 4n
823 Allow a process to exceed the resource limits imposed on it by
824 \fBsetrlimit\fR(2) and \fBsetrctl\fR(2).
825 .RE

827 .sp
828 .ne 2
829 .na
830 \fB\FBPRIV_SYS_SMB\fR\fR
831 .ad
832 .sp .6
833 .RS 4n
834 Allow a process to provide NetBIOS or SMB services: start SMB kernel threads or
835 bind to NetBIOS or SMB reserved ports: ports 137, 138, 139 (NetBIOS) and 445
836 (SMB).
837 .RE

839 .sp
840 .ne 2
841 .na
842 \fB\FBPRIV_SYS_SUSER_COMPAT\fR\fR
843 .ad
844 .sp .6
845 .RS 4n
846 Allow a process to successfully call a third party loadable module that calls
847 the kernel \fBbsuser()\fR function to check for allowed access. This privilege
848 exists only for third party loadable module compatibility and is not used by

```

```

849 Solaris proper.
850 .RE

852 .sp
853 .ne 2
854 .na
855 \fB\FBPRIV_SYS_TIME\fR\fR
856 .ad
857 .sp .6
858 .RS 4n
859 Allow a process to manipulate system time using any of the appropriate system
860 calls: \fBstime\fR(2), \fBbadjtime\fR(2), and \fBntp_adjtime\fR(2).
861 .RE

863 .sp
864 .ne 2
865 .na
866 \fB\FBPRIV_SYS_TRANS_LABEL\fR\fR
867 .ad
868 .sp .6
869 .RS 4n
870 Allow a process to translate labels that are not dominated by the process's
871 sensitivity label to and from an external string form.
872 .sp
873 This privilege is interpreted only if the system is configured with Trusted
874 Extensions.
875 .RE

877 .sp
878 .ne 2
879 .na
880 \fB\FBPRIV_VIRT_MANAGE\fR\fR
881 .ad
882 .sp .6
883 .RS 4n
884 Allows a process to manage virtualized environments such as \fBxvm\fR(5).
885 .RE

887 .sp
888 .ne 2
889 .na
890 \fB\FBPRIV_WIN_COLORMAP\fR\fR
891 .ad
892 .sp .6
893 .RS 4n
894 Allow a process to override colormap restrictions.
895 .sp
896 Allow a process to install or remove colormaps.
897 .sp
898 Allow a process to retrieve colormap cell entries allocated by other processes.
899 .sp
900 This privilege is interpreted only if the system is configured with Trusted
901 Extensions.
902 .RE

904 .sp
905 .ne 2
906 .na
907 \fB\FBPRIV_WIN_CONFIG\fR\fR
908 .ad
909 .sp .6
910 .RS 4n
911 Allow a process to configure or destroy resources that are permanently retained
912 by the X server.
913 .sp
914 Allow a process to use SetScreenSaver to set the screen saver timeout value

```

```

915 .sp
916 Allow a process to use ChangeHosts to modify the display access control list.
917 .sp
918 Allow a process to use GrabServer.
919 .sp
920 Allow a process to use the SetCloseDownMode request that can retain window,
921 pixmap, colormap, property, cursor, font, or graphic context resources.
922 .sp
923 This privilege is interpreted only if the system is configured with Trusted
924 Extensions.
925 .RE

927 .sp
928 .ne 2
929 .na
930 \fb\fbPRIV_WIN_DAC_READ\fr\fr
931 .ad
932 .sp .6
933 .RS 4n
934 Allow a process to read from a window resource that it does not own (has a
935 different user ID).
936 .sp
937 This privilege is interpreted only if the system is configured with Trusted
938 Extensions.
939 .RE

941 .sp
942 .ne 2
943 .na
944 \fb\fbPRIV_WIN_DAC_WRITE\fr\fr
945 .ad
946 .sp .6
947 .RS 4n
948 Allow a process to write to or create a window resource that it does not own
949 (has a different user ID). A newly created window property is created with the
950 window's user ID.
951 .sp
952 This privilege is interpreted only if the system is configured with Trusted
953 Extensions.
954 .RE

956 .sp
957 .ne 2
958 .na
959 \fb\fbPRIV_WIN_DEVICES\fr\fr
960 .ad
961 .sp .6
962 .RS 4n
963 Allow a process to perform operations on window input devices.
964 .sp
965 Allow a process to get and set keyboard and pointer controls.
966 .sp
967 Allow a process to modify pointer button and key mappings.
968 .sp
969 This privilege is interpreted only if the system is configured with Trusted
970 Extensions.
971 .RE

973 .sp
974 .ne 2
975 .na
976 \fb\fbPRIV_WIN_DGA\fr\fr
977 .ad
978 .sp .6
979 .RS 4n
980 Allow a process to use the direct graphics access (DGA) X protocol extensions.

```

```

981 Direct process access to the frame buffer is still required. Thus the process
982 must have MAC and DAC privileges that allow access to the frame buffer, or the
983 frame buffer must be allocated to the process.
984 .sp
985 This privilege is interpreted only if the system is configured with Trusted
986 Extensions.
987 .RE

989 .sp
990 .ne 2
991 .na
992 \fb\fbPRIV_WIN_DOWNGRADE_SL\fr\fr
993 .ad
994 .sp .6
995 .RS 4n
996 Allow a process to set the sensitivity label of a window resource to a
997 sensitivity label that does not dominate the existing sensitivity label.
998 .sp
999 This privilege is interpreted only if the system is configured with Trusted
1000 Extensions.
1001 .RE

1003 .sp
1004 .ne 2
1005 .na
1006 \fb\fbPRIV_WIN_FONTPATH\fr\fr
1007 .ad
1008 .sp .6
1009 .RS 4n
1010 Allow a process to set a font path.
1011 .sp
1012 This privilege is interpreted only if the system is configured with Trusted
1013 Extensions.
1014 .RE

1016 .sp
1017 .ne 2
1018 .na
1019 \fb\fbPRIV_WIN_MAC_READ\fr\fr
1020 .ad
1021 .sp .6
1022 .RS 4n
1023 Allow a process to read from a window resource whose sensitivity label is not
1024 equal to the process sensitivity label.
1025 .sp
1026 This privilege is interpreted only if the system is configured with Trusted
1027 Extensions.
1028 .RE

1030 .sp
1031 .ne 2
1032 .na
1033 \fb\fbPRIV_WIN_MAC_WRITE\fr\fr
1034 .ad
1035 .sp .6
1036 .RS 4n
1037 Allow a process to create a window resource whose sensitivity label is not
1038 equal to the process sensitivity label. A newly created window property is
1039 created with the window's sensitivity label.
1040 .sp
1041 This privilege is interpreted only if the system is configured with Trusted
1042 Extensions.
1043 .RE

1045 .sp
1046 .ne 2

```

```

1047 .na
1048 \fB\fBPRIV_WIN_SELECTION\fR\fR
1049 .ad
1050 .sp .6
1051 .RS 4n
1052 Allow a process to request inter-window data moves without the intervention of
1053 the selection confirmer.
1054 .sp
1055 This privilege is interpreted only if the system is configured with Trusted
1056 Extensions.
1057 .RE

1059 .sp
1060 .ne 2
1061 .na
1062 \fB\fBPRIV_WIN_UPGRADE_SL\fR\fR
1063 .ad
1064 .sp .6
1065 .RS 4n
1066 Allow a process to set the sensitivity label of a window resource to a
1067 sensitivity label that dominates the existing sensitivity label.
1068 .sp
1069 This privilege is interpreted only if the system is configured with Trusted
1070 Extensions.
1071 .RE

1073 .sp
1074 .ne 2
1075 .na
1076 \fB\fBPRIV_XVM_CONTROL\fR\fR
1077 .ad
1078 .sp .6
1079 .RS 4n
1080 Allows a process access to the \fBxVM\fR(5) control devices for managing guest
1081 domains and the hypervisor. This privilege is used only if booted into xVM on
1082 x86 platforms.
1083 .RE

1085 .sp
1086 .LP
1087 Of the privileges listed above, the privileges \fBPRIV_FILE_LINK_ANY\fR,
1088 \fBPRIV_PROC_INFO\fR, \fBPRIV_PROC_SESSION\fR, \fBPRIV_PROC_FORK\fR,
1089 \fBPRIV_FILE_READ\fR, \fBPRIV_FILE_WRITE\fR, \fBPRIV_NET_ACCESS\fR and
1090 \fBPRIV_PROC_EXEC\fR are considered "basic" privileges. These are privileges
1091 that used to be always available to unprivileged processes. By default,
1092 processes still have the basic privileges.
1093 .sp
1094 .LP
1095 The privileges \fBPRIV_PROC_SETID\fR and \fBPRIV_PROC_AUDIT\fR must be present
1096 in the Limit set (see below) of a process in order for set-uid root \fBExec\fRs
1097 to be successful, that is, get an effective UID of 0 and additional privileges.
1098 .sp
1099 .LP
1100 The privilege implementation in Solaris extends the process credential with
1101 four privilege sets:
1102 .sp
1103 .ne 2
1104 .na
1105 \fBBI, the inheritable set\fR
1106 .ad
1107 .RS 26n
1108 The privileges inherited on \fBExec\fR.
1109 .RE

1111 .sp
1112 .ne 2

```

```

1113 .na
1114 \fBFBP, the permitted set\fR
1115 .ad
1116 .RS 26n
1117 The maximum set of privileges for the process.
1118 .RE

1120 .sp
1121 .ne 2
1122 .na
1123 \fBFE, the effective set\fR
1124 .ad
1125 .RS 26n
1126 The privileges currently in effect.
1127 .RE

1129 .sp
1130 .ne 2
1131 .na
1132 \fBFL, the limit set\fR
1133 .ad
1134 .RS 26n
1135 The upper bound of the privileges a process and its offspring can obtain.
1136 Changes to L take effect on the next \fBExec\fR.
1137 .RE

1139 .sp
1140 .LP
1141 The sets I, P and E are typically identical to the basic set of privileges for
1142 unprivileged processes. The limit set is typically the full set of privileges.
1143 .sp
1144 .LP
1145 Each process has a Privilege Awareness State (PAS) that can take the value PA
1146 (privilege-aware) and NPA (not-PA). PAS is a transitional mechanism that allows
1147 a choice between full compatibility with the old superuser model and completely
1148 ignoring the effective UID.
1149 .sp
1150 .LP
1151 To facilitate the discussion, we introduce the notion of "observed effective
1152 set" (oE) and "observed permitted set" (oP) and the implementation sets iE and
1153 iP.
1154 .sp
1155 .LP
1156 A process becomes privilege-aware either by manipulating the effective,
1157 permitted, or limit privilege sets through \fBsetpriv\fR(2) or by using
1158 \fBsetpflags\fR(2). In all cases, oE and oP are invariant in the process of
1159 becoming privilege-aware. In the process of becoming privilege-aware, the
1160 following assignments take place:
1161 .sp
1162 .in +2
1163 .nf
1164 iE = oE
1165 iP = oP
1166 .fi
1167 .in -2

1169 .sp
1170 .LP
1171 When a process is privilege-aware, oE and oP are invariant under UID changes.
1172 When a process is not privilege-aware, oE and oP are observed as follows:
1173 .sp
1174 .in +2
1175 .nf
1176 oE = euid == 0 ? L : iE
1177 oP = (euid == 0 || ruid == 0 || suid == 0) ? L : iP
1178 .fi

```

```

1179 .in -2
1181 .sp
1182 .LP
1183 When a non-privilege-aware process has an effective UID of 0, it can exercise
1184 the privileges contained in its limit set, the upper bound of its privileges.
1185 If a non-privilege-aware process has any of the UIDs 0, it appears to be
1186 capable of potentially exercising all privileges in L.
1187 .sp
1188 .LP
1189 It is possible for a process to return to the non-privilege aware state using
1190 \fBsetpflags()\fR. The kernel always attempts this on \fBexec\fR(2). This
1191 operation is permitted only if the following conditions are met:
1192 .RS +4
1193 .TP
1194 .ie t \(\bu
1195 .el o
1196 If any of the UIDs is equal to 0, P must be equal to L.
1197 .RE
1198 .RS +4
1199 .TP
1200 .ie t \(\bu
1201 .el o
1202 If the effective UID is equal to 0, E must be equal to L.
1203 .RE
1204 .sp
1205 .LP
1206 When a process gives up privilege awareness, the following assignments take
1207 place:
1208 .sp
1209 .in +2
1210 .nf
1211 if (euid == 0) iE = L & I
1212 if (any uid == 0) iP = L & I
1213 .fi
1214 .in -2

1216 .sp
1217 .LP
1218 The privileges obtained when not having a UID of \fB0\fR are the inheritable
1219 set of the process restricted by the limit set.
1220 .sp
1221 .LP
1222 Only privileges in the process's (observed) effective privilege set allow the
1223 process to perform restricted operations. A process can use any of the
1224 privilege manipulation functions to add or remove privileges from the privilege
1225 sets. Privileges can be removed always. Only privileges found in the permitted
1226 set can be added to the effective and inheritable set. The limit set cannot
1227 grow. The inheritable set can be larger than the permitted set.
1228 .sp
1229 .LP
1230 When a process performs an \fBexec\fR(2), the kernel first tries to relinquish
1231 privilege awareness before making the following privilege set modifications:
1232 .sp
1233 .in +2
1234 .nf
1235 E' = P' = I' = L & I
1236 L is unchanged
1237 .fi
1238 .in -2

1240 .sp
1241 .LP
1242 If a process has not manipulated its privileges, the privilege sets effectively
1243 remain the same, as E, P and I are already identical.
1244 .sp

```

```

1245 .LP
1246 The limit set is enforced at \fBexec\fR time.
1247 .sp
1248 .LP
1249 To run a non-privilege-aware application in a backward-compatible manner, a
1250 privilege-aware application should start the non-privilege-aware application
1251 with I=basic.
1252 .sp
1253 .LP
1254 For most privileges, absence of the privilege simply results in a failure. In
1255 some instances, the absence of a privilege can cause system calls to behave
1256 differently. In other instances, the removal of a privilege can force a set-uid
1257 application to seriously malfunction. Privileges of this type are considered
1258 "unsafe". When a process is lacking any of the unsafe privileges from its limit
1259 set, the system does not honor the set-uid bit of set-uid root applications.
1260 The following unsafe privileges have been identified: \fBproc_setid\fR,
1261 \fBsys_resource\fR and \fBproc_audit\fR.
1262 .SS "Privilege Escalation"
1263 .LP
1264 In certain circumstances, a single privilege could lead to a process gaining
1265 one or more additional privileges that were not explicitly granted to that
1266 process. To prevent such an escalation of privileges, the security policy
1267 requires explicit permission for those additional privileges.
1268 .sp
1269 .LP
1270 Common examples of escalation are those mechanisms that allow modification of
1271 system resources through "raw" interfaces; for example, changing kernel data
1272 structures through \fB/dev/kmem\fR or changing files through \fB/dev/dsk/*\fR.
1273 Escalation also occurs when a process controls processes with more privileges
1274 than the controlling process. A special case of this is manipulating or
1275 creating objects owned by UID 0 or trying to obtain UID 0 using
1276 \fBsetuid\fR(2). The special treatment of UID 0 is needed because the UID 0
1277 owns all system configuration files and ordinary file protection mechanisms
1278 allow processes with UID 0 to modify the system configuration. With appropriate
1279 file modifications, a given process running with an effective UID of 0 can gain
1280 all privileges.
1281 .sp
1282 .LP
1283 In situations where a process might obtain UID 0, the security policy requires
1284 additional privileges, up to the full set of privileges. Such restrictions
1285 could be relaxed or removed at such time as additional mechanisms for
1286 protection of system files became available. There are no such mechanisms in
1287 the current Solaris release.
1288 .sp
1289 .LP
1290 The use of UID 0 processes should be limited as much as possible. They should
1291 be replaced with programs running under a different UID but with exactly the
1292 privileges they need.
1293 .sp
1294 .LP
1295 Daemons that never need to \fBexec\fR subprocesses should remove the
1296 \fBPRIV_PROC_EXEC\fR privilege from their permitted and limit sets.
1297 .SS "Assigned Privileges and Safeguards"
1298 .LP
1299 When privileges are assigned to a user, the system administrator could give
1300 that user more powers than intended. The administrator should consider whether
1301 safeguards are needed. For example, if the \fBPRIV_PROC_LOCK_MEMORY\fR
1302 privilege is given to a user, the administrator should consider setting the
1303 \fBproject.max-locked-memory\fR resource control as well, to prevent that user
1304 from locking all memory.
1305 .SS "Privilege Debugging"
1306 .LP
1307 When a system call fails with a permission error, it is not always immediately
1308 obvious what caused the problem. To debug such a problem, you can use a tool
1309 called \fBprivilege debugging\fR. When privilege debugging is enabled for a
1310 process, the kernel reports missing privileges on the controlling terminal of

```

```
1311 the process. (Enable debugging for a process with the \fB-D\fR option of
1312 \fBppriv\fR(1).) Additionally, the administrator can enable system-wide
1313 privilege debugging by setting the \fBsystem\fR(4) variable \fBpriv_debug\fR
1314 using:
1315 .sp
1316 .in +2
1317 .nf
1318 set priv_debug = 1
1319 .fi
1320 .in -2

1322 .sp
1323 .LP
1324 On a running system, you can use \fBmdb\fR(1) to change this variable.
1325 .SS "Privilege Administration"
1326 .LP
1327 The Solaris Management Console (see \fBsmc\fR(1M)) is the preferred method of
1328 modifying privileges for a command. Use \fBusermod\fR(1M) or \fBsmrole\fR(1M)
1329 to assign privileges to or modify privileges for, respectively, a user or a
1330 role. Use \fBppriv\fR(1) to enumerate the privileges supported on a system and
1331 \fBtruss\fR(1) to determine which privileges a program requires.
1332 .SH SEE ALSO
1333 .LP
1334 \fBmdb\fR(1), \fBppriv\fR(1), \fBadd_drv\fR(1M), \fBifconfig\fR(1M),
1335 \fBblockd\fR(1M), \fBbnfsd\fR(1M), \fBpppd\fR(1M), \fBrem_drv\fR(1M),
1336 \fBsmbd\fR(1M), \fBspptun\fR(1M), \fBupdate_drv\fR(1M), \fBintro\fR(2),
1337 \fBaccess\fR(2), \fBacct\fR(2), \fBacl\fR(2), \fBadjtime\fR(2), \fBaudit\fR(2),
1338 \fBauditon\fR(2), \fBchmod\fR(2), \fBchown\fR(2), \fBchroot\fR(2),
1339 \fBcreat\fR(2), \fBexec\fR(2), \fBfcntl\fR(2), \fBfork\fR(2),
1340 \fBfpathconf\fR(2), \fBgetacct\fR(2), \fBgetpflags\fR(2), \fBgetppriv\fR(2),
1341 \fBgetsid\fR(2), \fBkill\fR(2), \fBlink\fR(2), \fBmemcntl\fR(2),
1342 \fBmknod\fR(2), \fBmount\fR(2), \fBmsgctl\fR(2), \fBnice\fR(2),
1343 \fBntp_adjtime\fR(2), \fBopen\fR(2), \fBp_online\fR(2), \fBprioctl\fR(2),
1344 \fBprioctlset\fR(2), \fBprocessor_bind\fR(2), \fBpset_bind\fR(2),
1345 \fBpset_create\fR(2), \fBreadlink\fR(2), \fBresolvepath\fR(2), \fBmkdir\fR(2),
1346 \fBsemctl\fR(2), \fBsetauid\fR(2), \fBsetegid\fR(2), \fBseteuid\fR(2),
1347 \fBsetgid\fR(2), \fBsetgroups\fR(2), \fBsetpflags\fR(2), \fBsetppriv\fR(2),
1348 \fBsetrctl\fR(2), \fBsetregid\fR(2), \fBsetreuid\fR(2), \fBsetrlimit\fR(2),
1349 \fBsettaskid\fR(2), \fBsetuid\fR(2), \fBshmctl\fR(2), \fBshmget\fR(2),
1350 \fBshmop\fR(2), \fBsigsend\fR(2), \fBstat\fR(2), \fBstatvfs\fR(2),
1351 \fBstime\fR(2), \fBswapctl\fR(2), \fBsysinfo\fR(2), \fBuadmin\fR(2),
1352 \fBulimit\fR(2), \fBumount\fR(2), \fBunlink\fR(2), \fButime\fR(2),
1353 \fButimes\fR(2), \fBbind\fR(3SOCKET), \fBdoor_ucred\fR(3C),
1354 \fBpriv_addset\fR(3C), \fBpriv_set\fR(3C), \fBpriv_getbyname\fR(3C),
1355 \fBpriv_getbynum\fR(3C), \fBpriv_set_to_str\fR(3C), \fBpriv_str_to_set\fR(3C),
1356 \fBsocket\fR(3SOCKET), \fBt_bind\fR(3NSL), \fBtimer_create\fR(3C),
1357 \fBucred_get\fR(3C), \fBexec_attr\fR(4), \fBproc\fR(4), \fBsystem\fR(4),
1358 \fBuser_attr\fR(4), \fBxvm\fR(5), \fBddi_cred\fR(9F), \fBdrv_priv\fR(9F),
1359 \fBpriv_getbyname\fR(9F), \fBpriv_policy\fR(9F), \fBpriv_policy_choice\fR(9F),
1360 \fBpriv_policy_only\fR(9F)
1361 .sp
1362 .LP
1363 \fISystem Administration Guide: Security Services\fR
```

```

*****
3590 Thu Jun 30 22:02:52 2016
new/usr/src/man/man5/security-flags.5
Code review comments from jeffpc
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
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 2015, Richard Lowe.
12 .\"
13 .TH "SECURITY-FLAGS" "5" "June 6, 2016"
14 .SH "NAME"
15 \fBsecurity-flags\fR - process security flags
16 .SH "DESCRIPTION"
17 Each process on an illumos system has an associated set of security-flags
18 which describe additional per-process security and exploit mitigation
19 features which are enabled for that process.
20 .P
21 There are four sets of these flags for each process, the effective set
22 (abbreviated \fIE\fR) are the set which currently apply to the process and are
23 immutable. The inheritable set (abbreviated \fII\fR) are the flags which will
24 become effective the next time the process calls one of the \fBexec(2)\fR
25 family of functions, and will be inherited as both the effective and
26 inheritable sets by any child processes. The upper set (abbreviated \fIU\fR)
27 specify the maximal flags that a process can have in its inheritable set. The
28 lower set (abbreviated \fIL\fR) specify the minimal amount of flags that a
29 process must have in its inheritable set. The inheritable set may be changed
30 at any time, subject to permissions and the lower and upper sets.
31 .P
32 To change the security-flags of a process one must have both permissions
33 equivalent to those required to send a signal to the process and have the
34 \fBPRIV_PROC_SECFLAGS\fR privilege.
35 .P
36 Currently available features are:

38 .sp
39 .ne 2
40 .na
41 Address Space Layout Randomisation (\fBASLR\fR)
42 .ad
43 .RS 11n
44 The base addresses of the stack, heap and shared library (including
45 \fBld.so\fR) mappings are randomised, the bases of mapped regions other than
46 those using \fBMAP_FIXED\fR are randomised.
47 .P
48 Currently, executable base addresses are \fInot\fR randomised, due to which
49 the mitigation provided by this feature is currently limited.
50 .P
51 This flag may also be enabled by the presence of the \fBBDT_SUNW_ASLR\fR
52 dynamic tag in the \fBdynamic\fR section of the executable file. If this
53 tag has a value of 1, ASLR will be enabled. If the flag has a value of
54 \fB0\fR ASLR will be disabled. If the tag is not present, the value of the
55 ASLR flag will be inherited as normal.
56 .RE

```

```

58 .sp
59 .ne 2
60 .na
61 Forbid mappings at NULL (\fBFORBIDNULLMAP\fR)
62 .ad
63 .RS 11n
64 Mappings with an address of 0 are forbidden, and return EINVAL rather than
65 being honored.
66 .RE

68 .sp
69 .ne 2
70 .na
71 Make the userspace stack non-executable (\fBNOEXECSTACK\fR)
72 .ad
73 .RS 11n
74 The stack will be mapped without executable permission, and attempts to
75 execute it will fault.
76 .RE

78 System default security-flags are configured via properties on the
79 \fBsvc:/system/process-security\fR service, which contains a boolean property
80 per-flag in the \fBdefault\fR, \fBblower\fR and \fBbupper\fR, property groups.
81 For example, to enable ASLR by default you would execute the following
82 commands:
83 .sp
84 .in +2
85 .nf
86 # svccfg -s svc:/system/process-security setprop default/aslr = true
87 .fi
88 .in -2
89 .sp
90 .P
91 This can be done by any user with the \fBsolaris.smf.value.process-security\fR
92 authorization.
93 .P
94 Since security-flags are strictly inherited, this will not take effect until
95 the system or zone is next booted.

97 .SH "SEE ALSO"
98 .BR psecflags (1),
99 .BR svccfg (1M),
100 .BR brk (2),
101 .BR exec (2),
102 .BR mmap (2),
103 .BR mmapobj (2),
104 .BR privileges (5),
105 .BR rbac (5)
106 #endif /* ! codereview */

```



```

*****
14783 Thu Jun 30 22:02:52 2016
new/usr/src/man/man5/smf_method.5
Code review comments from jeffpc
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
1 \" te
2 .\\ Copyright (c) 2009, Sun Microsystems, Inc. All Rights Reserved.
3 .\\ The contents of this file are subject to the terms of the Common Development
4 .\\ See the License for the specific language governing permissions and limitat
5 .\\ the fields enclosed by brackets \"[]\" replaced with your own identifying info
6 .TH SMF_METHOD 5 \"June 6, 2016\"
6 .TH SMF_METHOD 5 \"May 20, 2009\"
7 .SH NAME
8 smf_method \- service management framework conventions for methods
9 .SH DESCRIPTION
10 .sp
10 .LP
11 The class of services managed by \fBsvc.startd\fR(1M) in the service management
12 framework, \fBsmf\fR(5), consists of applications that fit a simple
13 \fBfork\fR(2)-\fBexec\fR(2) model. The \fBsvc.startd\fR(1M) master daemon and
14 other restarters support the \fBfork\fR(2)-\fBexec\fR(2) model, potentially
15 with additional capabilities. The \fBsvc.startd\fR(1M) daemon and other
16 restarters require that the methods which activate, manipulate, or examine a
17 service instance follow the conventions described in this manual page.
18 .SS \"Invocation form\"
20 .sp
19 .LP
20 The form of a method invocation is not dictated by convention. In some cases, a
21 method invocation might consist of the direct invocation of the daemon or other
22 binary executable that provides the service. For cases in which an executable
23 script or other mediating executable is used, the convention recommends the
24 form:
25 .sp
26 .in +2
27 .nf
28 /path/to/method_executable abbr_method_name
29 .fi
30 .in -2

32 .sp
33 .LP
34 The \fIabbr_method_name\fR used for the recommended form is a supported method
35 such as \fBstart\fR or \fBstop\fR. The set of methods supported by a restarter
36 is given on the related restarter page. The \fBsvc.startd\fR(1M) daemon
37 supports \fBstart\fR, \fBstop\fR, and \fBrefresh\fR methods.
38 .sp
39 .LP
40 A restarter might define other kinds of methods beyond those referenced in this
41 page. The conventions surrounding such extensions are defined by the restarter
42 and might not be identical to those given here.
43 .SS \"Environment Variables\"
46 .sp
44 .LP
45 The restarter provides four environment variables to the method that determine
46 the context in which the method is invoked.
47 .sp
48 .ne 2
49 .na
50 \fB\BFSMF_FMRI\fR
51 .ad
52 .sp .6
53 .RS 4n

```

```

54 The service fault management resource identifier (FMRI) of the instance for
55 which the method is invoked.
56 .RE

58 .sp
59 .ne 2
60 .na
61 \fB\BFSMF_METHOD\fR
62 .ad
63 .sp .6
64 .RS 4n
65 The full name of the method being invoked, such as \fBstart\fR or \fBstop\fR.
66 .RE

68 .sp
69 .ne 2
70 .na
71 \fB\BFSMF_RESTARTER\fR
72 .ad
73 .sp .6
74 .RS 4n
75 The service FMRI of the restarter that invokes the method
76 .RE

78 .sp
79 .ne 2
80 .na
81 \fB\BFSMF_ZONENAME\fR
82 .ad
83 .sp .6
84 .RS 4n
85 The name of the zone in which the method is running. This can also be obtained
86 by using the \fBzonename\fR(1) command.
87 .RE

89 .sp
90 .LP
91 These variables should be removed from the environment prior to the invocation
92 of any persistent process by the method. A convenience shell function,
93 \fBsmf_clear_env\fR, is given for service authors who use Bourne-compatible
94 shell scripting to compose service methods in the include file described below.
95 .sp
96 .LP
97 The method context can cause other environment variables to be set as described
98 below.
99 .SS \"Method Definition\"
103 .sp
100 .LP
101 A method is defined minimally by three properties in a propertygroup of type
102 \fBmethod\fR.
103 .sp
104 .LP
105 These properties are:
106 .sp
107 .ne 2
108 .na
109 \fB\Bexec (\fIastring\fR)\fR
110 .ad
111 .RS 27n
112 Method executable string.
113 .RE

115 .sp
116 .ne 2
117 .na
118 \fB\Btimeout_seconds (\fIcount\fR)\fR

```

```

119 .ad
120 .RS 27n
121 Number of seconds before method times out. See the \fBTimeouts\fR section for
122 more detail.
123 .RE

125 .sp
126 .ne 2
127 .na
128 \fBtype (\fIastring\fR)\fR
129 .ad
130 .RS 27n
131 Method type. Currently always set to \fBmethod\fR.
132 .RE

134 .sp
135 .LP
136 A Method Context can be defined to further refine the execution environment of
137 the method. See the \fBMethod Context\fR section for more information.
138 .SS "Method Tokens"
139 .sp
140 When defined in the \fBexec\fR string of the method by the restarter
141 \fBsvc.startd\fR, a set of tokens are parsed and expanded with appropriate
142 value. Other restarters might not support method tokens. The delegated
143 restarter for inet services, \fBinetd\fR(1M), does not support the following
144 method expansions.
145 .sp
146 .ne 2
147 .na
148 \fB\fB%\fR\fR
149 .ad
150 .sp .6
151 .RS 4n
152 %
153 .RE

155 .sp
156 .ne 2
157 .na
158 \fB\fB:r\fR\fR
159 .ad
160 .sp .6
161 .RS 4n
162 Name of the restarter, such as \fBsvc.startd\fR
163 .RE

165 .sp
166 .ne 2
167 .na
168 \fB\fB:m\fR\fR
169 .ad
170 .sp .6
171 .RS 4n
172 The full name of the method being invoked, such as \fBstart\fR or \fBstop\fR.
173 .RE

175 .sp
176 .ne 2
177 .na
178 \fB\fB:s\fR\fR
179 .ad
180 .sp .6
181 .RS 4n
182 Name of the service
183 .RE

```

```

185 .sp
186 .ne 2
187 .na
188 \fB\fB:i\fR\fR
189 .ad
190 .sp .6
191 .RS 4n
192 Name of the instance
193 .RE

195 .sp
196 .ne 2
197 .na
198 \fB\fB:f\fR\fR
199 .ad
200 .sp .6
201 .RS 4n
202 FMRI of the instance
203 .RE

205 .sp
206 .ne 2
207 .na
208 \fB\fB{prop[:]}\fR\fR
209 .ad
210 .sp .6
211 .RS 4n
212 Value(s) of a property. The \fBprop\fR might be a property FMRI, a property
213 group name and a property name separated by a \fB/\fR, or a property name in
214 the \fBApplication\fR property group. These values can be followed by a \fB,\fR
215 (comma) or \fB:\fR (colon). If present, the separators are used to separate
216 multiple values. If absent, a space is used. The following shell metacharacters
217 encountered in string values are quoted with a \ (backslash):
218 .sp
219 .in +2
220 .nf
221 ; & ( ) | ^ < > newline space tab \ " '
222 .fi
223 .in -2

225 An invalid expansion constitutes method failure.
226 .RE

228 .sp
229 .LP
230 Two explicit tokens can be used in the place of method commands.
231 .sp
232 .ne 2
233 .na
234 \fB\fB:kill [-signal]\fR\fR
235 .ad
236 .sp .6
237 .RS 4n
238 Sends the specified signal, which is \fBSIGTERM\fR by default, to all processes
239 in the primary instance contract. Always returns \fBEXIT_OK\fR. This token
240 should be used to replace common \fBkill\fR invocations.
241 .RE

243 .sp
244 .ne 2
245 .na
246 \fB\fB:true\fR\fR
247 .ad
248 .sp .6
249 .RS 4n

```

```

250 Always returns \fB$EXIT_OK\fR. This token should be used for methods that
251 are required by the restarters but which are unnecessary for the particular
252 service implementation.
253 .RE

255 .SS "Exiting and Exit Status"
256 .sp
257 .LP
258 The required behavior of a start method is to delay exiting until the service
259 instance is ready to answer requests or is otherwise functional.
260 .sp
261 .LP
262 The following exit status codes are defined in \fB<libscf.h>\fR and in the
263 shell support file.

265 .sp
266 .TS
267 1 1 1
268 1 1 1 .
269 \fB$EXIT_OK\fR          \fB0\fR T{
270 Method exited, performing its operation successfully.
271 T}
272 \fB$EXIT_ERR_FATAL\fR   \fB95\fR T{
273 Method failed fatally and is unrecoverable without administrative intervention.
274 T}
275 \fB$EXIT_ERR_CONFIG\fR   \fB96\fR T{
276 Unrecoverable configuration error. A common condition that returns this exit sta
277 T}
278 \fB$EXIT_ERR_NOSMF\fR    \fB99\fR T{
279 Method has been mistakenly invoked outside the \fB$smf(5)\fR facility. Services t
280 T}
281 \fB$EXIT_ERR_PERM\fR     \fB100\fR T{
282 Method requires a form of permission such as file access, privilege, authorizati
283 T}
284 \fB$EXIT_ERR_OTHER\fR    \fBnon-zero\fR T{
285 Any non-zero exit status from a method is treated as an unknown error. A series
286 T}
287 .TE

289 .sp
290 .LP
291 Use of a precise exit code allows the responsible restarters to categorize an
292 error response as likely to be intermittent and worth pursuing restart or
293 permanent and request administrative intervention.
294 .SS "Timeouts"
295 .sp
296 .LP
297 Each method can have an independent timeout, given in seconds. The choice of a
298 particular timeout should be based on site expectations for detecting a method
299 failure due to non-responsiveness. Sites with replicated filesystems or other
300 failover resources can elect to lengthen method timeouts from the default.
301 Sites with no remote resources can elect to shorten the timeouts. Method
302 timeout is specified by the \fBtimeout_seconds\fR property.
303 .sp
304 .LP
305 If you specify \fB0 timeout_seconds\fR for a method, it declares to the
306 restarters that there is no timeout for the service. This setting is not
307 preferred, but is available for services that absolutely require it.
308 .sp
309 .LP
310 \fB-1 timeout_seconds\fR is also accepted, but is a deprecated specification.
311 .SS "Shell Programming Support"
312 .sp
313 A set of environment variables that define the above exit status values is

```

```

313 provided with convenience shell functions in the file
314 \fB/lib/svc/share/smf_include.sh\fR. This file is a Bourne shell script
315 suitable for inclusion via the source operator in any Bourne-compatible shell.
316 .sp
317 .LP
318 To assist in the composition of scripts that can serve as SMF methods as well
319 as \fB/etc/init.d\fR scripts, the \fBsmf_present()\fR shell function is
320 provided. If the \fB$smf(5)\fR facility is not available, \fBsmf_present()\fR
321 returns a non-zero exit status.
322 .sp
323 .LP
324 One possible structure for such a script follows:
325 .sp
326 .in +2
327 .nf
328 if smf_present; then
329     # Shell code to run application as managed service
330     ....
331
332     smf_clear_env
333 else
334     # Shell code to run application as /etc/init.d script
335     ....
336 fi
337 .fi
338 .in -2

340 .sp
341 .LP
342 This example shows the use of both convenience functions that are provided.
343 .SS "Method Context"
344 .sp
345 .LP
346 The service management facility offers a common mechanism set the context in
347 which the \fBfork(2)\fR-\fBexec(2)\fR model services execute.
348 .sp
349 .LP
350 The desired method context should be provided by the service developer. All
351 service instances should run with the lowest level of privileges possible to
352 limit potential security compromises.
353 .sp
354 .LP
355 A method context can contain the following properties:
356 .sp
357 .ne 2
358 .na
359 \fB$Buse_profile\fR
360 .ad
361 .sp .6
362 .RS 4n
363 A boolean that specifies whether the profile should be used instead of the
364 \fB$Buser\fR, \fB$Bgroup\fR, \fB$Bprivileges\fR, and \fB$Blimit_privileges\fR
365 properties.
366 .RE

367 .sp
368 .ne 2
369 .na
370 \fB$Benvironment\fR
371 .ad
372 .sp .6
373 .RS 4n
374 Environment variables to insert into the environment of the method, in the form
375 of a number of \fB$BNAME=value\fR strings.
376 .RE

```

```

378 .sp
379 .ne 2
380 .na
381 \fB\fBprofile\fR\fR
382 .ad
383 .sp .6
384 .RS 4n
385 The name of an RBAC (role-based access control) profile which, along with the
386 method executable, identifies an entry in \fBexec_attr\fR(4).
387 .RE

389 .sp
390 .ne 2
391 .na
392 \fB\fBuser\fR\fR
393 .ad
394 .sp .6
395 .RS 4n
396 The user ID in numeric or text form.
397 .RE

399 .sp
400 .ne 2
401 .na
402 \fB\fBgroup\fR\fR
403 .ad
404 .sp .6
405 .RS 4n
406 The group ID in numeric or text form.
407 .RE

409 .sp
410 .ne 2
411 .na
412 \fB\fBsupp_groups\fR\fR
413 .ad
414 .sp .6
415 .RS 4n
416 An optional string that specifies the supplemental group memberships by ID, in
417 numeric or text form.
418 .RE

420 .sp
421 .ne 2
422 .na
423 \fB\fBprivileges\fR\fR
424 .ad
425 .sp .6
426 .RS 4n
427 An optional string specifying the privilege set as defined in
428 \fBprivileges\fR(5).
429 .RE

431 .sp
432 .ne 2
433 .na
434 \fB\fBlimit_privileges\fR\fR
435 .ad
436 .sp .6
437 .RS 4n
438 An optional string specifying the limit privilege set as defined in
439 \fBprivileges\fR(5).
440 .RE

442 .sp
443 .ne 2

```

```

444 .na
445 \fB\fBworking_directory\fR\fR
446 .ad
447 .sp .6
448 .RS 4n
449 The home directory from which to launch the method. \fB:home\fR can be used as
450 a token to indicate the home directory of the user whose \fBuid\fR is used to
451 launch the method. If the property is unset, \fB:home\fR is used.
452 .RE

454 .sp
455 .ne 2
456 .na
457 \fB\fBsecurity_flags\fR\fR
458 .ad
459 .sp .6
460 .RS 4n
461 The security flags to apply when launching the method. See \fBsecurity-flags\fR
462 .sp
463 .LP
464 The "default" keyword specifies those flags specified in
465 \fBsvc:/system/process-security\fR. The "all" keyword enables all flags, the
466 "none" keyword enables no flags. Further flags may be added by specifying
467 their name, or removed by specifying their name prefixed by '-' or '!'.
468 .sp
469 .LP
470 Use of "all" has associated risks, as future versions of the system may
471 include further flags which may harm poorly implemented software.
472 .RE

474 .sp
475 .ne 2
476 .na
477 #endif /* ! codereview */
478 \fB\fBcorefile_pattern\fR\fR
479 .ad
480 .sp .6
481 .RS 4n
482 An optional string that specifies the corefile pattern to use for the service,
483 as per \fBcoreadm\fR(1M). Most restarters supply a default. Setting this
484 property overrides local customizations to the global core pattern.
485 .RE

487 .sp
488 .ne 2
489 .na
490 \fB\fBproject\fR\fR
491 .ad
492 .sp .6
493 .RS 4n
494 The project ID in numeric or text form. \fB:default\fR can be used as a token
495 to indicate a project identified by \fBgetdefaultproj\fR(3PROJECT) for the user
496 whose \fBuid\fR is used to launch the method.
497 .RE

499 .sp
500 .ne 2
501 .na
502 \fB\fBresource_pool\fR\fR
503 .ad
504 .sp .6
505 .RS 4n
506 The resource pool name on which to launch the method. \fB:default\fR can be
507 used as a token to indicate the pool specified in the \fBproject\fR(4) entry
508 given in the \fBproject\fR attribute above.
509 .RE

```

```

511 .sp
512 .LP
513 The method context can be set for the entire service instance by specifying a
514 \fBmethod_context\fR property group for the service or instance. A method might
515 override the instance method context by providing the method context properties
516 on the method property group.
517 .sp
518 .LP
519 Invalid method context settings always lead to failure of the method, with the
520 exception of invalid environment variables that issue warnings.
521 .sp
522 .LP
523 In addition to the context defined above, many \fBfork\fR(2)-\fBexec\fR(2)
524 model restarters also use the following conventions when invoking executables
525 as methods:
526 .sp
527 .ne 2
528 .na
529 \fBArgument array\fR
530 .ad
531 .sp .6
532 .RS 4n
533 The arguments in \fBargv[]\fR are set consistently with the result \fB/bin/sh
534 -c\fR of the \fBexec\fR string.
535 .RE

537 .sp
538 .ne 2
539 .na
540 \fBFile descriptors\fR
541 .ad
542 .sp .6
543 .RS 4n
544 File descriptor \fB0\fR is \fB/dev/null\fR. File descriptors \fB1\fR and
545 \fB2\fR are recommended to be a per-service log file.
546 .RE

548 .SH FILES
466 .sp
549 .ne 2
550 .na
551 \fB\fB/lib/svc/share/smf_include.sh\fR\fR
552 .ad
553 .sp .6
554 .RS 4n
555 Definitions of exit status values.
556 .RE

558 .sp
559 .ne 2
560 .na
561 \fB\fB/usr/include/libscf.h\fR\fR
562 .ad
563 .sp .6
564 .RS 4n
565 Definitions of exit status codes.
566 .RE

568 .SH SEE ALSO
487 .sp
569 .LP
570 \fB\fBzonename\fR(1), \fB\fBcoreadm\fR(1M), \fB\fBinetd\fR(1M), \fB\fBsvccfg\fR(1M),
571 \fB\fBsvc.startd\fR(1M), \fB\fBexec\fR(2), \fB\fBfork\fR(2),
572 \fB\fBgetdefaultproj\fR(3PROJECT), \fB\fBexec_attr\fR(4), \fB\fBproject\fR(4),
573 \fB\fBservice_bundle\fR(4), \fB\fBattributes\fR(5), \fB\fBprivileges\fR(5),

```

```

574 \fB\fBbac\fR(5), \fB\fBsmf\fR(5), \fB\fBsmf_bootstrap\fR(5), \fB\fBzones\fR(5),
575 \fB\fBsecurity-flags\fR(5)
493 \fB\fBbac\fR(5), \fB\fBsmf\fR(5), \fB\fBsmf_bootstrap\fR(5), \fB\fBzones\fR(5)
576 .SH NOTES
495 .sp
577 .LP
578 The present version of \fBsmf\fR(5) does not support multiple repositories.
579 .sp
580 .LP
581 When a service is configured to be started as root but with privileges
582 different from \fBlimit_privileges\fR, the resulting process is privilege
583 aware. This can be surprising to developers who expect \fBsetuid<non-zero
584 UID>\fR to reduce privileges to basic or less.

```

new/usr/src/pkg/manifests/SUNWcs.man5.inc

1

```
*****
2985 Thu Jun 30 22:02:54 2016
new/usr/src/pkg/manifests/SUNWcs.man5.inc
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
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
9 # at http://www.illumos.org/license/CDDL.
10 #
11 #
12 #
13 # Copyright 2011, Richard Lowe
14 # Copyright 2012 Nexenta Systems, Inc. All rights reserved.
15 #
16 #
17 file path=usr/share/man/man5/Intro.5
18 file path=usr/share/man/man5/acl.5
19 file path=usr/share/man/man5/ad.5
20 file path=usr/share/man/man5/ascii.5
21 file path=usr/share/man/man5/attributes.5
22 file path=usr/share/man/man5/device_clean.5
23 file path=usr/share/man/man5/dhcp.5
24 file path=usr/share/man/man5/filesystem.5
25 file path=usr/share/man/man5/formats.5
26 file path=usr/share/man/man5/iconv.5
27 file path=usr/share/man/man5/iconv_unicode.5
28 file path=usr/share/man/man5/privileges.5
29 file path=usr/share/man/man5/rbac.5
30 file path=usr/share/man/man5/resource_controls.5
31 file path=usr/share/man/man5/security-flags.5
32 #endif /* ! codereview */
33 file path=usr/share/man/man5/smf.5
34 file path=usr/share/man/man5/smf_bootstrap.5
35 file path=usr/share/man/man5/smf_method.5
36 file path=usr/share/man/man5/smf_restarter.5
37 file path=usr/share/man/man5/smf_security.5
38 file path=usr/share/man/man5/smf_template.5
39 file path=usr/share/man/man5/standards.5
40 file path=usr/share/man/man5/sticky.5
41 file path=usr/share/man/man5/term.5
42 link path=usr/share/man/man5/ANSI.5 target=standards.5
43 link path=usr/share/man/man5/C++.5 target=standards.5
44 link path=usr/share/man/man5/C.5 target=standards.5
45 link path=usr/share/man/man5/CSI.5 target=attributes.5
46 link path=usr/share/man/man5/ISO.5 target=standards.5
47 link path=usr/share/man/man5/MT-Level.5 target=attributes.5
48 link path=usr/share/man/man5/POSIX.1.5 target=standards.5
49 link path=usr/share/man/man5/POSIX.2.5 target=standards.5
50 link path=usr/share/man/man5/POSIX.5 target=standards.5
51 link path=usr/share/man/man5/RBAC.5 target=rbac.5
52 link path=usr/share/man/man5/SUS.5 target=standards.5
53 link path=usr/share/man/man5/SUSv2.5 target=standards.5
54 link path=usr/share/man/man5/SUSv3.5 target=standards.5
55 link path=usr/share/man/man5/SVID.5 target=standards.5
56 link path=usr/share/man/man5/SVID3.5 target=standards.5
57 link path=usr/share/man/man5/XNS.5 target=standards.5
58 link path=usr/share/man/man5/XNS4.5 target=standards.5
```

new/usr/src/pkg/manifests/SUNWcs.man5.inc

2

```
59 link path=usr/share/man/man5/XNS5.5 target=standards.5
60 link path=usr/share/man/man5/XPG.5 target=standards.5
61 link path=usr/share/man/man5/XPG3.5 target=standards.5
62 link path=usr/share/man/man5/XPG4.5 target=standards.5
63 link path=usr/share/man/man5/XPG4v2.5 target=standards.5
64 link path=usr/share/man/man5/architecture.5 target=attributes.5
65 link path=usr/share/man/man5/availability.5 target=attributes.5
66 link path=usr/share/man/man5/intro.5 target=Intro.5
67 link path=usr/share/man/man5/stability.5 target=attributes.5
68 link path=usr/share/man/man5/standard.5 target=attributes.5
```

new/usr/src/pkg/manifests/SUNWcs.mf

1

```
*****
87354 Thu Jun 30 22:02:55 2016
new/usr/src/pkg/manifests/SUNWcs.mf
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
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 #
23 # Copyright (c) 2010, Oracle and/or its affiliates. All rights reserved.
24 # Copyright (c) 2013 Gary Mills
25 # Copyright 2014 Garrett D'Amore <garrett@damore.org>
26 # Copyright 2015 Nexenta Systems, Inc. All rights reserved.
27 #
28 #
29 <include SUNWcs.man1.inc>
30 <include SUNWcs.man1m.inc>
31 <include SUNWcs.man4.inc>
32 <include SUNWcs.man5.inc>
33 <include SUNWcs.man7d.inc>
34 <include SUNWcs.man7fs.inc>
35 set name=pkg.fmri value=pkg:/SUNWcs@$(PKGVERS)
36 set name=pkg.description \
37     value="core software for a specific instruction-set architecture"
38 set name=pkg.summary value="Core Solaris"
39 set name=info.classification value=org.opensolaris.category.2008:System/Core
40 set name=variant.arch value=$(ARCH)
41 dir path=dev group=sys
42 dir path=etc group=sys
43 dir path=etc/cron.d group=sys
44 dir path=etc/crypto group=sys
45 dir path=etc/crypto/certs group=sys
46 dir path=etc/crypto/crls group=sys
47 dir path=etc/default group=sys
48 dir path=etc/dev group=sys
49 dir path=etc/devices group=sys
50 dir path=etc/dfs group=sys
51 dir path=etc/dhcp group=sys
52 dir path=etc/fs group=sys
53 dir path=etc/fs/dev group=sys
54 dir path=etc/fs/hsfs group=sys
55 dir path=etc/fs/ufs group=sys
56 dir path=etc/ftpd group=sys
57 dir path=etc/inet group=sys
58 dir path=etc/init.d group=sys
```

new/usr/src/pkg/manifests/SUNWcs.mf

2

```
59 dir path=etc/lib group=sys
60 dir path=etc/logadm.d group=sys
61 dir path=etc/mail group=mail
62 dir path=etc/net group=sys
63 dir path=etc/net/ticlts group=sys
64 dir path=etc/net/ticots group=sys
65 dir path=etc/net/ticotsord group=sys
66 dir path=etc/opt group=sys
67 dir path=etc/rc0.d group=sys
68 dir path=etc/rc1.d group=sys
69 dir path=etc/rc2.d group=sys
70 dir path=etc/rc3.d group=sys
71 dir path=etc/rcS.d group=sys
72 dir path=etc/rpcsec group=sys
73 dir path=etc/saf
74 dir path=etc/saf/zsmon group=sys
75 dir path=etc/sasl group=sys
76 dir path=etc/security group=sys
77 dir path=etc/security/audit group=sys
78 dir path=etc/security/audit/localhost group=sys
79 dir path=etc/security/auth_attr.d group=sys
80 dir path=etc/security/dev group=sys
81 dir path=etc/security/exec_attr.d group=sys
82 dir path=etc/security/lib group=sys
83 dir path=etc/security/prof_attr.d group=sys
84 dir path=etc/skel group=sys
85 dir path=etc/svc group=sys
86 dir path=etc/svc/profile group=sys
87 dir path=etc/svc/profile/site group=sys
88 dir path=etc/svc/volatile group=sys
89 dir path=etc/sysevent group=sys
90 dir path=etc/sysevent/config group=sys
91 dir path=etc/tm group=sys
92 dir path=etc/user_attr.d group=sys
93 dir path=export group=sys
94 dir path=home group=root mode=0555
95 dir path=lib
96 dir path=lib/crypto
97 dir path=lib/inet
98 dir path=lib/svc
99 dir path=lib/svc/bin
100 dir path=lib/svc/capture
101 dir path=lib/svc/manifest group=sys
102 dir path=lib/svc/manifest/application group=sys
103 dir path=lib/svc/manifest/application/management group=sys
104 dir path=lib/svc/manifest/application/security group=sys
105 dir path=lib/svc/manifest/device group=sys
106 dir path=lib/svc/manifest/milestone group=sys
107 dir path=lib/svc/manifest/network group=sys
108 dir path=lib/svc/manifest/network/dns group=sys
109 dir path=lib/svc/manifest/network/ipsec group=sys
110 dir path=lib/svc/manifest/network/ldap group=sys
111 dir path=lib/svc/manifest/network/routing group=sys
112 dir path=lib/svc/manifest/network/rpc group=sys
113 dir path=lib/svc/manifest/network/shares group=sys
114 dir path=lib/svc/manifest/network/ssl group=sys
115 dir path=lib/svc/manifest/platform group=sys
116 $(sparc_ONLY)dir path=lib/svc/manifest/platform/sun4u group=sys
117 dir path=lib/svc/manifest/site group=sys
118 dir path=lib/svc/manifest/system group=sys
119 dir path=lib/svc/manifest/system/device group=sys
120 dir path=lib/svc/manifest/system/filesystem group=sys
121 dir path=lib/svc/manifest/system/security group=sys
122 dir path=lib/svc/manifest/system/svc group=sys
123 dir path=lib/svc/method
124 dir path=lib/svc/monitor
```

```

125 dir path=lib/svc/seed
126 dir path=lib/svc/share
127 dir path=mnt group=sys
128 dir path=opt group=sys
129 dir path=proc group=root mode=0555
130 dir path=root group=root mode=0700
131 dir path=sbin group=sys
132 dir path=system group=root
133 dir path=system/boot group=root mode=0555
134 dir path=system/contract group=root mode=0555
135 dir path=system/object group=root mode=0555
136 dir path=tmp group=sys mode=1777
137 dir path=usr group=sys
138 dir path=usr/bin
139 dir path=usr/bin/$(ARCH32)
140 dir path=usr/bin/$(ARCH64)
141 dir path=usr/ccs
142 dir path=usr/ccs/bin
143 dir path=usr/demo
144 dir path=usr/games
145 dir path=usr/has
146 dir path=usr/has/bin
147 dir path=usr/has/lib
148 dir path=usr/has/man
149 dir path=usr/has/man/manlhas
150 dir path=usr/kernel group=sys
151 dir path=usr/kernel/drv group=sys
152 dir path=usr/kernel/drv/$(ARCH64) group=sys
153 dir path=usr/kernel/exec group=sys
154 dir path=usr/kernel/exec/$(ARCH64) group=sys
155 dir path=usr/kernel/fs group=sys
156 dir path=usr/kernel/fs/$(ARCH64) group=sys
157 dir path=usr/kernel/pcbe group=sys
158 dir path=usr/kernel/pcbe/$(ARCH64) group=sys
159 dir path=usr/kernel/sched group=sys
160 dir path=usr/kernel/sched/$(ARCH64) group=sys
161 dir path=usr/kernel/strmod group=sys
162 dir path=usr/kernel/strmod/$(ARCH64) group=sys
163 dir path=usr/kernel/sys group=sys
164 dir path=usr/kernel/sys/$(ARCH64) group=sys
165 dir path=usr/kvm
166 dir path=usr/lib
167 dir path=usr/lib/$(ARCH64)
168 dir path=usr/lib/audit
169 dir path=usr/lib/class
170 dir path=usr/lib/class/FX
171 dir path=usr/lib/class/IA
172 dir path=usr/lib/class/RT
173 dir path=usr/lib/class/SDC
174 dir path=usr/lib/class/TS
175 dir path=usr/lib/crypto
176 dir path=usr/lib/devfsadm group=sys
177 dir path=usr/lib/devfsadm/linkmod group=sys
178 dir path=usr/lib/fs group=sys
179 dir path=usr/lib/fs/autofs group=sys
180 dir path=usr/lib/fs/autofs/$(ARCH64) group=sys
181 dir path=usr/lib/fs/bootfs group=sys
182 dir path=usr/lib/fs/ctfs group=sys
183 dir path=usr/lib/fs/dev group=sys
184 dir path=usr/lib/fs/fd group=sys
185 dir path=usr/lib/fs/hsfs group=sys
186 dir path=usr/lib/fs/lofs group=sys
187 dir path=usr/lib/fs/mntfs group=sys
188 dir path=usr/lib/fs/nfs group=sys
189 dir path=usr/lib/fs/nfs/$(ARCH64) group=sys
190 dir path=usr/lib/fs/objfs group=sys

```

```

191 dir path=usr/lib/fs/proc group=sys
192 dir path=usr/lib/fs/sharefs group=sys
193 dir path=usr/lib/fs/tmpfs group=sys
194 dir path=usr/lib/fs/ufs group=sys
195 dir path=usr/lib/help
196 dir path=usr/lib/help/auths
197 dir path=usr/lib/help/auths/locale
198 dir path=usr/lib/help/auths/locale/C
199 dir path=usr/lib/help/profiles
200 dir path=usr/lib/help/profiles/locale
201 dir path=usr/lib/help/profiles/locale/C
202 dir path=usr/lib/iconv
203 dir path=usr/lib/inet
204 dir path=usr/lib/inet/$(ARCH32)
205 dir path=usr/lib/inet/$(ARCH64)
206 dir path=usr/lib/locale
207 dir path=usr/lib/locale/C
208 dir path=usr/lib/locale/C/LC_COLLATE
209 dir path=usr/lib/locale/C/LC_CTYPE
210 dir path=usr/lib/locale/C/LC_MESSAGES
211 dir path=usr/lib/locale/C/LC_MONETARY
212 dir path=usr/lib/locale/C/LC_NUMERIC
213 dir path=usr/lib/locale/C/LC_TIME
214 dir path=usr/lib/netsvc group=sys
215 dir path=usr/lib/pci
216 dir path=usr/lib/rcm
217 dir path=usr/lib/rcm/modules
218 dir path=usr/lib/rcm/scripts
219 dir path=usr/lib/reparse
220 dir path=usr/lib/saf
221 dir path=usr/lib/secure
222 dir path=usr/lib/secure/$(ARCH64)
223 dir path=usr/lib/security
224 dir path=usr/lib/sysevent
225 dir path=usr/lib/sysevent/modules
226 dir path=usr/net group=sys
227 dir path=usr/net/nls group=sys
228 dir path=usr/net/servers group=sys
229 dir path=usr/old
230 dir path=usr/platform group=sys
231 dir path=usr/sadm
232 dir path=usr/sadm/bin
233 dir path=usr/sadm/install
234 dir path=usr/sadm/install/scripts
235 dir path=usr/sbin
236 $(i386_ONLY)dir path=usr/sbin/$(ARCH32)
237 dir path=usr/sbin/$(ARCH64)
238 dir path=usr/share
239 dir path=usr/share/doc group=other
240 dir path=usr/share/doc/ksh
241 dir path=usr/share/doc/ksh/images
242 dir path=usr/share/doc/ksh/images/callouts
243 dir path=usr/share/lib
244 dir path=usr/share/lib/mailx
245 dir path=usr/share/lib/pub
246 dir path=usr/share/lib/tabset
247 dir path=usr/share/lib/xml group=sys
248 dir path=usr/share/lib/xml/dtd group=sys
249 dir path=usr/share/lib/xml/style group=sys
250 dir path=usr/share/man
251 dir path=usr/share/man/man1
252 dir path=usr/share/man/man1m
253 dir path=usr/share/man/man4
254 dir path=usr/share/man/man5
255 dir path=usr/share/man/man7d
256 dir path=usr/share/man/man7fs

```


new/usr/src/pkg/manifests/SUNWcs.mf

```

257 dir path=usr/share/src group=sys
258 dir path=var group=sys
259 dir path=var/adm group=sys mode=0775
260 dir path=var/adm/exacct group=adm owner=adm
261 dir path=var/adm/log group=adm owner=adm
262 dir path=var/adm/streams group=sys
263 dir path=var/audit group=sys
264 dir path=var/cores group=sys
265 dir path=var/cron group=sys
266 dir path=var/games
267 dir path=var/ldap group=daemon owner=daemon
268 dir path=var/inet group=sys
269 dir path=var/ld
270 dir path=var/ld/$(ARCH64)
271 dir path=var/log group=sys
272 dir path=var/logadm
273 dir path=var/mail group=mail mode=1777
274 dir path=var/mail/:saved group=mail mode=0775
275 dir path=var/news
276 dir path=var/opt group=sys
277 dir path=var/preserve mode=1777
278 dir path=var/run group=sys
279 dir path=var/saf
280 dir path=var/saf/zsmon group=sys
281 dir path=var/spool
282 dir path=var/spool/cron group=sys
283 dir path=var/spool/cron/atjobs group=sys
284 dir path=var/spool/cron/crontabs group=sys
285 dir path=var/spool/locks group=uucp owner=uucp
286 dir path=var/svc group=sys
287 dir path=var/svc/log group=sys
288 dir path=var/svc/manifest group=sys
289 dir path=var/svc/manifest/application group=sys
290 dir path=var/svc/manifest/application/management group=sys
291 dir path=var/svc/manifest/application/print group=sys
292 dir path=var/svc/manifest/application/security group=sys
293 dir path=var/svc/manifest/device group=sys
294 dir path=var/svc/manifest/milestone group=sys
295 dir path=var/svc/manifest/network group=sys
296 dir path=var/svc/manifest/network/dns group=sys
297 dir path=var/svc/manifest/network/ipsec group=sys
298 dir path=var/svc/manifest/network/ldap group=sys
299 dir path=var/svc/manifest/network/nfs group=sys
300 dir path=var/svc/manifest/network/nis group=sys
301 dir path=var/svc/manifest/network/routing group=sys
302 dir path=var/svc/manifest/network/rpc group=sys
303 dir path=var/svc/manifest/network/security group=sys
304 dir path=var/svc/manifest/network/shares group=sys
305 dir path=var/svc/manifest/network/ssl group=sys
306 dir path=var/svc/manifest/platform group=sys
307 $(sparc_ONLY)dir path=var/svc/manifest/platform/sun4u group=sys
308 $(sparc_ONLY)dir path=var/svc/manifest/platform/sun4v group=sys
309 dir path=var/svc/manifest/site group=sys
310 dir path=var/svc/manifest/system group=sys
311 dir path=var/svc/manifest/system/device group=sys
312 dir path=var/svc/manifest/system/filesystem group=sys
313 dir path=var/svc/manifest/system/security group=sys
314 dir path=var/svc/manifest/system/svc group=sys
315 dir path=var/svc/profile group=sys
316 dir path=var/tmp group=sys mode=1777
317 driver name=dump perms="dump 0660 root sys"
318 driver name=eventfd perms="* 0666 root sys"
319 driver name=fssnap \
320   policy="ctl read_priv_set=sys_config write_priv_set=sys_config" \
321   perms="* 0640 root sys" perms="ctl 0666 root sys"
322 driver name=kstat perms="* 0666 root sys"

```

5

new/usr/src/pkg/manifests/SUNWcs.mf

```

323 driver name=ksyms perms="* 0666 root sys"
324 driver name=logindmux
325 driver name=ptm clone_perms="ptmx 0666 root sys"
326 driver name=pts perms="* 0644 root sys" perms="0 0620 root tty" \
327   perms="1 0620 root tty" perms="2 0620 root tty" perms="3 0620 root tty"
328 driver name=timerfd perms="* 0666 root sys"
329 file path=etc/.login group=sys preserve=renamew
330 file path=etc/cron.d/.proto group=sys mode=0744
331 file path=etc/cron.d/at.deny group=sys preserve=true
332 file path=etc/cron.d/cron.deny group=sys preserve=true
333 file path=etc/cron.d/queuedefs group=sys
334 file path=etc/crypto/kmf.conf group=sys preserve=true
335 file path=etc/crypto/pkcs11.conf group=sys preserve=true
336 file path=etc/datensk group=sys mode=0444
337 file path=etc/default/cron group=sys preserve=true
338 file path=etc/default/devfsadm group=sys preserve=true
339 file path=etc/default/fs group=sys preserve=true
340 file path=etc/default/init group=sys preserve=true
341 file path=etc/default/keyserv group=sys preserve=true
342 file path=etc/default/login group=sys preserve=true
343 file path=etc/default/nss group=sys preserve=true
344 file path=etc/default/passwd group=sys preserve=true
345 file path=etc/default/su group=sys preserve=true
346 file path=etc/default/syslogd group=sys preserve=true
347 file path=etc/default/tar group=sys preserve=true
348 file path=etc/default/useradd group=sys preserve=true
349 file path=etc/default/utmpd group=sys preserve=true
350 file path=etc/dev/reserved_devnames group=sys preserve=true
351 file path=etc/device.tab group=root mode=0444 preserve=true
352 file path=etc/dfs/dfstab group=sys preserve=true
353 file path=etc/dfs/fstypes group=root preserve=true
354 file path=etc/dfs/sharetab group=root mode=0444 preserve=true
355 file path=etc/dgroup.tab group=sys mode=0444 preserve=true
356 file path=etc/dhcp/inittab group=sys preserve=true
357 file path=etc/dhcp/inittab6 group=sys preserve=true
358 file path=etc/dumpdates group=sys mode=0664 preserve=true
359 file path=etc/format.dat group=sys preserve=true
360 file path=etc/fs/dev/mount mode=0555
361 file path=etc/fs/hsfs/mount mode=0555
362 file path=etc/fs/ufs/mount mode=0555
363 file path=etc/ftpd/ftpusers group=sys preserve=true
364 file path=etc/group group=sys preserve=true
365 file path=etc/inet/hosts group=sys preserve=true
366 file path=etc/inet/inetd.conf group=sys preserve=true
367 file path=etc/inet/ipaddrsel.conf group=sys preserve=true
368 file path=etc/inet/netmasks group=sys preserve=true
369 file path=etc/inet/networks group=sys preserve=true
370 file path=etc/inet/protocols group=sys preserve=true
371 file path=etc/inet/services group=sys preserve=true
372 file path=etc/inet/wanboot.conf.sample group=sys mode=0444
373 file path=etc/init.d/PRESERVE group=sys mode=0744 preserve=true
374 file path=etc/init.d/README group=sys preserve=true
375 file path=etc/init.d/syssetup group=sys mode=0744 preserve=true
376 file path=etc/inittab group=sys preserve=true
377 file path=etc/ioctl.syscon group=sys preserve=true
378 file path=etc/ksh.kshrc group=sys preserve=renameold
379 file path=etc/logadm.conf group=sys preserve=true timestamp=19700101T000000Z
380 file path=etc/logindevperm group=sys preserve=true
381 file path=etc/magic mode=0444
382 file path=etc/mail/mailx.rc preserve=true
383 file path=etc/mailcap preserve=true
384 file path=etc/mime.types preserve=true
385 file path=etc/mmttab group=root mode=0444 preserve=true
386 file path=etc/motd group=sys preserve=true
387 file path=etc/net/ticlts/hosts group=sys
388 file path=etc/net/ticlts/services group=sys preserve=true

```

6

```

389 file path=etc/net/ticots/hosts group=sys
390 file path=etc/net/ticots/services group=sys preserve=true
391 file path=etc/net/ticotsord/hosts group=sys
392 file path=etc/net/ticotsord/services group=sys preserve=true
393 file path=etc/netconfig group=sys preserve=true
394 file path=etc/nscd.conf group=sys preserve=true
395 file path=etc/nsswitch.ad group=sys
396 file path=etc/nsswitch.conf group=sys preserve=true
397 file path=etc/nsswitch.dns group=sys
398 file path=etc/nsswitch.files group=sys
399 file path=etc/nsswitch.ldap group=sys
400 file path=etc/pam.conf group=sys preserve=true
401 file path=etc/passwd group=sys preserve=true
402 file path=etc/profile group=sys preserve=renamew
403 file path=etc/project group=sys preserve=true
404 file path=etc/rc2.d/README group=sys
405 file path=etc/rc3.d/README group=sys
406 file path=etc/rcS.d/README group=sys
407 file path=etc/remote preserve=true
408 file path=etc/rpc group=sys preserve=true
409 file path=etc/saf/_sactab group=sys preserve=true
410 file path=etc/saf/_sysconfig group=sys preserve=true
411 file path=etc/saf/zsmon/_pmtab group=sys preserve=true
412 file path=etc/security/audit_class group=sys preserve=renamew
413 file path=etc/security/audit_event group=sys preserve=renamew
414 file path=etc/security/audit_warn group=sys mode=0740 preserve=renamew
415 file path=etc/security/auth_attr group=sys preserve=true \
416     timestamp=19700101T000000Z
417 file path=etc/security/auth_attr.d/SUNWcs group=sys
418 file path=etc/security/crypt.conf group=sys preserve=renamew
419 file path=etc/security/dev/audio mode=0400
420 file path=etc/security/dev/fd0 mode=0400
421 file path=etc/security/dev/sr0 mode=0400
422 file path=etc/security/dev/st0 mode=0400
423 file path=etc/security/dev/st1 mode=0400
424 file path=etc/security/exec_attr group=sys preserve=true \
425     timestamp=19700101T000000Z
426 file path=etc/security/exec_attr.d/SUNWcs group=sys
427 file path=etc/security/kmfpolicy.xml
428 file path=etc/security/lib/audio_clean group=sys mode=0555
429 file path=etc/security/lib/fd_clean group=sys mode=0555
430 file path=etc/security/lib/sr_clean group=sys mode=0555
431 file path=etc/security/lib/st_clean group=sys mode=0555
432 file path=etc/security/policy.conf group=sys preserve=true
433 file path=etc/security/priv_names group=sys preserve=renameold
434 file path=etc/security/prof_attr group=sys preserve=true \
435     timestamp=19700101T000000Z
436 file path=etc/security/prof_attr.d/SUNWcs group=sys
437 file path=etc/shadow group=sys mode=0400 preserve=true
438 file path=etc/skel/.profile group=other preserve=true
439 file path=etc/skel/local.cshrc group=sys preserve=true
440 file path=etc/skel/local.login group=sys preserve=true
441 file path=etc/skel/local.profile group=sys preserve=true
442 file path=etc/svc/profile/generic_limited_net.xml group=sys mode=0444
443 file path=etc/svc/profile/generic_open.xml group=sys mode=0444
444 file path=etc/svc/profile/inetd_generic.xml group=sys mode=0444
445 file path=etc/svc/profile/inetd_upgrade.xml group=sys mode=0444
446 file path=etc/svc/profile/ns_dns.xml group=sys mode=0444
447 file path=etc/svc/profile/ns_files.xml group=sys mode=0444
448 file path=etc/svc/profile/ns_ldap.xml group=sys mode=0444
449 file path=etc/svc/profile/ns_nis.xml group=sys mode=0444
450 file path=etc/svc/profile/ns_none.xml group=sys mode=0444
451 $(sparc_ONLY)file path=etc/svc/profile/platform_SUNW,SPARC-Enterprise.xml \
452     group=sys mode=0444
453 $(sparc_ONLY)file path=etc/svc/profile/platform_SUNW,Sun-Fire-15000.xml \
454     group=sys mode=0444

```

```

455 $(sparc_ONLY)file path=etc/svc/profile/platform_SUNW,Sun-Fire-880.xml \
456     group=sys mode=0444
457 $(sparc_ONLY)file path=etc/svc/profile/platform_SUNW,Sun-Fire.xml group=sys \
458     mode=0444
459 $(sparc_ONLY)file \
460     path=etc/svc/profile/platform_SUNW,Ultra-Enterprise-10000.xml group=sys \
461     mode=0444
462 $(sparc_ONLY)file \
463     path=etc/svc/profile/platform_SUNW,UltraSPARC-III-Netrtract.xml group=sys \
464     mode=0444
465 file path=etc/svc/profile/platform_none.xml group=sys mode=0444
466 $(sparc_ONLY)file path=etc/svc/profile/platform_sun4v.xml group=sys mode=0444
467 file path=etc/sysevent/config/README group=sys mode=0444
468 file path=etc/sysevent/config/SUNW,EC_dr,ESC_dr_req,sysevent.conf group=sys
469 file path=etc/syslog.conf group=sys preserve=true
470 file path=etc/ttydefs group=sys preserve=true
471 file path=etc/ttyrch group=sys preserve=true
472 file path=etc/user_attr group=sys preserve=true timestamp=19700101T000000Z
473 file path=etc/user_attr.d/SUNWcs group=sys
474 file path=etc/vfstab group=sys preserve=true
475 file path=lib/inet/in.mpathd mode=0555
476 file path=lib/inet/impd mode=0555
477 file path=lib/inet/netcfgd mode=0555
478 file path=lib/inet/nwamd mode=0555
479 file path=lib/svc/bin/lsvcrun group=sys mode=0555
480 file path=lib/svc/bin/mfstscan group=sys mode=0555
481 file path=lib/svc/bin/restore_repository group=sys mode=0555
482 file path=lib/svc/bin/sqlite group=sys mode=0555
483 file path=lib/svc/bin/svc.configd group=sys mode=0555
484 file path=lib/svc/bin/svc.ipfd group=sys mode=0555
485 file path=lib/svc/bin/svc.startd group=sys mode=0555
486 file path=lib/svc/manifest/milestone/multi-user-server.xml group=sys mode=0444
487 file path=lib/svc/manifest/milestone/multi-user.xml group=sys mode=0444
488 file path=lib/svc/manifest/milestone/name-services.xml group=sys mode=0444
489 file path=lib/svc/manifest/milestone/network.xml group=sys mode=0444
490 file path=lib/svc/manifest/milestone/single-user.xml group=sys mode=0444
491 file path=lib/svc/manifest/milestone/sysconfig.xml group=sys mode=0444
492 file path=lib/svc/manifest/network/dlmgmt.xml group=sys mode=0444
493 file path=lib/svc/manifest/network/dns/client.xml group=sys mode=0444
494 file path=lib/svc/manifest/network/dns/install.xml group=sys mode=0444
495 file path=lib/svc/manifest/network/forwarding.xml group=sys mode=0444
496 file path=lib/svc/manifest/network/inetd-upgrade.xml group=sys mode=0444
497 file path=lib/svc/manifest/network/inetd.xml group=sys mode=0444
498 file path=lib/svc/manifest/network/ipsec/ike.xml group=sys mode=0444
499 file path=lib/svc/manifest/network/ipsec/ipsecalgs.xml group=sys mode=0444
500 file path=lib/svc/manifest/network/ipsec/manual-key.xml group=sys mode=0444
501 file path=lib/svc/manifest/network/ipsec/policy.xml group=sys mode=0444
502 file path=lib/svc/manifest/network/ldap/client.xml group=sys mode=0444
503 file path=lib/svc/manifest/network/network-initial.xml group=sys mode=0444
504 file path=lib/svc/manifest/network/network-install.xml group=sys mode=0444
505 file path=lib/svc/manifest/network/network-ipmgmt.xml group=sys mode=0444
506 file path=lib/svc/manifest/network/network-ipqos.xml group=sys mode=0444
507 file path=lib/svc/manifest/network/network-iptun.xml group=sys mode=0444
508 file path=lib/svc/manifest/network/network-location.xml group=sys mode=0444
509 file path=lib/svc/manifest/network/network-loopback.xml group=sys mode=0444
510 file path=lib/svc/manifest/network/network-netcfg.xml group=sys mode=0444
511 file path=lib/svc/manifest/network/network-netmask.xml group=sys mode=0444
512 file path=lib/svc/manifest/network/network-physical.xml group=sys mode=0444
513 file path=lib/svc/manifest/network/network-routing-setup.xml group=sys \
514     mode=0444
515 file path=lib/svc/manifest/network/network-service.xml group=sys mode=0444
516 file path=lib/svc/manifest/network/routing/legacy-routing.xml group=sys \
517     mode=0444
518 file path=lib/svc/manifest/network/rpc/bind.xml group=sys mode=0444
519 file path=lib/svc/manifest/network/rpc/keyserv.xml group=sys mode=0444
520 file path=lib/svc/manifest/network/shares/group.xml group=sys mode=0444

```

```

521 file path=lib/svc/manifest/network/shares/reparsed.xml group=sys mode=0444
522 file path=lib/svc/manifest/network/socket-filter-kssl.xml group=sys mode=0444
523 file path=lib/svc/manifest/network/ssl/kssl-proxy.xml group=sys mode=0444
524 file path=lib/svc/manifest/system/auditd.xml group=sys mode=0444
525 file path=lib/svc/manifest/system/auditset.xml group=sys mode=0444
526 file path=lib/svc/manifest/system/boot-archive-update.xml group=sys mode=0444
527 file path=lib/svc/manifest/system/boot-archive.xml group=sys mode=0444
528 file path=lib/svc/manifest/system/boot-config.xml group=sys mode=0444
529 file path=lib/svc/manifest/system/consadm.xml group=sys mode=0444
530 file path=lib/svc/manifest/system/console-login.xml group=sys mode=0444
531 file path=lib/svc/manifest/system/coreadm.xml group=sys mode=0444
532 file path=lib/svc/manifest/system/cron.xml group=sys mode=0444
533 file path=lib/svc/manifest/system/cryptosvc.xml group=sys mode=0444
534 file path=lib/svc/manifest/system/device/allocate.xml group=sys mode=0444
535 file path=lib/svc/manifest/system/device/devices-audio.xml group=sys mode=0444
536 file path=lib/svc/manifest/system/device/devices-local.xml group=sys mode=0444
537 file path=lib/svc/manifest/system/device/mpxio-upgrade.xml group=sys mode=0444
538 file path=lib/svc/manifest/system/early-manifest-import.xml group=sys \
539 mode=0444
540 file path=lib/svc/manifest/system/extended-accounting.xml group=sys mode=0444
541 file path=lib/svc/manifest/system/filesystem/local-fs.xml group=sys mode=0444
542 file path=lib/svc/manifest/system/filesystem/minimal-fs.xml group=sys \
543 mode=0444
544 file path=lib/svc/manifest/system/filesystem/root-fs.xml group=sys mode=0444
545 file path=lib/svc/manifest/system/filesystem/usr-fs.xml group=sys mode=0444
546 $(i386_ONLY)file path=lib/svc/manifest/system/hostid.xml group=sys mode=0444
547 file path=lib/svc/manifest/system/hotplug.xml group=sys mode=0444
548 file path=lib/svc/manifest/system/identity.xml group=sys mode=0444
549 file path=lib/svc/manifest/system/idmap.xml group=sys mode=0444
550 file path=lib/svc/manifest/system/keymap.xml group=sys mode=0444
551 file path=lib/svc/manifest/system/logadm-upgrade.xml group=sys mode=0444
552 file path=lib/svc/manifest/system/manifest-import.xml group=sys mode=0444
553 file path=lib/svc/manifest/system/name-service-cache.xml group=sys mode=0444
554 file path=lib/svc/manifest/system/pfexecd.xml group=sys mode=0444
555 file path=lib/svc/manifest/system/process-security.xml group=sys mode=0444
556 #endif /* ! codereview */
557 file path=lib/svc/manifest/system/rbac.xml group=sys mode=0444
558 file path=lib/svc/manifest/system/rmtmpfiles.xml group=sys mode=0444
559 file path=lib/svc/manifest/system/sac.xml group=sys mode=0444
560 file path=lib/svc/manifest/system/svc/global.xml group=sys mode=0444
561 file path=lib/svc/manifest/system/svc/restarter.xml group=sys mode=0444
562 file path=lib/svc/manifest/system/system-log.xml group=sys mode=0444
563 file path=lib/svc/manifest/system/utmp.xml group=sys mode=0444
564 file path=lib/svc/manifest/system/vtdaemon.xml group=sys mode=0444
565 file path=lib/svc/method/boot-archive mode=0555
566 file path=lib/svc/method/boot-archive-update mode=0555
567 file path=lib/svc/method/console-login mode=0555
568 file path=lib/svc/method/devices-audio mode=0555
569 file path=lib/svc/method/devices-local mode=0555
570 file path=lib/svc/method/dns-install mode=0555
571 file path=lib/svc/method/fs-local mode=0555
572 file path=lib/svc/method/fs-minimal mode=0555
573 file path=lib/svc/method/fs-root mode=0555
574 file path=lib/svc/method/fs-usr mode=0555
575 file path=lib/svc/method/identity-domain mode=0555
576 file path=lib/svc/method/identity-node mode=0555
577 file path=lib/svc/method/inetd-upgrade mode=0555
578 file path=lib/svc/method/keymap mode=0555
579 file path=lib/svc/method/ldap-client mode=0555
580 file path=lib/svc/method/logadm-upgrade mode=0555
581 file path=lib/svc/method/manifest-import mode=0555
582 file path=lib/svc/method/mpxio-upgrade mode=0555
583 file path=lib/svc/method/net-init mode=0555
584 file path=lib/svc/method/net-install mode=0555
585 file path=lib/svc/method/net-ipmgmt mode=0555
586 file path=lib/svc/method/net-ipqos mode=0555

```

```

587 file path=lib/svc/method/net-iptun mode=0555
588 file path=lib/svc/method/net-loc mode=0555
589 file path=lib/svc/method/net-loopback mode=0555
590 file path=lib/svc/method/net-netmask mode=0555
591 file path=lib/svc/method/net-nwam mode=0555
592 file path=lib/svc/method/net-physical mode=0555
593 file path=lib/svc/method/net-routing-setup mode=0555
594 file path=lib/svc/method/net-svc mode=0555
595 file path=lib/svc/method/rmtmpfiles mode=0555
596 file path=lib/svc/method/rpc-bind mode=0555
597 file path=lib/svc/method/svc-allocate mode=0555
598 file path=lib/svc/method/svc-auditd mode=0555
599 file path=lib/svc/method/svc-auditset mode=0555
600 file path=lib/svc/method/svc-boot-config mode=0555
601 file path=lib/svc/method/svc-consadm mode=0555
602 file path=lib/svc/method/svc-cron mode=0555
603 file path=lib/svc/method/svc-dlmgmt mode=0555
604 file path=lib/svc/method/svc-forwarding mode=0555
605 $(i386_ONLY)file path=lib/svc/method/svc-hostid mode=0555
606 file path=lib/svc/method/svc-hotplug mode=0555
607 file path=lib/svc/method/svc-legacy-routing mode=0555
608 file path=lib/svc/method/svc-nscd mode=0555
609 file path=lib/svc/method/svc-rbac mode=0555
610 file path=lib/svc/method/svc-sockfilter mode=0555
611 file path=lib/svc/method/svc-utmpd mode=0555
612 file path=lib/svc/method/system-log mode=0555
613 file path=lib/svc/method/vtdaemon mode=0555
614 file path=lib/svc/method/yp mode=0555
615 # global.db is not needed in non-global zones, and it's pretty large.
616 file path=lib/svc/seed/global.db group=sys mode=0444 \
617 variant.opensolaris.zone=global
618 # symmetrically, nonglobal.db is not needed in global zones.
619 file path=lib/svc/seed/nonglobal.db group=sys mode=0444 \
620 variant.opensolaris.zone=nonglobal
621 file path=lib/svc/share/README mode=0444
622 file path=lib/svc/share/fs_include.sh mode=0444
623 file path=lib/svc/share/ipf_include.sh mode=0444
624 file path=lib/svc/share/mfsthistory mode=0444
625 file path=lib/svc/share/net_include.sh mode=0444
626 file path=lib/svc/share/routing_include.sh mode=0444
627 file path=lib/svc/share/smf_include.sh mode=0444
628 file path=root/.bashrc group=root preserve=true
629 file path=root/.profile group=root preserve=true
630 file path=sbin/autopush mode=0555
631 $(i386_ONLY)file path=sbin/biosdev mode=0555
632 file path=sbin/bootadm mode=0555
633 file path=sbin/cryptoadm mode=0555
634 file path=sbin/devprop mode=0555
635 file path=sbin/dhcpagent mode=0555
636 file path=sbin/dhcpinfo mode=0555
637 file path=sbin/dlmgmt mode=0555
638 file path=sbin/fdisk mode=0555
639 file path=sbin/fiocompress mode=0555
640 file path=sbin/hostconfig mode=0555
641 file path=sbin/ifconfig mode=0555
642 file path=sbin/ifparse mode=0555
643 file path=sbin/init group=sys mode=0555
644 $(i386_ONLY)file path=sbin/installgrub group=sys mode=0555
645 file path=sbin/impstat mode=0555
646 file path=sbin/mount mode=0555
647 file path=sbin/mountall group=sys mode=0555
648 file path=sbin/netstrategy mode=0555
649 file path=sbin/rc0 group=sys mode=0744
650 file path=sbin/rc1 group=sys mode=0744
651 file path=sbin/rc2 group=sys mode=0744
652 file path=sbin/rc3 group=sys mode=0744

```

```

653 file path=sbin/rcS group=sys mode=0744
654 file path=sbin/route mode=0555
655 file path=sbin/routeadm mode=0555
656 file path=sbin/soconfig mode=0555
657 file path=sbin/su.static group=sys mode=0555
658 file path=sbin/sulogin mode=0555
659 file path=sbin/swapadd group=sys mode=0744
660 file path=sbin/sync mode=0555
661 file path=sbin/tzreload mode=0555
662 file path=sbin/uadmin group=sys mode=0555
663 file path=sbin/umount mode=0555
664 file path=sbin/umountall group=sys mode=0555
665 file path=sbin/uname mode=0555
666 file path=sbin/zonename mode=0555
667 $(i386_ONLY)file path=usr/bin/$(ARCH32)/amt mode=0555
668 file path=usr/bin/$(ARCH32)/decrypt mode=0555
669 file path=usr/bin/$(ARCH32)/digest mode=0555
670 file path=usr/bin/$(ARCH32)/ksh93 mode=0555
671 $(i386_ONLY)file path=usr/bin/$(ARCH32)/newtask group=sys mode=4555
672 $(i386_ONLY)file path=usr/bin/$(ARCH32)/nohup mode=0555
673 $(i386_ONLY)file path=usr/bin/$(ARCH32)/prctl mode=0555
674 $(i386_ONLY)file path=usr/bin/$(ARCH32)/prstat mode=0555
675 $(i386_ONLY)file path=usr/bin/$(ARCH32)/ps mode=0555
676 file path=usr/bin/$(ARCH32)/savecore mode=0555
677 $(i386_ONLY)file path=usr/bin/$(ARCH32)/setuname mode=0555
678 $(i386_ONLY)file path=usr/bin/$(ARCH32)/uptime mode=4555
679 file path=usr/bin/$(ARCH64)/amt mode=0555
680 file path=usr/bin/$(ARCH64)/crle mode=0555
681 file path=usr/bin/$(ARCH64)/decrypt mode=0555
682 file path=usr/bin/$(ARCH64)/digest mode=0555
683 file path=usr/bin/$(ARCH64)/ksh93 mode=0555
684 file path=usr/bin/$(ARCH64)/ls mode=0555
685 file path=usr/bin/$(ARCH64)/moe mode=0555
686 file path=usr/bin/$(ARCH64)/newtask group=sys mode=4555
687 file path=usr/bin/$(ARCH64)/nohup mode=0555
688 file path=usr/bin/$(ARCH64)/prctl mode=0555
689 file path=usr/bin/$(ARCH64)/prstat mode=0555
690 file path=usr/bin/$(ARCH64)/ps mode=0555
691 file path=usr/bin/$(ARCH64)/savecore mode=0555
692 file path=usr/bin/$(ARCH64)/setuname mode=0555
693 file path=usr/bin/$(ARCH64)/uptime mode=4555
694 $(i386_ONLY)file path=usr/bin/addbadsec mode=0555
695 file path=usr/bin/alias mode=0555
696 file path=usr/bin/amt mode=0555
697 file path=usr/bin/arch mode=0555
698 file path=usr/bin/at group=sys mode=4755
699 file path=usr/bin/atq group=sys mode=4755
700 file path=usr/bin/atrm group=sys mode=4755
701 file path=usr/bin/auths mode=0555
702 file path=usr/bin/basename mode=0555
703 file path=usr/bin/busstat mode=0555
704 file path=usr/bin/captainfo mode=0555
705 file path=usr/bin/cat mode=0555
706 file path=usr/bin/chgrp mode=0555
707 file path=usr/bin/chmod mode=0555
708 file path=usr/bin/chown mode=0555
709 file path=usr/bin/ckdate mode=0555
710 file path=usr/bin/ckgid mode=0555
711 file path=usr/bin/ckint mode=0555
712 file path=usr/bin/ckitem mode=0555
713 file path=usr/bin/ckkeywd mode=0555
714 file path=usr/bin/ckpath mode=0555
715 file path=usr/bin/ckrange mode=0555
716 file path=usr/bin/ckstr mode=0555
717 file path=usr/bin/cktime mode=0555
718 file path=usr/bin/ckuid mode=0555

```

```

719 file path=usr/bin/ckyorn mode=0555
720 file path=usr/bin/clear mode=0555
721 file path=usr/bin/coreadm mode=0555
722 file path=usr/bin/cp mode=0555
723 file path=usr/bin/cpio mode=0555
724 file path=usr/bin/crle mode=0555
725 file path=usr/bin/crontab mode=4555
726 file path=usr/bin/crypt mode=0555
727 file path=usr/bin/csh mode=0555
728 file path=usr/bin/ctrun mode=0555
729 file path=usr/bin/ctstat mode=0555
730 file path=usr/bin/ctwatch mode=0555
731 file path=usr/bin/date mode=0555
732 file path=usr/bin/dd mode=0555
733 file path=usr/bin/devattr mode=0555
734 file path=usr/bin/devfree mode=0555
735 file path=usr/bin/devreserv mode=0555
736 file path=usr/bin/dirname mode=0555
737 $(i386_ONLY)file path=usr/bin/diskscan mode=0555
738 file path=usr/bin/domainname mode=0555
739 file path=usr/bin/du mode=0555
740 file path=usr/bin/dumpcs mode=0555
741 file path=usr/bin/dumpkeys mode=0555
742 file path=usr/bin/echo mode=0555
743 file path=usr/bin/ed mode=0555
744 file path=usr/bin/egrep mode=0555
745 file path=usr/bin/eject mode=0555
746 file path=usr/bin/env mode=0555
747 file path=usr/bin/expr mode=0555
748 file path=usr/bin/false mode=0555
749 file path=usr/bin/fdetach mode=0555
750 file path=usr/bin/fdformat mode=4555
751 file path=usr/bin/fgrep mode=0555
752 file path=usr/bin/file mode=0555
753 file path=usr/bin/find mode=0555
754 file path=usr/bin/fmt mode=0555
755 file path=usr/bin/fmtmsg mode=0555
756 file path=usr/bin/fold mode=0555
757 file path=usr/bin/fsstat mode=0555
758 file path=usr/bin/geniconvtbl mode=0555
759 file path=usr/bin/getconf mode=0555
760 file path=usr/bin/getdev mode=0555
761 file path=usr/bin/getdgrp mode=0555
762 file path=usr/bin/getent mode=0555
763 file path=usr/bin/getfacl mode=0555
764 file path=usr/bin/getopt mode=0555
765 file path=usr/bin/gettext mode=0555
766 file path=usr/bin/getvol mode=0555
767 file path=usr/bin/grep mode=0555
768 file path=usr/bin/groups mode=0555
769 file path=usr/bin/head mode=0555
770 file path=usr/bin/hostid mode=0555
771 file path=usr/bin/hostname mode=0555
772 file path=usr/bin/iconv mode=0555
773 file path=usr/bin/id mode=0555
774 file path=usr/bin/infocmp mode=0555
775 file path=usr/bin/iostat mode=0555
776 file path=usr/bin/isainfo mode=0555
777 file path=usr/bin/isalist mode=0555
778 file path=usr/bin/kbd mode=0555
779 file path=usr/bin/keylogin mode=0555
780 file path=usr/bin/keylogout mode=0555
781 file path=usr/bin/kmfcfg mode=0555
782 file path=usr/bin/kvmstat mode=0555
783 file path=usr/bin/line mode=0555
784 file path=usr/bin/listdgrp mode=0555

```

```

785 file path=usr/bin/listusers mode=0555
786 file path=usr/bin/loadkeys mode=0555
787 file path=usr/bin/logger mode=0555
788 file path=usr/bin/login mode=4555
789 file path=usr/bin/logins mode=0750
790 file path=usr/bin/ls mode=0555
791 file path=usr/bin/m4 mode=0555
792 file path=usr/bin/mach mode=0555
793 file path=usr/bin/mail group=mail mode=2511
794 file path=usr/bin/mailx group=mail mode=2511
795 file path=usr/bin/makedev mode=0555
796 file path=usr/bin/mesg mode=0555
797 file path=usr/bin/mkdir mode=0555
798 file path=usr/bin/mkpwdict mode=0555
799 file path=usr/bin/mktemp mode=0555
800 file path=usr/bin/moe mode=0555
801 file path=usr/bin/more mode=0555
802 file path=usr/bin/mpstat mode=0555
803 file path=usr/bin/mt mode=0555
804 file path=usr/bin/netstat mode=0555
805 file path=usr/bin/newgrp group=sys mode=4755
806 file path=usr/bin/nice mode=0555
807 file path=usr/bin/optisa mode=0555
808 file path=usr/bin/pagesize mode=0555
809 file path=usr/bin/passwd group=sys mode=6555
810 file path=usr/bin/pathchk mode=0555
811 file path=usr/bin/pax mode=0555
812 file path=usr/bin/pfexec mode=0555
813 file path=usr/bin/pg mode=0555
814 file path=usr/bin/pgrep mode=0555
815 file path=usr/bin/pktool mode=0555
816 file path=usr/bin/pr mode=0555
817 file path=usr/bin/printf mode=0555
818 file path=usr/bin/pricntnl mode=0555
819 file path=usr/bin/profiles mode=0555
820 file path=usr/bin/projects mode=0555
821 file path=usr/bin/putdev mode=0555
822 file path=usr/bin/putdgrp mode=0555
823 file path=usr/bin/pwd mode=0555
824 file path=usr/bin/renice mode=0555
825 file path=usr/bin/rm mode=0555
826 file path=usr/bin/rmdir mode=0555
827 file path=usr/bin/roles mode=0555
828 file path=usr/bin/rpcinfo mode=0555
829 file path=usr/bin/runat mode=0555
830 file path=usr/bin/script mode=0555
831 file path=usr/bin/sed mode=0555
832 file path=usr/bin/setfacl mode=0555
833 file path=usr/bin/setpgrp group=sys mode=0555
834 file path=usr/bin/settime mode=0555
835 file path=usr/bin/shcomp mode=0555
836 file path=usr/bin/strchg group=root mode=0555
837 file path=usr/bin/strconf group=root mode=0555
838 file path=usr/bin/stty mode=0555
839 file path=usr/bin/su group=sys mode=4555
840 file path=usr/bin/svcprop mode=0555
841 file path=usr/bin/svcs mode=0555
842 file path=usr/bin/tabs mode=0555
843 file path=usr/bin/tail mode=0555
844 file path=usr/bin/tic mode=0555
845 file path=usr/bin/time mode=0555
846 file path=usr/bin/tip mode=4511 owner=uucp
847 file path=usr/bin/tpmadm mode=0555
848 file path=usr/bin/tput mode=0555
849 file path=usr/bin/tr mode=0555
850 file path=usr/bin/true mode=0555

```

```

851 file path=usr/bin/tty mode=0555
852 file path=usr/bin/tzselect mode=0555
853 file path=usr/bin/userattr mode=0555
854 file path=usr/bin/uuidgen mode=0555
855 file path=usr/bin/vmstat mode=0555
856 file path=usr/bin/which mode=0555
857 file path=usr/bin/who mode=0555
858 file path=usr/bin/wracct mode=0555
859 file path=usr/bin/write group=tty mode=2555
860 file path=usr/bin/xargs mode=0555
861 file path=usr/bin/xstr mode=0555
862 file path=usr/has/bin/edit mode=0555
863 file path=usr/has/bin/sh mode=0555
864 file path=usr/has/man/man1has/edit.lhas
865 file path=usr/has/man/man1has/ex.lhas
866 file path=usr/has/man/man1has/sh.lhas
867 file path=usr/has/man/man1has/vi.lhas
868 file path=usr/kernel/drv/$(ARCH64)/dump group=sys
869 file path=usr/kernel/drv/$(ARCH64)/eventfd group=sys
870 file path=usr/kernel/drv/$(ARCH64)/fssnap group=sys
871 file path=usr/kernel/drv/$(ARCH64)/kstat group=sys
872 file path=usr/kernel/drv/$(ARCH64)/ksyms group=sys
873 file path=usr/kernel/drv/$(ARCH64)/logindmux group=sys
874 file path=usr/kernel/drv/$(ARCH64)/ptm group=sys
875 file path=usr/kernel/drv/$(ARCH64)/pts group=sys
876 file path=usr/kernel/drv/$(ARCH64)/timerfd group=sys
877 $(i386_ONLY)file path=usr/kernel/drv/dump group=sys
878 file path=usr/kernel/drv/dump.conf group=sys
879 $(i386_ONLY)file path=usr/kernel/drv/eventfd group=sys
880 file path=usr/kernel/drv/eventfd.conf group=sys
881 $(i386_ONLY)file path=usr/kernel/drv/fssnap group=sys
882 file path=usr/kernel/drv/fssnap.conf group=sys
883 $(i386_ONLY)file path=usr/kernel/drv/kstat group=sys
884 file path=usr/kernel/drv/kstat.conf group=sys
885 $(i386_ONLY)file path=usr/kernel/drv/ksyms group=sys
886 file path=usr/kernel/drv/ksyms.conf group=sys
887 $(i386_ONLY)file path=usr/kernel/drv/logindmux group=sys
888 file path=usr/kernel/drv/logindmux.conf group=sys
889 $(i386_ONLY)file path=usr/kernel/drv/ptm group=sys
890 file path=usr/kernel/drv/ptm.conf group=sys
891 $(i386_ONLY)file path=usr/kernel/drv/pts group=sys
892 file path=usr/kernel/drv/pts.conf group=sys
893 $(i386_ONLY)file path=usr/kernel/drv/timerfd group=sys
894 file path=usr/kernel/drv/timerfd.conf group=sys
895 file path=usr/kernel/exec/$(ARCH64)/javaexec group=sys mode=0755
896 file path=usr/kernel/exec/$(ARCH64)/shbinexec group=sys mode=0755
897 $(i386_ONLY)file path=usr/kernel/exec/javaexec group=sys mode=0755
898 $(i386_ONLY)file path=usr/kernel/exec/shbinexec group=sys mode=0755
899 file path=usr/kernel/fs/$(ARCH64)/fdfs group=sys mode=0755
900 file path=usr/kernel/fs/$(ARCH64)/pcfs group=sys mode=0755
901 $(i386_ONLY)file path=usr/kernel/fs/fdfs group=sys mode=0755
902 $(i386_ONLY)file path=usr/kernel/fs/pcfs group=sys mode=0755
903 file path=usr/kernel/sched/$(ARCH64)/FX group=sys mode=0755
904 file path=usr/kernel/sched/$(ARCH64)/FX_DPTBL group=sys mode=0755
905 file path=usr/kernel/sched/$(ARCH64)/IA group=sys mode=0755
906 file path=usr/kernel/sched/$(ARCH64)/RT group=sys mode=0755
907 file path=usr/kernel/sched/$(ARCH64)/RT_DPTBL group=sys mode=0755
908 $(i386_ONLY)file path=usr/kernel/sched/FX group=sys mode=0755
909 $(i386_ONLY)file path=usr/kernel/sched/FX_DPTBL group=sys mode=0755
910 $(i386_ONLY)file path=usr/kernel/sched/IA group=sys mode=0755
911 $(i386_ONLY)file path=usr/kernel/sched/RT group=sys mode=0755
912 $(i386_ONLY)file path=usr/kernel/sched/RT_DPTBL group=sys mode=0755
913 file path=usr/kernel/strmod/$(ARCH64)/cryptmod group=sys mode=0755
914 file path=usr/kernel/strmod/$(ARCH64)/r1mod group=sys mode=0755
915 file path=usr/kernel/strmod/$(ARCH64)/telmod group=sys mode=0755
916 $(i386_ONLY)file path=usr/kernel/strmod/cryptmod group=sys mode=0755

```

```

917 $(i386_ONLY)file path=usr/kernel/strmod/rlmod group=sys mode=0755
918 $(i386_ONLY)file path=usr/kernel/strmod/telmod group=sys mode=0755
919 file path=usr/kernel/sys/$(ARCH64)/acctctl group=sys mode=0755
920 file path=usr/kernel/sys/$(ARCH64)/exaccts group=sys mode=0755
921 file path=usr/kernel/sys/$(ARCH64)/sysacct group=sys mode=0755
922 $(i386_ONLY)file path=usr/kernel/sys/acctctl group=sys mode=0755
923 $(i386_ONLY)file path=usr/kernel/sys/exaccts group=sys mode=0755
924 $(i386_ONLY)file path=usr/kernel/sys/sysacct group=sys mode=0755
925 file path=usr/kvm/README group=sys
926 file path=usr/lib/$(ARCH64)/libshare.so.1
927 file path=usr/lib/audit/audit_record_attr mode=0444
928 file path=usr/lib/calprog mode=0555
929 file path=usr/lib/class/FX/FXdispadmin mode=0555
930 file path=usr/lib/class/FX/FXprioctl mode=0555
931 file path=usr/lib/class/IA/IAdispadmin mode=0555
932 file path=usr/lib/class/IA/IAprioctl mode=0555
933 file path=usr/lib/class/RT/RTdispadmin mode=0555
934 file path=usr/lib/class/RT/RTprioctl mode=0555
935 file path=usr/lib/class/SDC/SDCdispadmin mode=0555
936 file path=usr/lib/class/SDC/SDCprioctl mode=0555
937 file path=usr/lib/class/TS/TSdispadmin mode=0555
938 file path=usr/lib/class/TS/TSprioctl mode=0555
939 file path=usr/lib/devfsadm/linkmod/SUNW_audio_link.so group=sys
940 file path=usr/lib/devfsadm/linkmod/SUNW_cfg_link.so group=sys
941 file path=usr/lib/devfsadm/linkmod/SUNW_disk_link.so group=sys
942 file path=usr/lib/devfsadm/linkmod/SUNW_fssnap_link.so group=sys
943 file path=usr/lib/devfsadm/linkmod/SUNW_ieee1394_link.so group=sys
944 file path=usr/lib/devfsadm/linkmod/SUNW_lofi_link.so group=sys
945 file path=usr/lib/devfsadm/linkmod/SUNW_md_link.so group=sys
946 file path=usr/lib/devfsadm/linkmod/SUNW_misc_link.so group=sys
947 file path=usr/lib/devfsadm/linkmod/SUNW_misc_link.$(ARCH).so group=sys
948 file path=usr/lib/devfsadm/linkmod/SUNW_port_link.so group=sys
949 file path=usr/lib/devfsadm/linkmod/SUNW_ramdisk_link.so group=sys
950 file path=usr/lib/devfsadm/linkmod/SUNW_sgen_link.so group=sys
951 file path=usr/lib/devfsadm/linkmod/SUNW_smp_link.so group=sys
952 file path=usr/lib/devfsadm/linkmod/SUNW_tape_link.so group=sys
953 file path=usr/lib/devfsadm/linkmod/SUNW_usb_link.so group=sys
954 $(i386_ONLY)file path=usr/lib/devfsadm/linkmod/SUNW_xen_link.so group=sys
955 file path=usr/lib/diffh mode=0555
956 file path=usr/lib/expreserve mode=0555
957 file path=usr/lib/exrecovery mode=0555
958 file path=usr/lib/fs/bootfs/mount mode=0555
959 file path=usr/lib/fs/ctfs/mount mode=0555
960 file path=usr/lib/fs/fd/mount mode=0555
961 file path=usr/lib/fs/hfs/fstyp.so.1 mode=0555
962 file path=usr/lib/fs/hfs/labelit mode=0555
963 file path=usr/lib/fs/lofs/mount mode=0555
964 file path=usr/lib/fs/mntfs/mount mode=0555
965 file path=usr/lib/fs/objfs/mount mode=0555
966 file path=usr/lib/fs/proc/mount mode=0555
967 file path=usr/lib/fs/shares/mount mode=0555
968 file path=usr/lib/fs/tmpfs/mount mode=0555
969 file path=usr/lib/fs/ufs/clri mode=0555
970 file path=usr/lib/fs/ufs/df mode=0555
971 file path=usr/lib/fs/ufs/edquota mode=0555
972 file path=usr/lib/fs/ufs/ff mode=0555
973 file path=usr/lib/fs/ufs/fsck mode=0555
974 file path=usr/lib/fs/ufs/fsckall mode=0555
975 file path=usr/lib/fs/ufs/fsdb mode=0555
976 file path=usr/lib/fs/ufs/fsirand mode=0555
977 file path=usr/lib/fs/ufs/fssnap mode=0555
978 file path=usr/lib/fs/ufs/fstyp.so.1 mode=0555
979 file path=usr/lib/fs/ufs/labelit mode=0555
980 file path=usr/lib/fs/ufs/lockfs mode=0555
981 file path=usr/lib/fs/ufs/mkfs mode=0555
982 file path=usr/lib/fs/ufs/ncheck mode=0555

```

```

983 file path=usr/lib/fs/ufs/newfs mode=0555
984 file path=usr/lib/fs/ufs/quot mode=0555
985 file path=usr/lib/fs/ufs/quota mode=4555
986 file path=usr/lib/fs/ufs/quotacheck mode=0555
987 file path=usr/lib/fs/ufs/quotaoff mode=0555
988 file path=usr/lib/fs/ufs/repquota mode=0555
989 file path=usr/lib/fs/ufs/tunefs mode=0555
990 file path=usr/lib/fs/ufs/ufsdump mode=4555
991 file path=usr/lib/fs/ufs/ufsrestore mode=4555
992 file path=usr/lib/fs/ufs/volcopy mode=0555
993 file path=usr/lib/getoptcvt mode=0555
994 file path=usr/lib/help/auths/locale/C/AllSolAuthsHeader.html
995 file path=usr/lib/help/auths/locale/C/AuditHeader.html
996 file path=usr/lib/help/auths/locale/C/AuthJobsAdmin.html
997 file path=usr/lib/help/auths/locale/C/AuthJobsUser.html
998 file path=usr/lib/help/auths/locale/C/AuthProfmgrAssign.html
999 file path=usr/lib/help/auths/locale/C/AuthProfmgrDelegate.html
1000 file path=usr/lib/help/auths/locale/C/AuthProfmgrExecattrWrite.html
1001 file path=usr/lib/help/auths/locale/C/AuthProfmgrRead.html
1002 file path=usr/lib/help/auths/locale/C/AuthProfmgrWrite.html
1003 file path=usr/lib/help/auths/locale/C/AuthReadNDMP.html
1004 file path=usr/lib/help/auths/locale/C/AuthReadSMB.html
1005 file path=usr/lib/help/auths/locale/C/AuthRoleAssign.html
1006 file path=usr/lib/help/auths/locale/C/AuthRoleDelegate.html
1007 file path=usr/lib/help/auths/locale/C/AuthRoleWrite.html
1008 file path=usr/lib/help/auths/locale/C/BindStates.html
1009 file path=usr/lib/help/auths/locale/C/DevAllocHeader.html
1010 file path=usr/lib/help/auths/locale/C/DevAllocate.html
1011 file path=usr/lib/help/auths/locale/C/DevConfig.html
1012 file path=usr/lib/help/auths/locale/C/DevGrant.html
1013 file path=usr/lib/help/auths/locale/C/DevRevoke.html
1014 file path=usr/lib/help/auths/locale/C/DhccpMgrHeader.html
1015 file path=usr/lib/help/auths/locale/C/DhccpMgrWrite.html
1016 file path=usr/lib/help/auths/locale/C/HotplugHeader.html
1017 file path=usr/lib/help/auths/locale/C/HotplugModifier.html
1018 file path=usr/lib/help/auths/locale/C/IdmapRules.html
1019 file path=usr/lib/help/auths/locale/C/JobHeader.html
1020 file path=usr/lib/help/auths/locale/C/JobGrant.html
1021 file path=usr/lib/help/auths/locale/C/LinkSecurity.html
1022 file path=usr/lib/help/auths/locale/C/LoginEnable.html
1023 file path=usr/lib/help/auths/locale/C/LoginHeader.html
1024 file path=usr/lib/help/auths/locale/C/LoginRemote.html
1025 file path=usr/lib/help/auths/locale/C/NetworkAutoconfRead.html
1026 file path=usr/lib/help/auths/locale/C/NetworkAutoconfSelect.html
1027 file path=usr/lib/help/auths/locale/C/NetworkAutoconfWlan.html
1028 file path=usr/lib/help/auths/locale/C/NetworkAutoconfWrite.html
1029 file path=usr/lib/help/auths/locale/C/NetworkHeader.html
1030 file path=usr/lib/help/auths/locale/C/NetworkILBconf.html
1031 file path=usr/lib/help/auths/locale/C/NetworkILBenable.html
1032 file path=usr/lib/help/auths/locale/C/NetworkInterfaceConfig.html
1033 file path=usr/lib/help/auths/locale/C/NetworkVRRP.html
1034 file path=usr/lib/help/auths/locale/C/PriAdmin.html
1035 file path=usr/lib/help/auths/locale/C/ProfmgrHeader.html
1036 file path=usr/lib/help/auths/locale/C/RoleHeader.html
1037 file path=usr/lib/help/auths/locale/C/SmfAllocate.html
1038 file path=usr/lib/help/auths/locale/C/SmfAutofsStates.html
1039 file path=usr/lib/help/auths/locale/C/SmfCoreadmStates.html
1040 file path=usr/lib/help/auths/locale/C/SmfCronStates.html
1041 file path=usr/lib/help/auths/locale/C/SmfExacctFlowStates.html
1042 file path=usr/lib/help/auths/locale/C/SmfExacctNetStates.html
1043 file path=usr/lib/help/auths/locale/C/SmfExacctProcessStates.html
1044 file path=usr/lib/help/auths/locale/C/SmfExacctTaskStates.html
1045 file path=usr/lib/help/auths/locale/C/SmfHeader.html
1046 file path=usr/lib/help/auths/locale/C/SmfILBStates.html
1047 file path=usr/lib/help/auths/locale/C/SmfIPsecStates.html
1048 file path=usr/lib/help/auths/locale/C/SmfIdmapStates.html

```

```

1049 file path=usr/lib/help/auths/locale/C/SmfInetdStates.html
1050 file path=usr/lib/help/auths/locale/C/SmfLocationStates.html
1051 file path=usr/lib/help/auths/locale/C/SmfMDNSStates.html
1052 file path=usr/lib/help/auths/locale/C/SmfManageAudit.html
1053 file path=usr/lib/help/auths/locale/C/SmfManageHeader.html
1054 file path=usr/lib/help/auths/locale/C/SmfManageHotplug.html
1055 file path=usr/lib/help/auths/locale/C/SmfManageZFSnap.html
1056 file path=usr/lib/help/auths/locale/C/SmfModifyAppl.html
1057 file path=usr/lib/help/auths/locale/C/SmfModifyDepend.html
1058 file path=usr/lib/help/auths/locale/C/SmfModifyFramework.html
1059 file path=usr/lib/help/auths/locale/C/SmfModifyHeader.html
1060 file path=usr/lib/help/auths/locale/C/SmfModifyMethod.html
1061 file path=usr/lib/help/auths/locale/C/SmfNADDStates.html
1062 file path=usr/lib/help/auths/locale/C/SmfNDMPStates.html
1063 file path=usr/lib/help/auths/locale/C/SmfNWAMStates.html
1064 file path=usr/lib/help/auths/locale/C/SmfNscdStates.html
1065 file path=usr/lib/help/auths/locale/C/SmfPowerStates.html
1066 file path=usr/lib/help/auths/locale/C/SmfReparseStates.html
1067 file path=usr/lib/help/auths/locale/C/SmfRoutingStates.html
1068 file path=usr/lib/help/auths/locale/C/SmfSMBFSStates.html
1069 file path=usr/lib/help/auths/locale/C/SmfSMBStates.html
1070 file path=usr/lib/help/auths/locale/C/SmfSendmailStates.html
1071 file path=usr/lib/help/auths/locale/C/SmfSshStates.html
1072 file path=usr/lib/help/auths/locale/C/SmfSyslogStates.html
1073 file path=usr/lib/help/auths/locale/C/SmfVRRPStates.html
1074 file path=usr/lib/help/auths/locale/C/SmfValueAudit.html
1075 file path=usr/lib/help/auths/locale/C/SmfValueCoreadm.html
1076 file path=usr/lib/help/auths/locale/C/SmfValueExAcctFlow.html
1077 file path=usr/lib/help/auths/locale/C/SmfValueExAcctNet.html
1078 file path=usr/lib/help/auths/locale/C/SmfValueExAcctProcess.html
1079 file path=usr/lib/help/auths/locale/C/SmfValueExAcctTask.html
1080 file path=usr/lib/help/auths/locale/C/SmfValueFirewall.html
1081 file path=usr/lib/help/auths/locale/C/SmfValueHeader.html
1082 file path=usr/lib/help/auths/locale/C/SmfValueIPsec.html
1083 file path=usr/lib/help/auths/locale/C/SmfValueIdmap.html
1084 file path=usr/lib/help/auths/locale/C/SmfValueInetd.html
1085 file path=usr/lib/help/auths/locale/C/SmfValueMDNS.html
1086 file path=usr/lib/help/auths/locale/C/SmfValueNADD.html
1087 file path=usr/lib/help/auths/locale/C/SmfValueNDMP.html
1088 file path=usr/lib/help/auths/locale/C/SmfValueNWAM.html
1089 file path=usr/lib/help/auths/locale/C/SmfValueProcSec.html
1090 #endif /* ! codereview */
1091 file path=usr/lib/help/auths/locale/C/SmfValueRouting.html
1092 file path=usr/lib/help/auths/locale/C/SmfValueSMB.html
1093 file path=usr/lib/help/auths/locale/C/SmfValueVscan.html
1094 file path=usr/lib/help/auths/locale/C/SmfValueVt.html
1095 file path=usr/lib/help/auths/locale/C/SmfVscanStates.html
1096 file path=usr/lib/help/auths/locale/C/SmfVtStates.html
1097 file path=usr/lib/help/auths/locale/C/SmfWpaStates.html
1098 file path=usr/lib/help/auths/locale/C/SysCpuPowerMgmt.html
1099 file path=usr/lib/help/auths/locale/C/SysDate.html
1100 file path=usr/lib/help/auths/locale/C/SysHeader.html
1101 file path=usr/lib/help/auths/locale/C/SysMaintenance.html
1102 file path=usr/lib/help/auths/locale/C/SysPowerMgmtBrightness.html
1103 file path=usr/lib/help/auths/locale/C/SysPowerMgmtHeader.html
1104 file path=usr/lib/help/auths/locale/C/SysPowerMgmtSuspend.html
1105 file path=usr/lib/help/auths/locale/C/SysPowerMgmtSuspendtoDisk.html
1106 file path=usr/lib/help/auths/locale/C/SysPowerMgmtSuspendtoRAM.html
1107 file path=usr/lib/help/auths/locale/C/SysShutdown.html
1108 file path=usr/lib/help/auths/locale/C/SysSyseventRead.html
1109 file path=usr/lib/help/auths/locale/C/SysSyseventWrite.html
1110 file path=usr/lib/help/auths/locale/C/WifiConfig.html
1111 file path=usr/lib/help/auths/locale/C/WifiWep.html
1112 file path=usr/lib/help/auths/locale/C/ZoneCloneFrom.html
1113 file path=usr/lib/help/auths/locale/C/ZoneHeader.html
1114 file path=usr/lib/help/auths/locale/C/ZoneLogin.html

```

```

1115 file path=usr/lib/help/auths/locale/C/ZoneManage.html
1116 file path=usr/lib/help/profiles/locale/C/RtAcctadm.html
1117 file path=usr/lib/help/profiles/locale/C/RtAll.html
1118 file path=usr/lib/help/profiles/locale/C/RtAuditCfg.html
1119 file path=usr/lib/help/profiles/locale/C/RtAuditCtrl.html
1120 file path=usr/lib/help/profiles/locale/C/RtAuditReview.html
1121 file path=usr/lib/help/profiles/locale/C/RtCPUPowerManagement.html
1122 file path=usr/lib/help/profiles/locale/C/RtConsUser.html
1123 file path=usr/lib/help/profiles/locale/C/RtContractObserver.html
1124 file path=usr/lib/help/profiles/locale/C/RtCronMngmnt.html
1125 file path=usr/lib/help/profiles/locale/C/RtCryptoMngmnt.html
1126 file path=usr/lib/help/profiles/locale/C/RtDHCPMngmnt.html
1127 file path=usr/lib/help/profiles/locale/C/RtDatAdmin.html
1128 file path=usr/lib/help/profiles/locale/C/RtDefault.html
1129 file path=usr/lib/help/profiles/locale/C/RtDeviceMngmnt.html
1130 file path=usr/lib/help/profiles/locale/C/RtDeviceSecurity.html
1131 file path=usr/lib/help/profiles/locale/C/RtExAcctFlow.html
1132 file path=usr/lib/help/profiles/locale/C/RtExAcctNet.html
1133 file path=usr/lib/help/profiles/locale/C/RtExAcctProcess.html
1134 file path=usr/lib/help/profiles/locale/C/RtExAcctTask.html
1135 file path=usr/lib/help/profiles/locale/C/RtFTPMngmnt.html
1136 file path=usr/lib/help/profiles/locale/C/RtFileSysMngmnt.html
1137 file path=usr/lib/help/profiles/locale/C/RtFileSysSecurity.html
1138 file path=usr/lib/help/profiles/locale/C/RtHotplugMngmnt.html
1139 file path=usr/lib/help/profiles/locale/C/RtIPFilterMngmnt.html
1140 file path=usr/lib/help/profiles/locale/C/RtIdmapMngmnt.html
1141 file path=usr/lib/help/profiles/locale/C/RtIdmapNameRulesMngmnt.html
1142 file path=usr/lib/help/profiles/locale/C/RtInetdMngmnt.html
1143 file path=usr/lib/help/profiles/locale/C/RtKerberosClntMngmnt.html
1144 file path=usr/lib/help/profiles/locale/C/RtKerberosSrvrMngmnt.html
1145 file path=usr/lib/help/profiles/locale/C/RtLogMngmnt.html
1146 file path=usr/lib/help/profiles/locale/C/RtMailMngmnt.html
1147 file path=usr/lib/help/profiles/locale/C/RtMaintAndRepair.html
1148 file path=usr/lib/help/profiles/locale/C/RtMediaBkup.html
1149 file path=usr/lib/help/profiles/locale/C/RtMediaCtlg.html
1150 file path=usr/lib/help/profiles/locale/C/RtMediaRestore.html
1151 file path=usr/lib/help/profiles/locale/C/RtNDMPMngmnt.html
1152 file path=usr/lib/help/profiles/locale/C/RtNameServiceAdmin.html
1153 file path=usr/lib/help/profiles/locale/C/RtNameServiceSecure.html
1154 file path=usr/lib/help/profiles/locale/C/RtNetAutoconfAdmin.html
1155 file path=usr/lib/help/profiles/locale/C/RtNetAutoconfUser.html
1156 file path=usr/lib/help/profiles/locale/C/RtNetILB.html
1157 file path=usr/lib/help/profiles/locale/C/RtNetIPsec.html
1158 file path=usr/lib/help/profiles/locale/C/RtNetLinkSecure.html
1159 file path=usr/lib/help/profiles/locale/C/RtNetMngmnt.html
1160 file path=usr/lib/help/profiles/locale/C/RtNetObservability.html
1161 file path=usr/lib/help/profiles/locale/C/RtNetSecure.html
1162 file path=usr/lib/help/profiles/locale/C/RtNetVRRP.html
1163 file path=usr/lib/help/profiles/locale/C/RtNetWifiMngmnt.html
1164 file path=usr/lib/help/profiles/locale/C/RtNetWifiSecure.html
1165 file path=usr/lib/help/profiles/locale/C/RtObAccessMngmnt.html
1166 file path=usr/lib/help/profiles/locale/C/RtOperator.html
1167 file path=usr/lib/help/profiles/locale/C/RtPriAdmin.html
1168 file path=usr/lib/help/profiles/locale/C/RtPrntAdmin.html
1169 file path=usr/lib/help/profiles/locale/C/RtProcManagement.html
1170 file path=usr/lib/help/profiles/locale/C/RtReparseMngmnt.html
1171 file path=usr/lib/help/profiles/locale/C/RtReservedProfile.html
1172 file path=usr/lib/help/profiles/locale/C/RtRightsDelegate.html
1173 file path=usr/lib/help/profiles/locale/C/RtSMBFSMngmnt.html
1174 file path=usr/lib/help/profiles/locale/C/RtSMBMngmnt.html
1175 file path=usr/lib/help/profiles/locale/C/RtSoftwareInstall.html
1176 file path=usr/lib/help/profiles/locale/C/RtSysAdmin.html
1177 file path=usr/lib/help/profiles/locale/C/RtSysEvMngmnt.html
1178 file path=usr/lib/help/profiles/locale/C/RtSysPowerMgmt.html
1179 file path=usr/lib/help/profiles/locale/C/RtSysPowerMgmtBrightness.html
1180 file path=usr/lib/help/profiles/locale/C/RtSysPowerMgmtSuspend.html

```

```

1181 file path=usr/lib/help/profiles/locale/C/RtSysPowerMgmtSuspendtoDisk.html
1182 file path=usr/lib/help/profiles/locale/C/RtSysPowerMgmtSuspendtoRAM.html
1183 file path=usr/lib/help/profiles/locale/C/RtUserMngmnt.html
1184 file path=usr/lib/help/profiles/locale/C/RtUserSecurity.html
1185 file path=usr/lib/help/profiles/locale/C/RtVscanMngmnt.html
1186 file path=usr/lib/help/profiles/locale/C/RtZFSFileSysMngmnt.html
1187 file path=usr/lib/help/profiles/locale/C/RtZFSStorageMngmnt.html
1188 file path=usr/lib/help/profiles/locale/C/RtZoneMngmnt.html
1189 file path=usr/lib/help/profiles/locale/C/RtZoneSecurity.html
1190 file path=usr/lib/hotplugd mode=0555
1191 file path=usr/lib/iconv/646da.8859.t mode=0444
1192 file path=usr/lib/iconv/646de.8859.t mode=0444
1193 file path=usr/lib/iconv/646en.8859.t mode=0444
1194 file path=usr/lib/iconv/646es.8859.t mode=0444
1195 file path=usr/lib/iconv/646fr.8859.t mode=0444
1196 file path=usr/lib/iconv/646it.8859.t mode=0444
1197 file path=usr/lib/iconv/646sv.8859.t mode=0444
1198 file path=usr/lib/iconv/8859.646.t mode=0444
1199 file path=usr/lib/iconv/8859.646da.t mode=0444
1200 file path=usr/lib/iconv/8859.646de.t mode=0444
1201 file path=usr/lib/iconv/8859.646en.t mode=0444
1202 file path=usr/lib/iconv/8859.646es.t mode=0444
1203 file path=usr/lib/iconv/8859.646fr.t mode=0444
1204 file path=usr/lib/iconv/8859.646it.t mode=0444
1205 file path=usr/lib/iconv/8859.646sv.t mode=0444
1206 file path=usr/lib/iconv/iconv_data mode=0444
1207 file path=usr/lib/idmapd mode=0555
1208 file path=usr/lib/inet/$(ARCH32)/in.iked mode=0555
1209 file path=usr/lib/inet/$(ARCH64)/in.iked mode=0555
1210 file path=usr/lib/inet/certdb mode=0555
1211 file path=usr/lib/inet/certlocal mode=0555
1212 file path=usr/lib/inet/certrldb mode=0555
1213 file path=usr/lib/inet/inetd mode=0555
1214 file path=usr/lib/intrd mode=0555
1215 file path=usr/lib/isaexec mode=0555
1216 file path=usr/lib/kssladm mode=0555
1217 $(sparc_ONLY)file path=usr/lib/ld.so
1218 file path=usr/lib/libshare.so.1
1219 file path=usr/lib/makekey mode=0555
1220 file path=usr/lib/more.help
1221 file path=usr/lib/newsyslog group=sys mode=0555
1222 file path=usr/lib/passmgmt group=sys mode=0555
1223 file path=usr/lib/pci/pcidr mode=0555
1224 file path=usr/lib/pci/pcidr_plugin.so
1225 file path=usr/lib/pfexecd mode=0555
1226 file path=usr/lib/platexec mode=0555
1227 file path=usr/lib/rcm/modules/SUNW_aggr_rcm.so mode=0555
1228 file path=usr/lib/rcm/modules/SUNW_cluster_rcm.so mode=0555
1229 file path=usr/lib/rcm/modules/SUNW_dump_rcm.so mode=0555
1230 file path=usr/lib/rcm/modules/SUNW_filesys_rcm.so mode=0555
1231 file path=usr/lib/rcm/modules/SUNW_ibpart_rcm.so mode=0555
1232 file path=usr/lib/rcm/modules/SUNW_ip_anon_rcm.so mode=0555
1233 file path=usr/lib/rcm/modules/SUNW_ip_rcm.so mode=0555
1234 file path=usr/lib/rcm/modules/SUNW_mpxio_rcm.so mode=0555
1235 file path=usr/lib/rcm/modules/SUNW_network_rcm.so mode=0555
1236 file path=usr/lib/rcm/modules/SUNW_swap_rcm.so mode=0555
1237 $(sparc_ONLY)file path=usr/lib/rcm/modules/SUNW_ttymux_rcm.so mode=0555
1238 file path=usr/lib/rcm/modules/SUNW_vlan_rcm.so mode=0555
1239 file path=usr/lib/rcm/modules/SUNW_vnic_rcm.so mode=0555
1240 file path=usr/lib/rcm/rcm_daemon mode=0555
1241 file path=usr/lib/reparse/reparsed group=sys mode=0555
1242 file path=usr/lib/saf/listen group=sys mode=0755
1243 file path=usr/lib/saf/nlps_server group=sys mode=0755
1244 file path=usr/lib/saf/sac group=sys mode=0555
1245 file path=usr/lib/saf/ttymon group=sys mode=0555
1246 file path=usr/lib/sysevent/modules/datalink_mod.so

```

```

1247 file path=usr/lib/sysevent/modules/devfsadm_mod.so
1248 file path=usr/lib/sysevent/modules/sysevent_conf_mod.so
1249 file path=usr/lib/sysevent/modules/sysevent_reg_mod.so
1250 file path=usr/lib/sysevent/syseventconfd mode=0555
1251 file path=usr/lib/sysevent/syseventd mode=0555
1252 file path=usr/lib/utmp_update mode=4555
1253 file path=usr/lib/utmpd mode=0555
1254 file path=usr/lib/vtdaemon mode=0555
1255 file path=usr/lib/vtinfo mode=0555
1256 file path=usr/lib/vtxlock mode=0555
1257 file path=usr/sadm/bin/puttext mode=0555
1258 file path=usr/sadm/install/miniroot.db group=sys mode=0444
1259 file path=usr/sadm/install/scripts/i.ipsecalgs group=sys mode=0555
1260 file path=usr/sadm/install/scripts/i.kcfconf group=sys mode=0555
1261 file path=usr/sadm/install/scripts/i.kmfconf group=sys mode=0555
1262 file path=usr/sadm/install/scripts/i.manifest group=sys mode=0555
1263 file path=usr/sadm/install/scripts/i.pkcs11conf group=sys mode=0555
1264 file path=usr/sadm/install/scripts/i.rbac group=sys mode=0555
1265 file path=usr/sadm/install/scripts/r.ipsecalgs group=sys mode=0555
1266 file path=usr/sadm/install/scripts/r.kcfconf group=sys mode=0555
1267 file path=usr/sadm/install/scripts/r.kmfconf group=sys mode=0555
1268 file path=usr/sadm/install/scripts/r.manifest group=sys mode=0555
1269 file path=usr/sadm/install/scripts/r.pkcs11conf group=sys mode=0555
1270 file path=usr/sadm/install/scripts/r.rbac group=sys mode=0555
1271 file path=usr/sadm/updates mode=0444
1272 $(i386_ONLY)file path=usr/sbin/$(ARCH32)/add_drv group=sys mode=0555
1273 $(i386_ONLY)file path=usr/sbin/$(ARCH32)/modinfo group=sys mode=0555
1274 $(i386_ONLY)file path=usr/sbin/$(ARCH32)/modload group=sys mode=0555
1275 $(i386_ONLY)file path=usr/sbin/$(ARCH32)/modunload group=sys mode=0555
1276 $(i386_ONLY)file path=usr/sbin/$(ARCH32)/pbind group=sys mode=0555
1277 $(i386_ONLY)file path=usr/sbin/$(ARCH32)/prtconf group=sys mode=2555
1278 $(i386_ONLY)file path=usr/sbin/$(ARCH32)/psrset group=sys mode=0555
1279 $(i386_ONLY)file path=usr/sbin/$(ARCH32)/rem_drv group=sys mode=0555
1280 $(i386_ONLY)file path=usr/sbin/$(ARCH32)/swap group=sys mode=2555
1281 $(i386_ONLY)file path=usr/sbin/$(ARCH32)/sysdef group=sys mode=2555
1282 $(i386_ONLY)file path=usr/sbin/$(ARCH32)/update_drv group=sys mode=0555
1283 $(i386_ONLY)file path=usr/sbin/$(ARCH32)/whodo mode=4555
1284 file path=usr/sbin/$(ARCH64)/add_drv group=sys mode=0555
1285 file path=usr/sbin/$(ARCH64)/modinfo group=sys mode=0555
1286 file path=usr/sbin/$(ARCH64)/modload group=sys mode=0555
1287 file path=usr/sbin/$(ARCH64)/modunload group=sys mode=0555
1288 file path=usr/sbin/$(ARCH64)/pbind group=sys mode=0555
1289 file path=usr/sbin/$(ARCH64)/prtconf group=sys mode=2555
1290 file path=usr/sbin/$(ARCH64)/psrset group=sys mode=0555
1291 file path=usr/sbin/$(ARCH64)/rem_drv group=sys mode=0555
1292 file path=usr/sbin/$(ARCH64)/swap group=sys mode=2555
1293 file path=usr/sbin/$(ARCH64)/sysdef group=sys mode=2555
1294 file path=usr/sbin/$(ARCH64)/update_drv group=sys mode=0555
1295 file path=usr/sbin/$(ARCH64)/whodo mode=4555
1296 file path=usr/sbin/6to4relay mode=0555
1297 file path=usr/sbin/acctadm mode=0555
1298 file path=usr/sbin/allocate mode=4555
1299 file path=usr/sbin/arp mode=0555
1300 file path=usr/sbin/audit mode=0555
1301 file path=usr/sbin/auditconfig mode=0555
1302 file path=usr/sbin/auditd mode=0555
1303 file path=usr/sbin/auditrecord mode=0555
1304 file path=usr/sbin/auditreduce mode=0555
1305 file path=usr/sbin/auditstat mode=0555
1306 file path=usr/sbin/cfgadm mode=0555
1307 file path=usr/sbin/chroot mode=0555
1308 file path=usr/sbin/clear_locks mode=0555
1309 file path=usr/sbin/clinfo mode=0555
1310 file path=usr/sbin/clri mode=0555
1311 file path=usr/sbin/consadm group=sys mode=0555
1312 file path=usr/sbin/cron group=sys mode=0555

```



```

1313 file path=usr/sbin/devfsadm group=sys mode=0755
1314 file path=usr/sbin/devinfo mode=0555
1315 file path=usr/sbin/df mode=0555
1316 file path=usr/sbin/dfmounts mode=0555
1317 file path=usr/sbin/dispadm mode=0555
1318 file path=usr/sbin/dminfo mode=0555
1319 file path=usr/sbin/dumpadm mode=0555
1320 file path=usr/sbin/EEPROM group=sys mode=2555
1321 file path=usr/sbin/ff mode=0555
1322 file path=usr/sbin/fmthard group=sys mode=0555
1323 file path=usr/sbin/format mode=0555
1324 file path=usr/sbin/fsck mode=0555
1325 file path=usr/sbin/fstyp group=sys mode=0555
1326 file path=usr/sbin/fuser mode=0555
1327 file path=usr/sbin/getdevpolicy group=sys mode=0555
1328 file path=usr/sbin/getmajor group=sys mode=0755
1329 file path=usr/sbin/groupadd group=sys mode=0555
1330 file path=usr/sbin/groupdel group=sys mode=0555
1331 file path=usr/sbin/groupmod group=sys mode=0555
1332 file path=usr/sbin/grpck mode=0555
1333 file path=usr/sbin/halt mode=0755
1334 file path=usr/sbin/hotplug mode=0555
1335 file path=usr/sbin/imap mode=0555
1336 file path=usr/sbin/if_mpadm mode=0555
1337 file path=usr/sbin/ikeadm mode=0555
1338 file path=usr/sbin/ikecert mode=0555
1339 file path=usr/sbin/inetadm mode=0555
1340 file path=usr/sbin/inetconv mode=0555
1341 file path=usr/sbin/install mode=0555
1342 file path=usr/sbin/installboot group=sys mode=0555
1343 file path=usr/sbin/ipaddrsel mode=0555
1344 file path=usr/sbin/ipsecalg mode=0555
1345 file path=usr/sbin/ipsecconf mode=0555
1346 file path=usr/sbin/ipseckey mode=0555
1347 file path=usr/sbin/keyserv group=sys mode=0555
1348 file path=usr/sbin/killall mode=0555
1349 file path=usr/sbin/ksslcfg mode=0555
1350 file path=usr/sbin/link mode=0555
1351 file path=usr/sbin/locator mode=0555
1352 file path=usr/sbin/lofiadm mode=0555
1353 file path=usr/sbin/logadm mode=0555
1354 file path=usr/sbin/makedbm mode=0555
1355 file path=usr/sbin/mkdevalloc mode=0555
1356 file path=usr/sbin/mkfile mode=0555
1357 file path=usr/sbin/mknod mode=0555
1358 file path=usr/sbin/mountall group=sys mode=0555
1359 file path=usr/sbin/msgid mode=0555
1360 file path=usr/sbin/mvdir mode=0555
1361 file path=usr/sbin/ndd mode=0555
1362 file path=usr/sbin/ndp mode=0555
1363 file path=usr/sbin/nlsadmin group=adm mode=0755
1364 file path=usr/sbin/nltest mode=0555
1365 file path=usr/sbin/nscd mode=0555
1366 file path=usr/sbin/nwamadm mode=0555
1367 file path=usr/sbin/nwamcfg mode=0555
1368 file path=usr/sbin/ping mode=4555
1369 file path=usr/sbin/pmadm group=sys mode=0555
1370 file path=usr/sbin/praudit mode=0555
1371 $(i386_ONLY)file path=usr/sbin/prtdiag group=sys mode=2755
1372 file path=usr/sbin/prtvtoc group=sys mode=0555
1373 file path=usr/sbin/psradm group=sys mode=0555
1374 file path=usr/sbin/psrinfo group=sys mode=0555
1375 file path=usr/sbin/pwck mode=0555
1376 file path=usr/sbin/pwconv group=sys mode=0555
1377 file path=usr/sbin/raidctl mode=0555
1378 file path=usr/sbin/ramdiskadm mode=0555

```

```

1379 file path=usr/sbin/rctldm mode=0555
1380 file path=usr/sbin/root_archive group=sys mode=0555
1381 file path=usr/sbin/rpcbind mode=0555
1382 $(i386_ONLY)file path=usr/sbin/rtc mode=0555
1383 file path=usr/sbin/sacadm group=sys mode=4755
1384 file path=usr/sbin/setmnt mode=0555
1385 file path=usr/sbin/shareall mode=0555
1386 file path=usr/sbin/sharectl mode=0555
1387 file path=usr/sbin/sharemgr mode=0555
1388 file path=usr/sbin/shutdown group=sys mode=0755
1389 file path=usr/sbin/smbios mode=0555
1390 file path=usr/sbin/stmsboot mode=0555
1391 file path=usr/sbin/strace group=sys mode=0555
1392 file path=usr/sbin/strclean group=sys mode=0555
1393 file path=usr/sbin/strerr group=sys mode=0555
1394 file path=usr/sbin/sttydefs group=sys mode=0755
1395 file path=usr/sbin/svcadm mode=0555
1396 file path=usr/sbin/svccfg mode=0555
1397 file path=usr/sbin/syncinit mode=0555
1398 file path=usr/sbin/syncloop mode=0555
1399 file path=usr/sbin/syncstat mode=0555
1400 file path=usr/sbin/syseventadm group=sys mode=0555
1401 file path=usr/sbin/syslogd group=sys mode=0555
1402 file path=usr/sbin/tar mode=0555
1403 file path=usr/sbin/traceroute mode=4555
1404 file path=usr/sbin/trapstat mode=0555
1405 file path=usr/sbin/ttyadm group=sys mode=0755
1406 $(i386_ONLY)file path=usr/sbin/ucodeadm mode=0555
1407 file path=usr/sbin/umountall group=sys mode=0555
1408 file path=usr/sbin/unlink mode=0555
1409 file path=usr/sbin/unshareall mode=0555
1410 file path=usr/sbin/useradd group=sys mode=0555
1411 file path=usr/sbin/userdel group=sys mode=0555
1412 file path=usr/sbin/usermod group=sys mode=0555
1413 $(sparc_ONLY)file path=usr/sbin/virtinfo mode=0555
1414 file path=usr/sbin/volcopy mode=0555
1415 file path=usr/sbin/wall group=tty mode=2555
1416 file path=usr/sbin/zdump mode=0555
1417 file path=usr/sbin/zic mode=0555
1418 file path=usr/share/doc/ksh/COMPATIBILITY
1419 file path=usr/share/doc/ksh/DESIGN
1420 file path=usr/share/doc/ksh/OBSOLETE
1421 file path=usr/share/doc/ksh/README
1422 file path=usr/share/doc/ksh/RELEASE
1423 file path=usr/share/doc/ksh/TYPES
1424 file path=usr/share/doc/ksh/images/callouts/1.png
1425 file path=usr/share/doc/ksh/images/callouts/10.png
1426 file path=usr/share/doc/ksh/images/callouts/2.png
1427 file path=usr/share/doc/ksh/images/callouts/3.png
1428 file path=usr/share/doc/ksh/images/callouts/4.png
1429 file path=usr/share/doc/ksh/images/callouts/5.png
1430 file path=usr/share/doc/ksh/images/callouts/6.png
1431 file path=usr/share/doc/ksh/images/callouts/7.png
1432 file path=usr/share/doc/ksh/images/callouts/8.png
1433 file path=usr/share/doc/ksh/images/callouts/9.png
1434 file path=usr/share/doc/ksh/images/tag_bourne.png
1435 file path=usr/share/doc/ksh/images/tag_i18n.png
1436 file path=usr/share/doc/ksh/images/tag_ksh.png
1437 file path=usr/share/doc/ksh/images/tag_ksh88.png
1438 file path=usr/share/doc/ksh/images/tag_ksh93.png
1439 file path=usr/share/doc/ksh/images/tag_l10n.png
1440 file path=usr/share/doc/ksh/images/tag_perf.png
1441 file path=usr/share/doc/ksh/shell_styleguide.docbook
1442 file path=usr/share/lib/mailx/mailx.help
1443 file path=usr/share/lib/mailx/mailx.help.~
1444 file path=usr/share/lib/tabset/3101

```

```

1445 file path=usr/share/lib/tabset/bee hive
1446 file path=usr/share/lib/tabset/hds
1447 file path=usr/share/lib/tabset/hds3
1448 file path=usr/share/lib/tabset/std
1449 file path=usr/share/lib/tabset/stdcrt
1450 file path=usr/share/lib/tabset/teleray
1451 file path=usr/share/lib/tabset/vt100
1452 file path=usr/share/lib/tabset/wyse-adds
1453 file path=usr/share/lib/tabset/xerox1720
1454 file path=usr/share/lib/termcap
1455 file path=usr/share/lib/unittab
1456 file path=usr/share/lib/xml/dtd/adt_record.dtd.1
1457 file path=usr/share/lib/xml/dtd/kmfpolicy.dtd
1458 file path=usr/share/lib/xml/dtd/service_bundle.dtd.1 group=sys
1459 file path=usr/share/lib/xml/style/adt_record.xsl.1
1460 file path=var/adm/aculog mode=0600 owner=ucpc preserve=true
1461 file path=var/adm/spellhist mode=0666 preserve=true
1462 file path=var/adm/utmpx preserve=true
1463 file path=var/adm/wtmpx group=adm owner=adm preserve=true
1464 file path=var/log/authlog group=sys mode=0600 preserve=true
1465 file path=var/log/syslog group=sys preserve=true
1466 file path=var/saf/zsmon/log group=sys preserve=true
1467 file path=var/spool/cron/crontabs/adm group=sys mode=0600 preserve=true
1468 file path=var/spool/cron/crontabs/root group=sys mode=0600 preserve=true
1469 hardlink path=etc/rc2.d/S20syssetup target=../etc/init.d/syssetup
1470 hardlink path=etc/rc2.d/S89PRESERVE target=../etc/init.d/PRESERVE
1471 $(sparc_ONLY)hardlink path=etc/svc/profile/platform_SUNW,Sun-Fire-V890.xml \
1472     target=../platform_SUNW,Sun-Fire-880.xml
1473 $(sparc_ONLY)hardlink \
1474     path=etc/svc/profile/platform_SUNW,UltraSPARC-IIe-NetraCT-40.xml \
1475     target=../platform_SUNW,UltraSPARC-IIi-NetraCT.xml
1476 $(sparc_ONLY)hardlink \
1477     path=etc/svc/profile/platform_SUNW,UltraSPARC-IIe-NetraCT-60.xml \
1478     target=../platform_SUNW,UltraSPARC-IIi-NetraCT.xml
1479 hardlink path=sbin/rc5 target=../sbin/rc0
1480 hardlink path=sbin/rc6 target=../sbin/rc0
1481 hardlink path=usr/bin/$(ARCH32)/encrypt target=decrypt
1482 hardlink path=usr/bin/$(ARCH32)/ksh target=ksh93
1483 hardlink path=usr/bin/$(ARCH32)/mac target=digest
1484 hardlink path=usr/bin/$(ARCH32)/rksh target=ksh93
1485 hardlink path=usr/bin/$(ARCH32)/rksh93 target=ksh93
1486 $(i386_ONLY)hardlink path=usr/bin/$(ARCH32)/w target=uptime
1487 hardlink path=usr/bin/$(ARCH64)/encrypt target=decrypt
1488 hardlink path=usr/bin/$(ARCH64)/ksh target=ksh93
1489 hardlink path=usr/bin/$(ARCH64)/mac target=digest
1490 hardlink path=usr/bin/$(ARCH64)/rksh target=ksh93
1491 hardlink path=usr/bin/$(ARCH64)/rksh93 target=ksh93
1492 hardlink path=usr/bin/$(ARCH64)/w target=uptime
1493 hardlink path=usr/bin/bg target=../usr/bin/alias
1494 hardlink path=usr/bin/cd target=../usr/bin/alias
1495 hardlink path=usr/bin/cksum target=../usr/bin/alias
1496 hardlink path=usr/bin/cmp target=../usr/bin/alias
1497 hardlink path=usr/bin/comm target=../usr/bin/alias
1498 hardlink path=usr/bin/command target=../usr/bin/alias
1499 hardlink path=usr/bin/cut target=../usr/bin/alias
1500 hardlink path=usr/bin/decrypt target=../usr/lib/isaexec
1501 hardlink path=usr/bin/digest target=../usr/lib/isaexec
1502 hardlink path=usr/bin/dispgid target=../usr/bin/ckgid
1503 hardlink path=usr/bin/dispuid target=../usr/bin/ckuid
1504 hardlink path=usr/bin/edit target=../has/bin/edit
1505 hardlink path=usr/bin/encrypt target=../usr/lib/isaexec
1506 hardlink path=usr/bin/fc target=../usr/bin/alias
1507 hardlink path=usr/bin/fg target=../usr/bin/alias
1508 hardlink path=usr/bin/getopts target=../usr/bin/alias
1509 hardlink path=usr/bin/hash target=../usr/bin/alias
1510 hardlink path=usr/bin/jobs target=../usr/bin/alias

```

```

1511 hardlink path=usr/bin/join target=../usr/bin/alias
1512 hardlink path=usr/bin/kill target=../usr/bin/alias
1513 hardlink path=usr/bin/ksh target=../usr/lib/isaexec
1514 hardlink path=usr/bin/ksh93 target=../usr/lib/isaexec
1515 hardlink path=usr/bin/ln target=../usr/bin/cp
1516 hardlink path=usr/bin/logname target=../usr/bin/alias
1517 hardlink path=usr/bin/mac target=../usr/lib/isaexec
1518 hardlink path=usr/bin/mv target=../usr/bin/cp
1519 hardlink path=usr/bin/newtask target=../usr/lib/isaexec
1520 hardlink path=usr/bin/nohup target=../usr/lib/isaexec
1521 hardlink path=usr/bin/page target=../usr/bin/more
1522 hardlink path=usr/bin/paste target=../usr/bin/alias
1523 hardlink path=usr/bin/pfbash target=../usr/bin/pfexec
1524 hardlink path=usr/bin/pfcsh target=../usr/bin/pfexec
1525 hardlink path=usr/bin/pfksh target=../usr/bin/pfexec
1526 hardlink path=usr/bin/pfksh93 target=../usr/bin/pfexec
1527 hardlink path=usr/bin/pfkrsh target=../usr/bin/pfexec
1528 hardlink path=usr/bin/pfkrsh93 target=../usr/bin/pfexec
1529 hardlink path=usr/bin/pfsh target=../usr/bin/pfexec
1530 hardlink path=usr/bin/pftcsh target=../usr/bin/pfexec
1531 hardlink path=usr/bin/pfzsh target=../usr/bin/pfexec
1532 hardlink path=usr/bin/pkill target=../usr/bin/pgrep
1533 hardlink path=usr/bin/prctl target=../usr/lib/isaexec
1534 hardlink path=usr/bin/print target=../usr/bin/alias
1535 hardlink path=usr/bin/prstat target=../usr/lib/isaexec
1536 hardlink path=usr/bin/ps target=../usr/lib/isaexec
1537 hardlink path=usr/bin/read target=../usr/bin/alias
1538 hardlink path=usr/bin/red target=../usr/bin/ed
1539 hardlink path=usr/bin/rev target=../usr/bin/alias
1540 hardlink path=usr/bin/rksh target=../usr/lib/isaexec
1541 hardlink path=usr/bin/rksh93 target=../usr/lib/isaexec
1542 hardlink path=usr/bin/savecore target=../usr/lib/isaexec
1543 hardlink path=usr/bin/setuname target=../usr/lib/isaexec
1544 hardlink path=usr/bin/sleep target=../usr/bin/alias
1545 hardlink path=usr/bin/sum target=../usr/bin/alias
1546 hardlink path=usr/bin/tee target=../usr/bin/alias
1547 hardlink path=usr/bin/test target=../usr/bin/alias
1548 hardlink path=usr/bin/touch target=../usr/bin/settime
1549 hardlink path=usr/bin/type target=../usr/bin/alias
1550 hardlink path=usr/bin/ulimit target=../usr/bin/alias
1551 hardlink path=usr/bin/umask target=../usr/bin/alias
1552 hardlink path=usr/bin/unalias target=../usr/bin/alias
1553 hardlink path=usr/bin/uniq target=../usr/bin/alias
1554 hardlink path=usr/bin/uptime target=../usr/lib/isaexec
1555 hardlink path=usr/bin/vedit target=../has/bin/edit
1556 hardlink path=usr/bin/w target=../usr/lib/isaexec
1557 hardlink path=usr/bin/wait target=../usr/bin/alias
1558 hardlink path=usr/bin/wc target=../usr/bin/alias
1559 hardlink path=usr/bin/ex target=edit
1560 hardlink path=usr/bin/pfsh target=../bin/pfexec
1561 hardlink path=usr/bin/vedit target=edit
1562 hardlink path=usr/bin/vi target=edit
1563 hardlink path=usr/bin/view target=edit
1564 hardlink path=usr/lib/fs/ufs/fsfstyp target=../sbin/fstyp
1565 hardlink path=usr/lib/fs/ufs/dcopy target=../usr/lib/fs/ufs/clri
1566 hardlink path=usr/lib/fs/ufs/fstyp target=../sbin/fstyp
1567 hardlink path=usr/lib/fs/ufs/quotaoon \
1568     target=../usr/lib/fs/ufs/quotaooff
1569 hardlink path=usr/lib/inet/in.iked target=../usr/lib/isaexec
1570 hardlink path=usr/sbin/bin/dispgid target=../usr/bin/ckgid
1571 hardlink path=usr/sbin/bin/dispuid target=../usr/bin/ckuid
1572 hardlink path=usr/sbin/bin/errange target=../usr/bin/ckrange
1573 hardlink path=usr/sbin/bin/errdate target=../usr/bin/ckdate
1574 hardlink path=usr/sbin/bin/errgid target=../usr/bin/ckgid
1575 hardlink path=usr/sbin/bin/errint target=../usr/bin/ckint
1576 hardlink path=usr/sbin/bin/erritem target=../usr/bin/ckitem

```

```

1577 hardlink path=usr/sadm/bin/errpath target=../../usr/bin/ckpath
1578 hardlink path=usr/sadm/bin/errstr target=../../usr/bin/ckstr
1579 hardlink path=usr/sadm/bin/errtime target=../../usr/bin/cktime
1580 hardlink path=usr/sadm/bin/erruid target=../../usr/bin/ckuid
1581 hardlink path=usr/sadm/bin/erryor target=../../usr/bin/ckyor
1582 hardlink path=usr/sadm/bin/helpdate target=../../usr/bin/ckdate
1583 hardlink path=usr/sadm/bin/helpgid target=../../usr/bin/ckgid
1584 hardlink path=usr/sadm/bin/helpint target=../../usr/bin/ckint
1585 hardlink path=usr/sadm/bin/helpitem target=../../usr/bin/ckitem
1586 hardlink path=usr/sadm/bin/helppath target=../../usr/bin/ckpath
1587 hardlink path=usr/sadm/bin/helpprange target=../../usr/bin/ckrange
1588 hardlink path=usr/sadm/bin/helpstr target=../../usr/bin/ckstr
1589 hardlink path=usr/sadm/bin/helptime target=../../usr/bin/cktime
1590 hardlink path=usr/sadm/bin/helpuid target=../../usr/bin/ckuid
1591 hardlink path=usr/sadm/bin/helpyor target=../../usr/bin/ckyor
1592 hardlink path=usr/sadm/bin/valdate target=../../usr/bin/ckdate
1593 hardlink path=usr/sadm/bin/valgid target=../../usr/bin/ckgid
1594 hardlink path=usr/sadm/bin/valint target=../../usr/bin/ckint
1595 hardlink path=usr/sadm/bin/valpath target=../../usr/bin/ckpath
1596 hardlink path=usr/sadm/bin/valrange target=../../usr/bin/ckrange
1597 hardlink path=usr/sadm/bin/valstr target=../../usr/bin/ckstr
1598 hardlink path=usr/sadm/bin/valtime target=../../usr/bin/cktime
1599 hardlink path=usr/sadm/bin/valid target=../../usr/bin/ckuid
1600 hardlink path=usr/sadm/bin/valyor target=../../usr/bin/ckyor
1601 hardlink path=usr/sbin/add_drv target=../../usr/lib/isaexec
1602 hardlink path=usr/sbin/audlinks target=../devfsadm
1603 hardlink path=usr/sbin/consadm target=../../usr/sbin/consadm
1604 hardlink path=usr/sbin/deallocate target=../../usr/sbin/allocate
1605 hardlink path=usr/sbin/devlinks target=../devfsadm
1606 hardlink path=usr/sbin/dfshares target=../../usr/sbin/dfmounts
1607 hardlink path=usr/sbin/disks target=../devfsadm
1608 hardlink path=usr/sbin/drvconfig target=../devfsadm
1609 hardlink path=usr/sbin/list_devices target=../../usr/sbin/allocate
1610 hardlink path=usr/sbin/mkdevmaps target=../../usr/sbin/mkdevalloc
1611 hardlink path=usr/sbin/modinfo target=../../usr/lib/isaexec
1612 hardlink path=usr/sbin/modload target=../../usr/lib/isaexec
1613 hardlink path=usr/sbin/modunload target=../../usr/lib/isaexec
1614 hardlink path=usr/sbin/pbind target=../../usr/lib/isaexec
1615 hardlink path=usr/sbin/ports target=../devfsadm
1616 hardlink path=usr/sbin/poweroff target=../halt
1617 hardlink path=usr/sbin/prtconf target=../../usr/lib/isaexec
1618 $(sparc_ONLY)hardlink path=usr/sbin/prtdiag target=../../usr/lib/platexec
1619 hardlink path=usr/sbin/psrset target=../../usr/lib/isaexec
1620 hardlink path=usr/sbin/reboot target=../halt
1621 hardlink path=usr/sbin/rem_drv target=../../usr/lib/isaexec
1622 hardlink path=usr/sbin/roleadd target=../../usr/sbin/useradd
1623 hardlink path=usr/sbin/roledel target=../../usr/sbin/userdel
1624 hardlink path=usr/sbin/rolemod target=../../usr/sbin/usermod
1625 hardlink path=usr/sbin/share target=../../usr/sbin/sharemgr
1626 hardlink path=usr/sbin/swap target=../../usr/lib/isaexec
1627 hardlink path=usr/sbin/sysdef target=../../usr/lib/isaexec
1628 hardlink path=usr/sbin/tapes target=../devfsadm
1629 hardlink path=usr/sbin/unshare target=../../usr/sbin/sharemgr
1630 hardlink path=usr/sbin/update_drv target=../../usr/lib/isaexec
1631 hardlink path=usr/sbin/whodo target=../../usr/lib/isaexec
1632 legacy pkg=SUNWcsr \
1633   desc="core software for a specific instruction-set architecture" \
1634   name="Core Solaris, (Root)"
1635 legacy pkg=SUNWcsu \
1636   desc="core software for a specific instruction-set architecture" \
1637   name="Core Solaris, (USR)"
1638 legacy pkg=SUNWftpr desc="FTP Server Configuration Files" \
1639   name="FTP Server, (Root)"
1640 license cr_ATT license=cr_ATT
1641 license cr_Sun license=cr_Sun
1642 license lic_CDDL license=lic_CDDL

```

```

1643 license usr/src/cmd/cmd-inet/sbin/ifparse/THIRDPARTYLICENSE \
1644   license=usr/src/cmd/cmd-inet/sbin/ifparse/THIRDPARTYLICENSE
1645 license usr/src/cmd/cmd-inet/usr.lib/in.mpathd/THIRDPARTYLICENSE \
1646   license=usr/src/cmd/cmd-inet/usr.lib/in.mpathd/THIRDPARTYLICENSE
1647 license usr/src/cmd/cmd-inet/usr/sbin/THIRDPARTYLICENSE.arp \
1648   license=usr/src/cmd/cmd-inet/usr/sbin/THIRDPARTYLICENSE.arp
1649 license usr/src/cmd/cmd-inet/usr/sbin/THIRDPARTYLICENSE.route \
1650   license=usr/src/cmd/cmd-inet/usr/sbin/THIRDPARTYLICENSE.route
1651 license usr/src/cmd/cmd-inet/usr/sbin/ifconfig/THIRDPARTYLICENSE \
1652   license=usr/src/cmd/cmd-inet/usr/sbin/ifconfig/THIRDPARTYLICENSE
1653 license usr/src/cmd/cmd-inet/usr/sbin/traceroute/THIRDPARTYLICENSE \
1654   license=usr/src/cmd/cmd-inet/usr/sbin/traceroute/THIRDPARTYLICENSE
1655 license usr/src/cmd/cron/THIRDPARTYLICENSE \
1656   license=usr/src/cmd/cron/THIRDPARTYLICENSE
1657 license usr/src/cmd/csh/THIRDPARTYLICENSE \
1658   license=usr/src/cmd/csh/THIRDPARTYLICENSE
1659 license usr/src/cmd/eeprom/THIRDPARTYLICENSE \
1660   license=usr/src/cmd/eeprom/THIRDPARTYLICENSE
1661 license usr/src/cmd/fs.d/ufs/THIRDPARTYLICENSE \
1662   license=usr/src/cmd/fs.d/ufs/THIRDPARTYLICENSE
1663 license usr/src/cmd/mt/THIRDPARTYLICENSE \
1664   license=usr/src/cmd/mt/THIRDPARTYLICENSE
1665 license usr/src/cmd/script/THIRDPARTYLICENSE \
1666   license=usr/src/cmd/script/THIRDPARTYLICENSE
1667 license usr/src/cmd/sed/THIRDPARTYLICENSE \
1668   license=usr/src/cmd/sed/THIRDPARTYLICENSE
1669 license usr/src/cmd/stat/vmstat/THIRDPARTYLICENSE \
1670   license=usr/src/cmd/stat/vmstat/THIRDPARTYLICENSE
1671 license usr/src/cmd/tail/THIRDPARTYLICENSE \
1672   license=usr/src/cmd/tail/THIRDPARTYLICENSE
1673 license usr/src/cmd/tip/THIRDPARTYLICENSE \
1674   license=usr/src/cmd/tip/THIRDPARTYLICENSE
1675 license usr/src/cmd/tr/THIRDPARTYLICENSE \
1676   license=usr/src/cmd/tr/THIRDPARTYLICENSE
1677 license usr/src/cmd/vi/THIRDPARTYLICENSE \
1678   license=usr/src/cmd/vi/THIRDPARTYLICENSE
1679 license usr/src/cmd/which/THIRDPARTYLICENSE \
1680   license=usr/src/cmd/which/THIRDPARTYLICENSE
1681 license usr/src/cmd/xstr/THIRDPARTYLICENSE \
1682   license=usr/src/cmd/xstr/THIRDPARTYLICENSE
1683 license usr/src/common/bzip2/LICENSE license=usr/src/common/bzip2/LICENSE
1684 link path=bin target=../usr/bin
1685 link path=etc/TIMEZONE target=../default/init
1686 link path=etc/autopush target=../sbin/autopush
1687 link path=etc/cfgadm target=../../usr/sbin/cfgadm
1688 link path=etc/clri target=../../usr/sbin/clri
1689 link path=etc/cron target=../../usr/sbin/cron
1690 link path=etc/dcopy target=../../usr/sbin/dcopy
1691 link path=etc/ff target=../../usr/sbin/ff
1692 link path=etc/fmthard target=../../usr/sbin/fmthard
1693 link path=etc/format target=../../usr/sbin/format
1694 link path=etc/fsck target=../../usr/sbin/fsck
1695 link path=etc/fsdb target=../../usr/sbin/fsdb
1696 link path=etc/fstyp target=../../usr/sbin/fstyp
1697 link path=etc/ftusers target=../../ftpd/ftusers
1698 link path=etc/getty target=../../usr/lib/saf/ttymon
1699 link path=etc/grpck target=../../usr/sbin/grpck
1700 link path=etc/halt target=../../usr/sbin/halt
1701 link path=etc/hosts target=../inet/hosts
1702 link path=etc/inet/ipnodes target=../hosts
1703 link path=etc/inetd.conf target=../inet/inetd.conf
1704 link path=etc/init target=../../sbin/init
1705 link path=etc/install target=../../usr/sbin/install
1706 link path=etc/killall target=../../usr/sbin/killall
1707 link path=etc/labelit target=../../usr/sbin/labelit
1708 link path=etc/lib/ld.so.1 target=../../lib/ld.so.1

```

```

1709 link path=etc/lib/libdl.so.1 target=../lib/libdl.so.1
1710 link path=etc/lib/nss_files.so.1 target=../lib/nss_files.so.1
1711 link path=etc/log target=../var/adm/log
1712 link path=etc/mkfs target=../usr/sbin/mkfs
1713 link path=etc/mknod target=../usr/sbin/mknod
1714 link path=etc/mount target=../sbin/mount
1715 link path=etc/mountall target=../sbin/mountall
1716 link path=etc/ncheck target=../usr/sbin/ncheck
1717 link path=etc/netmasks target=../inet/netmasks
1718 link path=etc/networks target=../inet/networks
1719 link path=etc/protocols target=../inet/protocols
1720 link path=etc/prtconf target=../usr/sbin/prtconf
1721 link path=etc/prtvtoc target=../usr/sbin/prtvtoc
1722 link path=etc/rc0 target=../sbin/rc0
1723 link path=etc/rc1 target=../sbin/rc1
1724 link path=etc/rc2 target=../sbin/rc2
1725 link path=etc/rc3 target=../sbin/rc3
1726 link path=etc/rc5 target=../sbin/rc5
1727 link path=etc/rc6 target=../sbin/rc6
1728 link path=etc/rcS target=../sbin/rcS
1729 link path=etc/reboot target=../usr/sbin/halt
1730 link path=etc/security/audit/localhost/files target=../var/audit
1731 link path=etc/services target=../inet/services
1732 link path=etc/setmnt target=../usr/sbin/setmnt
1733 link path=etc/shutdown target=../usr/sbin/shutdown
1734 link path=etc/sulogin target=../sbin/sulogin
1735 link path=etc/swap target=../usr/sbin/swap
1736 link path=etc/swapadd target=../sbin/swapadd
1737 link path=etc/sysdef target=../usr/sbin/sysdef
1738 link path=etc/tar target=../usr/sbin/tar
1739 link path=etc/telinit target=../sbin/init
1740 link path=etc/uadmin target=../sbin/uadmin
1741 link path=etc/umount target=../sbin/umount
1742 link path=etc/umountall target=../sbin/umountall
1743 link path=etc/utmpx target=../var/adm/utmpx
1744 link path=etc/volcopy target=../usr/sbin/volcopy
1745 link path=etc/wall target=../usr/sbin/wall
1746 link path=etc/whodo target=../usr/sbin/whodo
1747 link path=etc/wtmpx target=../var/adm/wtmpx
1748 link path=sbin/in.mpathd target=../lib/inet/in.mpathd
1749 link path=sbin/jsh target=../usr/bin/ksh93
1750 link path=sbin/pfsh target=../usr/bin/pfexec
1751 link path=sbin/sh target=../usr/bin/$(ARCH32)/ksh93
1752 link path=sbin/su target=../usr/bin/su
1753 link path=usr/adm target=../var/adm
1754 link path=usr/bin/df target=../sbin/df
1755 link path=usr/bin/jsh target=ksh93
1756 link path=usr/bin/pwconv target=../sbin/pwconv
1757 link path=usr/bin/rmail target=../mail
1758 link path=usr/bin/sh target=$(ARCH32)/ksh93
1759 link path=usr/bin/strclean target=../sbin/strclean
1760 link path=usr/bin/strerr target=../sbin/strerr
1761 link path=usr/bin/sync target=../sbin/sync
1762 link path=usr/bin/tar target=../sbin/tar
1763 link path=usr/bin/uname target=../sbin/uname
1764 link path=usr/ccs/bin/m4 target=../bin/m4
1765 link path=usr/has/bin/jsh target=sh
1766 link path=usr/has/lib/rsh target=../bin/sh
1767 link path=usr/lib/$(ARCH64)/ld.so.1 target=../lib/$(ARCH64)/ld.so.1
1768 link path=usr/lib/cron target=../etc/cron.d
1769 link path=usr/lib/devfsadm/devfsadmd target=../sbin/devfsadm
1770 link path=usr/lib/embedded_su target=../bin/su
1771 link path=usr/lib/fs/dev/mount target=../etc/fs/dev/mount
1772 link path=usr/lib/fs/hfs/mount target=../etc/fs/hfs/mount
1773 link path=usr/lib/fs/ufs/mount target=../etc/fs/ufs/mount
1774 link path=usr/lib/inet/in.mpathd target=../lib/inet/in.mpathd

```

```

1775 link path=usr/lib/ld.so.1 target=../lib/ld.so.1
1776 link path=usr/lib/locale/POSIX target=../C
1777 link path=usr/lib/rsh target=../bin/ksh93
1778 link path=usr/lib/secure/32 target=..
1779 link path=usr/lib/secure/64 target=$(ARCH64)
1780 link path=usr/mail target=../var/mail
1781 link path=usr/net/nls/listen target=../lib/saf/listen
1782 link path=usr/net/nls/nlps_server target=../lib/saf/nlps_server
1783 link path=usr/news target=../var/news
1784 link path=usr/preserve target=../var/preserve
1785 link path=usr/pub target=../share/lib/pub
1786 link path=usr/sbin/autopush target=../sbin/autopush
1787 link path=usr/sbin/bootadm target=../sbin/bootadm
1788 link path=usr/sbin/cryptoadm target=../sbin/cryptoadm
1789 link path=usr/sbin/dcopy target=../clri
1790 link path=usr/sbin/devnm target=../df
1791 link path=usr/sbin/dladm target=../sbin/dladm
1792 link path=usr/sbin/dlstat target=../sbin/dlstat
1793 link path=usr/sbin/edquota target=../lib/fs/ufs/edquota
1794 link path=usr/sbin/fdisk target=../sbin/fdisk
1795 link path=usr/sbin/fiocompress target=../sbin/fiocompress
1796 link path=usr/sbin/flowadm target=../sbin/flowadm
1797 link path=usr/sbin/flowstat target=../sbin/flowstat
1798 link path=usr/sbin/fsdb target=../clri
1799 link path=usr/sbin/fsirand target=../lib/fs/ufs/fsirand
1800 link path=usr/sbin/fssnap target=../clri
1801 link path=usr/sbin/hostconfig target=../sbin/hostconfig
1802 link path=usr/sbin/ifconfig target=../sbin/ifconfig
1803 link path=usr/sbin/inetd target=../lib/inet/inetd
1804 link path=usr/sbin/init target=../sbin/init
1805 $(i386_ONLY)link path=usr/sbin/installgrub target=../sbin/installgrub
1806 link path=usr/sbin/ipadm target=../sbin/ipadm
1807 link path=usr/sbin/impstat target=../sbin/impstat
1808 link path=usr/sbin/labelit target=../clri
1809 link path=usr/sbin/lockfs target=../lib/fs/ufs/lockfs
1810 link path=usr/sbin/mkfs target=../clri
1811 link path=usr/sbin/mount target=../sbin/mount
1812 link path=usr/sbin/ncheck target=../ff
1813 link path=usr/sbin/newfs target=../lib/fs/ufs/newfs
1814 link path=usr/sbin/quot target=../lib/fs/ufs/quot
1815 link path=usr/sbin/quota target=../lib/fs/ufs/quota
1816 link path=usr/sbin/quotacheck target=../lib/fs/ufs/quotacheck
1817 link path=usr/sbin/quotaooff target=../lib/fs/ufs/quotaooff
1818 link path=usr/sbin/quotaoon target=../lib/fs/ufs/quotaoon
1819 link path=usr/sbin/repquota target=../lib/fs/ufs/repquota
1820 link path=usr/sbin/route target=../sbin/route
1821 link path=usr/sbin/routeadm target=../sbin/routeadm
1822 link path=usr/sbin/sync target=../sbin/sync
1823 link path=usr/sbin/tunefs target=../lib/fs/ufs/tunefs
1824 link path=usr/sbin/tzreload target=../sbin/tzreload
1825 link path=usr/sbin/uadmin target=../sbin/uadmin
1826 link path=usr/sbin/ufsdump target=../lib/fs/ufs/ufsdump
1827 link path=usr/sbin/ufsrestore target=../lib/fs/ufs/ufsrestore
1828 link path=usr/sbin/umount target=../sbin/umount
1829 link path=usr/spool target=../var/spool
1830 link path=usr/src target=../share/src
1831 link path=usr/tmp target=../var/tmp
1832 link path=var/ld/32 target=..
1833 link path=var/ld/64 target=$(ARCH64)
1834 #
1835 # The bootadm binary needs the etc/release file.
1836 #
1837 depend fmri=release/name type=require
1838 #
1839 # intrd and others use the illumos-defaulted perl interpreter
1840 #

```

```
1841 depend fmri=runtime/perl$(PERL_PKGVERS) type=require
1842 #
1843 # intrd uses sun-solaris Perl modules
1844 #
1845 depend fmri=runtime/perl$(PERL_PKGVERS)/module/sun-solaris type=require
1846 #
1847 # The loadkeys binary needs the keytables.
1848 #
1849 depend fmri=system/data/keyboard/keytables type=require
1850 #
1851 # Depend on terminfo data.
1852 #
1853 depend fmri=system/data/terminfo type=require
1854 #
1855 # Depend on zoneinfo data.
1856 #
1857 depend fmri=system/data/zoneinfo type=require
1858 #
1859 # The mailx binary calls /usr/lib/sendmail provided by mailwrapper
1860 #
1861 depend fmri=system/network/mailwrapper type=require
```

new/usr/src/pkg/manifests/consolidation-osnet-osnet-message-files.mf

1

```
*****
17590 Thu Jun 30 22:02:56 2016
new/usr/src/pkg/manifests/consolidation-osnet-osnet-message-files.mf
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
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 (c) 2010, Oracle and/or its affiliates. All rights reserved.
23 # Copyright 2014 Garrett D'Amore <garrett@damore.org>
24 #
25 set name=pkg.fmri \
26     value=pkg:/consolidation/osnet/osnet-message-files@$(PKGVERS)
27 set name=pkg.description \
28     value="localizable message files for the OS-Networking consolidation"
29 set name=pkg.summary value="Localizable ON message files"
30 set name=info.classification \
31     value=org.opensolaris.category.2008:Development/System
32 #
33 #
34 # This package should not have automated dependencies generated because
35 # it provides messages only.
36 #
37 set name=org.opensolaris.nodepend value=true
38 set name=variant.arch value=$(ARCH)
39 dir path=usr group=sys
40 dir path=usr/lib
41 dir path=usr/lib/help
42 dir path=usr/lib/help/auths
43 dir path=usr/lib/help/auths/locale
44 dir path=usr/lib/help/profiles
45 dir path=usr/lib/help/profiles/locale
46 dir path=usr/lib/locale
47 dir path=usr/lib/locale/C
48 dir path=usr/lib/locale/C/LC_MESSAGES
49 dir path=usr/lib/locale/C/LC_TIME
50 dir path=usr/share
51 dir path=usr/share/lib
52 dir path=usr/share/lib/locale
53 dir path=usr/share/lib/locale/com
54 dir path=usr/share/lib/locale/com/sun
55 dir path=usr/share/lib/locale/com/sun/admin
56 dir path=usr/share/lib/locale/com/sun/admin/pm
57 dir path=usr/share/lib/locale/com/sun/admin/pm/client
58 dir path=usr/share/lib/locale/com/sun/slp
```

new/usr/src/pkg/manifests/consolidation-osnet-osnet-message-files.mf

2

```
59 file path=usr/lib/help/auths/locale/AllSolAuthsHeader.html
60 file path=usr/lib/help/auths/locale/AuditHeader.html
61 file path=usr/lib/help/auths/locale/AuthJobsAdmin.html
62 file path=usr/lib/help/auths/locale/AuthJobsUser.html
63 file path=usr/lib/help/auths/locale/AuthProfmgrAssign.html
64 file path=usr/lib/help/auths/locale/AuthProfmgrDelegate.html
65 file path=usr/lib/help/auths/locale/AuthProfmgrExecattrWrite.html
66 file path=usr/lib/help/auths/locale/AuthProfmgrRead.html
67 file path=usr/lib/help/auths/locale/AuthProfmgrWrite.html
68 file path=usr/lib/help/auths/locale/AuthReadNDMP.html
69 file path=usr/lib/help/auths/locale/AuthReadSMB.html
70 file path=usr/lib/help/auths/locale/AuthRoleAssign.html
71 file path=usr/lib/help/auths/locale/AuthRoleDelegate.html
72 file path=usr/lib/help/auths/locale/AuthRoleWrite.html
73 file path=usr/lib/help/auths/locale/BindStates.html
74 file path=usr/lib/help/auths/locale/DevAllocHeader.html
75 file path=usr/lib/help/auths/locale/DevAllocate.html
76 file path=usr/lib/help/auths/locale/DevCDRW.html
77 file path=usr/lib/help/auths/locale/DevConfig.html
78 file path=usr/lib/help/auths/locale/DevGrant.html
79 file path=usr/lib/help/auths/locale/DevRevoke.html
80 file path=usr/lib/help/auths/locale/DhcppmgrHeader.html
81 file path=usr/lib/help/auths/locale/DhcppmgrWrite.html
82 file path=usr/lib/help/auths/locale/FileChown.html
83 file path=usr/lib/help/auths/locale/FileHeader.html
84 file path=usr/lib/help/auths/locale/FileOwner.html
85 file path=usr/lib/help/auths/locale/HotplugHeader.html
86 file path=usr/lib/help/auths/locale/HotplugModify.html
87 file path=usr/lib/help/auths/locale/IdmapRules.html
88 file path=usr/lib/help/auths/locale/JobHeader.html
89 file path=usr/lib/help/auths/locale/JobsGrant.html
90 file path=usr/lib/help/auths/locale/LabelFileDowngrade.html
91 file path=usr/lib/help/auths/locale/LabelFileUpgrade.html
92 file path=usr/lib/help/auths/locale/LabelHeader.html
93 file path=usr/lib/help/auths/locale/LabelPrint.html
94 file path=usr/lib/help/auths/locale/LabelRange.html
95 file path=usr/lib/help/auths/locale/LabelServer.html
96 file path=usr/lib/help/auths/locale/LabelWinDowngrade.html
97 file path=usr/lib/help/auths/locale/LabelWinNoView.html
98 file path=usr/lib/help/auths/locale/LabelWinUpgrade.html
99 file path=usr/lib/help/auths/locale/LinkSecurity.html
100 file path=usr/lib/help/auths/locale/LoginEnable.html
101 file path=usr/lib/help/auths/locale/LoginHeader.html
102 file path=usr/lib/help/auths/locale/LoginRemote.html
103 file path=usr/lib/help/auths/locale/MailHeader.html
104 file path=usr/lib/help/auths/locale/MailQueue.html
105 file path=usr/lib/help/auths/locale/NetworkAutoconfRead.html
106 file path=usr/lib/help/auths/locale/NetworkAutoconfSelect.html
107 file path=usr/lib/help/auths/locale/NetworkAutoconfWlan.html
108 file path=usr/lib/help/auths/locale/NetworkAutoconfWrite.html
109 file path=usr/lib/help/auths/locale/NetworkHeader.html
110 file path=usr/lib/help/auths/locale/NetworkILBconf.html
111 file path=usr/lib/help/auths/locale/NetworkILBenable.html
112 file path=usr/lib/help/auths/locale/NetworkInterfaceConfig.html
113 file path=usr/lib/help/auths/locale/NetworkVRRP.html
114 file path=usr/lib/help/auths/locale/PriAdmin.html
115 file path=usr/lib/help/auths/locale/PrintAdmin.html
116 file path=usr/lib/help/auths/locale/PrintCancel.html
117 file path=usr/lib/help/auths/locale/PrintHeader.html
118 file path=usr/lib/help/auths/locale/PrintList.html
119 file path=usr/lib/help/auths/locale/PrintNoBanner.html
120 file path=usr/lib/help/auths/locale/PrintPs.html
121 file path=usr/lib/help/auths/locale/PrintUnlabeled.html
122 file path=usr/lib/help/auths/locale/ProfmgrHeader.html
123 file path=usr/lib/help/auths/locale/RoleHeader.html
124 file path=usr/lib/help/auths/locale/SmfAllocate.html
```

```
125 file path=usr/lib/help/auths/locale/SmfAutofsStates.html
126 file path=usr/lib/help/auths/locale/SmfCoreadmStates.html
127 file path=usr/lib/help/auths/locale/SmfCronStates.html
128 file path=usr/lib/help/auths/locale/SmfExAcctFlowStates.html
129 file path=usr/lib/help/auths/locale/SmfExAcctNetStates.html
130 file path=usr/lib/help/auths/locale/SmfExAcctProcessStates.html
131 file path=usr/lib/help/auths/locale/SmfExAcctTaskStates.html
132 file path=usr/lib/help/auths/locale/SmfHeader.html
133 file path=usr/lib/help/auths/locale/SmfILBStates.html
134 file path=usr/lib/help/auths/locale/SmfIPsecStates.html
135 file path=usr/lib/help/auths/locale/SmfIdmapStates.html
136 file path=usr/lib/help/auths/locale/SmfInetdStates.html
137 file path=usr/lib/help/auths/locale/SmfLocationStates.html
138 file path=usr/lib/help/auths/locale/SmfMDNSStates.html
139 file path=usr/lib/help/auths/locale/SmfManageAudit.html
140 file path=usr/lib/help/auths/locale/SmfManageHeader.html
141 file path=usr/lib/help/auths/locale/SmfManageHotplug.html
142 file path=usr/lib/help/auths/locale/SmfManageZFSSnap.html
143 file path=usr/lib/help/auths/locale/SmfModifyAppl.html
144 file path=usr/lib/help/auths/locale/SmfModifyDepend.html
145 file path=usr/lib/help/auths/locale/SmfModifyFramework.html
146 file path=usr/lib/help/auths/locale/SmfModifyHeader.html
147 file path=usr/lib/help/auths/locale/SmfModifyMethod.html
148 file path=usr/lib/help/auths/locale/SmfNADDStates.html
149 file path=usr/lib/help/auths/locale/SmfNDMPStates.html
150 file path=usr/lib/help/auths/locale/SmfNWAMStates.html
151 file path=usr/lib/help/auths/locale/SmfNscdStates.html
152 file path=usr/lib/help/auths/locale/SmfPowerStates.html
153 file path=usr/lib/help/auths/locale/SmfReparseStates.html
154 file path=usr/lib/help/auths/locale/SmfRoutingStates.html
155 file path=usr/lib/help/auths/locale/SmfSMBFSStates.html
156 file path=usr/lib/help/auths/locale/SmfSMBStates.html
157 file path=usr/lib/help/auths/locale/SmfSendmailStates.html
158 file path=usr/lib/help/auths/locale/SmfSshStates.html
159 file path=usr/lib/help/auths/locale/SmfSyslogStates.html
160 file path=usr/lib/help/auths/locale/SmfVRRPStates.html
161 file path=usr/lib/help/auths/locale/SmfValueAudit.html
162 file path=usr/lib/help/auths/locale/SmfValueCoreadm.html
163 file path=usr/lib/help/auths/locale/SmfValueExAcctFlow.html
164 file path=usr/lib/help/auths/locale/SmfValueExAcctNet.html
165 file path=usr/lib/help/auths/locale/SmfValueExAcctProcess.html
166 file path=usr/lib/help/auths/locale/SmfValueExAcctTask.html
167 file path=usr/lib/help/auths/locale/SmfValueFirewall.html
168 file path=usr/lib/help/auths/locale/SmfValueHeader.html
169 file path=usr/lib/help/auths/locale/SmfValueIPsec.html
170 file path=usr/lib/help/auths/locale/SmfValueIdmap.html
171 file path=usr/lib/help/auths/locale/SmfValueInetd.html
172 file path=usr/lib/help/auths/locale/SmfValueMDNS.html
173 file path=usr/lib/help/auths/locale/SmfValueNADD.html
174 file path=usr/lib/help/auths/locale/SmfValueNDMP.html
175 file path=usr/lib/help/auths/locale/SmfValueNWAM.html
176 file path=usr/lib/help/auths/locale/SmfValueProcSec.html
177 #endif /* ! codereview */
178 file path=usr/lib/help/auths/locale/SmfValueRouting.html
179 file path=usr/lib/help/auths/locale/SmfValueSMB.html
180 file path=usr/lib/help/auths/locale/SmfValueVscan.html
181 file path=usr/lib/help/auths/locale/SmfValueVt.html
182 file path=usr/lib/help/auths/locale/SmfVscanStates.html
183 file path=usr/lib/help/auths/locale/SmfVtStates.html
184 file path=usr/lib/help/auths/locale/SmfWpaStates.html
185 file path=usr/lib/help/auths/locale/SysCpuPowerMgmt.html
186 file path=usr/lib/help/auths/locale/SysDate.html
187 file path=usr/lib/help/auths/locale/SysHeader.html
188 file path=usr/lib/help/auths/locale/SysMaintenance.html
189 file path=usr/lib/help/auths/locale/SysPowerMgmtBrightness.html
190 file path=usr/lib/help/auths/locale/SysPowerMgmtHeader.html
```

```
191 file path=usr/lib/help/auths/locale/SysPowerMgmtSuspend.html
192 file path=usr/lib/help/auths/locale/SysPowerMgmtSuspendtoDisk.html
193 file path=usr/lib/help/auths/locale/SysPowerMgmtSuspendtoRAM.html
194 file path=usr/lib/help/auths/locale/SysShutdown.html
195 file path=usr/lib/help/auths/locale/SysSyseventRead.html
196 file path=usr/lib/help/auths/locale/SysSyseventWrite.html
197 file path=usr/lib/help/auths/locale/TNDAemon.html
198 file path=usr/lib/help/auths/locale/TNctl.html
199 file path=usr/lib/help/auths/locale/ValueTND.html
200 file path=usr/lib/help/auths/locale/WifiConfig.html
201 file path=usr/lib/help/auths/locale/WifiWep.html
202 file path=usr/lib/help/auths/locale/ZoneCloneFrom.html
203 file path=usr/lib/help/auths/locale/ZoneHeader.html
204 file path=usr/lib/help/auths/locale/ZoneLogin.html
205 file path=usr/lib/help/auths/locale/ZoneManage.html
206 file path=usr/lib/help/profiles/locale/RtAcctadm.html
207 file path=usr/lib/help/profiles/locale/RtAll.html
208 file path=usr/lib/help/profiles/locale/RtAuditCfg.html
209 file path=usr/lib/help/profiles/locale/RtAuditCtrl.html
210 file path=usr/lib/help/profiles/locale/RtAuditReview.html
211 file path=usr/lib/help/profiles/locale/RtCPUPowerManagement.html
212 file path=usr/lib/help/profiles/locale/RtConsUser.html
213 file path=usr/lib/help/profiles/locale/RtContractObserver.html
214 file path=usr/lib/help/profiles/locale/RtCronMngmnt.html
215 file path=usr/lib/help/profiles/locale/RtCryptoMngmnt.html
216 file path=usr/lib/help/profiles/locale/RtDHCPMngmnt.html
217 file path=usr/lib/help/profiles/locale/RtDatAdmin.html
218 file path=usr/lib/help/profiles/locale/RtDefault.html
219 file path=usr/lib/help/profiles/locale/RtDeviceMngmnt.html
220 file path=usr/lib/help/profiles/locale/RtDeviceSecurity.html
221 file path=usr/lib/help/profiles/locale/RtExAcctFlow.html
222 file path=usr/lib/help/profiles/locale/RtExAcctNet.html
223 file path=usr/lib/help/profiles/locale/RtExAcctProcess.html
224 file path=usr/lib/help/profiles/locale/RtExAcctTask.html
225 file path=usr/lib/help/profiles/locale/RtFTPMngmnt.html
226 file path=usr/lib/help/profiles/locale/RtFileSysMngmnt.html
227 file path=usr/lib/help/profiles/locale/RtFileSysSecurity.html
228 file path=usr/lib/help/profiles/locale/RtHotplugMngmnt.html
229 file path=usr/lib/help/profiles/locale/RtIPFilterMngmnt.html
230 file path=usr/lib/help/profiles/locale/RtIdmapMngmnt.html
231 file path=usr/lib/help/profiles/locale/RtIdmapNameRulesMngmnt.html
232 file path=usr/lib/help/profiles/locale/RtInetdMngmnt.html
233 file path=usr/lib/help/profiles/locale/RtInfoSec.html
234 file path=usr/lib/help/profiles/locale/RtKerberosClntMngmnt.html
235 file path=usr/lib/help/profiles/locale/RtKerberosSrvrMngmnt.html
236 file path=usr/lib/help/profiles/locale/RtLogMngmnt.html
237 file path=usr/lib/help/profiles/locale/RtMailMngmnt.html
238 file path=usr/lib/help/profiles/locale/RtMaintAndRepair.html
239 file path=usr/lib/help/profiles/locale/RtMediaBkup.html
240 file path=usr/lib/help/profiles/locale/RtMediaCtlg.html
241 file path=usr/lib/help/profiles/locale/RtMediaRestore.html
242 file path=usr/lib/help/profiles/locale/RtNDMPMngmnt.html
243 file path=usr/lib/help/profiles/locale/RtNameServiceAdmin.html
244 file path=usr/lib/help/profiles/locale/RtNameServiceSecure.html
245 file path=usr/lib/help/profiles/locale/RtNetAutoconfAdmin.html
246 file path=usr/lib/help/profiles/locale/RtNetAutoconfUser.html
247 file path=usr/lib/help/profiles/locale/RtNetILB.html
248 file path=usr/lib/help/profiles/locale/RtNetIPsec.html
249 file path=usr/lib/help/profiles/locale/RtNetLinkSecure.html
250 file path=usr/lib/help/profiles/locale/RtNetMngmnt.html
251 file path=usr/lib/help/profiles/locale/RtNetObservability.html
252 file path=usr/lib/help/profiles/locale/RtNetSecure.html
253 file path=usr/lib/help/profiles/locale/RtNetVRRP.html
254 file path=usr/lib/help/profiles/locale/RtNetWifiMngmnt.html
255 file path=usr/lib/help/profiles/locale/RtNetWifiSecure.html
256 file path=usr/lib/help/profiles/locale/RtObAccessMngmnt.html
```

```

257 file path=usr/lib/help/profiles/locale/RtObjectLabelMngmnt.html
258 file path=usr/lib/help/profiles/locale/RtOperator.html
259 file path=usr/lib/help/profiles/locale/RtOutsideAccred.html
260 file path=usr/lib/help/profiles/locale/RtPriAdmin.html
261 file path=usr/lib/help/profiles/locale/RtPrntAdmin.html
262 file path=usr/lib/help/profiles/locale/RtProcManagement.html
263 file path=usr/lib/help/profiles/locale/RtReparseMngmnt.html
264 file path=usr/lib/help/profiles/locale/RtReservedProfile.html
265 file path=usr/lib/help/profiles/locale/RtRightsDelegate.html
266 file path=usr/lib/help/profiles/locale/RtSMBFSMngmnt.html
267 file path=usr/lib/help/profiles/locale/RtSMBMngmnt.html
268 file path=usr/lib/help/profiles/locale/RtSoftwareInstall.html
269 file path=usr/lib/help/profiles/locale/RtSysAdmin.html
270 file path=usr/lib/help/profiles/locale/RtSysEvMngmnt.html
271 file path=usr/lib/help/profiles/locale/RtSysPowerMgmt.html
272 file path=usr/lib/help/profiles/locale/RtSysPowerMgmtBrightness.html
273 file path=usr/lib/help/profiles/locale/RtSysPowerMgmtSuspend.html
274 file path=usr/lib/help/profiles/locale/RtSysPowerMgmtSuspendtoDisk.html
275 file path=usr/lib/help/profiles/locale/RtSysPowerMgmtSuspendtoRAM.html
276 file path=usr/lib/help/profiles/locale/RtUserMngmnt.html
277 file path=usr/lib/help/profiles/locale/RtUserSecurity.html
278 file path=usr/lib/help/profiles/locale/RtVscanMngmnt.html
279 file path=usr/lib/help/profiles/locale/RtZFSFileSysMngmnt.html
280 file path=usr/lib/help/profiles/locale/RtZFSStorageMngmnt.html
281 file path=usr/lib/help/profiles/locale/RtZoneMngmnt.html
282 file path=usr/lib/help/profiles/locale/RtZoneSecurity.html
283 file path=usr/lib/locale/C/LC_MESSAGES/AMD.po
284 file path=usr/lib/locale/C/LC_MESSAGES/DISK.po
285 file path=usr/lib/locale/C/LC_MESSAGES/FMD.po
286 file path=usr/lib/locale/C/LC_MESSAGES/FMNOTIFY.po
287 file path=usr/lib/locale/C/LC_MESSAGES/GMCA.po
288 file path=usr/lib/locale/C/LC_MESSAGES/INTEL.po
289 file path=usr/lib/locale/C/LC_MESSAGES/NXGE.po
290 file path=usr/lib/locale/C/LC_MESSAGES/PCI.po
291 file path=usr/lib/locale/C/LC_MESSAGES/PCIEX.po
292 file path=usr/lib/locale/C/LC_MESSAGES/SCA1000.po
293 file path=usr/lib/locale/C/LC_MESSAGES/SCA500.po
294 file path=usr/lib/locale/C/LC_MESSAGES/SCF.po
295 file path=usr/lib/locale/C/LC_MESSAGES/SENSOR.po
296 file path=usr/lib/locale/C/LC_MESSAGES/SMF.po
297 file path=usr/lib/locale/C/LC_MESSAGES/STORAGE.po
298 file path=usr/lib/locale/C/LC_MESSAGES/SUN4.po
299 file path=usr/lib/locale/C/LC_MESSAGES/SUN4U.po
300 file path=usr/lib/locale/C/LC_MESSAGES/SUN4V.po
301 file path=usr/lib/locale/C/LC_MESSAGES/SUNOS.po
302 file path=usr/lib/locale/C/LC_MESSAGES/SUNW_ADMIN.po group=sys
303 file path=usr/lib/locale/C/LC_MESSAGES/SUNW_OST_LINFO group=sys
304 file path=usr/lib/locale/C/LC_MESSAGES/SUNW_OST_NETRPC.po group=sys
305 file path=usr/lib/locale/C/LC_MESSAGES/SUNW_OST_OSCMD.po group=sys
306 file path=usr/lib/locale/C/LC_MESSAGES/SUNW_OST_OSLIB.po group=sys
307 file path=usr/lib/locale/C/LC_MESSAGES/SUNW_OST_SGS.po group=sys
308 file path=usr/lib/locale/C/LC_MESSAGES/SUNW_OST_SYSPAM.po group=sys
309 file path=usr/lib/locale/C/LC_MESSAGES/SUNW_OST_UCBCMD.po group=sys
310 file path=usr/lib/locale/C/LC_MESSAGES/SUNW_OST_ZONEINFO.po group=sys
311 file path=usr/lib/locale/C/LC_MESSAGES/ZFS.po
312 file path=usr/lib/locale/C/LC_MESSAGES/libast group=sys
313 file path=usr/lib/locale/C/LC_MESSAGES/libcmd group=sys
314 file path=usr/lib/locale/C/LC_MESSAGES/libdll group=sys
315 file path=usr/lib/locale/C/LC_MESSAGES/libshell group=sys
316 file path=usr/lib/locale/C/LC_MESSAGES/libsum group=sys
317 file path=usr/lib/locale/C/LC_MESSAGES/magic group=sys
318 file path=usr/lib/locale/C/LC_MESSAGES/mailx.help group=sys
319 file path=usr/lib/locale/C/LC_MESSAGES/more.help group=sys
320 file path=usr/lib/locale/C/LC_MESSAGES/priv_names group=sys
321 file path=usr/lib/locale/C/LC_MESSAGES/uxlibc.src group=sys
322 file path=usr/lib/locale/C/LC_TIME/SUNW_OST_OSCMD.po group=sys

```

```

323 file path=usr/lib/locale/C/LC_TIME/SUNW_OST_OSLIB.po group=sys
324 file path=usr/share/lib/locale/com/sun/admin/pm/client/pmHelpResources.java \
325 group=lp
326 file path=usr/share/lib/locale/com/sun/admin/pm/client/pmResources.java \
327 group=lp
328 file path=usr/share/lib/locale/com/sun/slp/ClientLib_en.properties group=sys
329 file path=usr/share/lib/locale/com/sun/slp/Server_en.properties group=sys
330 legacy pkg=SUNW0on arch=all \
331 desc="localizable message files for the OS-Networking consolidation" \
332 name="Localizable ON message files" version=11.11,REV=2009.11.10
333 license cr_Sun license=cr_Sun
334 license lic_CDDL license=lic_CDDL

```



```

*****
12099 Thu Jun 30 22:02:57 2016
new/usr/src/pkg/manifests/system-extended-system-utilities.mf
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
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 #
23 # Copyright (c) 2010, Oracle and/or its affiliates. All rights reserved.
24 # Copyright 2012 Nexenta Systems, Inc. All rights reserved.
25 #
26 #
27 set name=pkg.fmri value=pkg:/system/extended-system-utilities@$(PKGVERS)
28 set name=pkg.description \
29     value="additional UNIX system utilities, including awk, bc, cal, compress, d
30 set name=pkg.summary value="Extended System Utilities"
31 set name=info.classification value=org.opensolaris.category.2008:System/Core
32 set name=variant.arch value=$(ARCH)
33 dir path=usr group=sys
34 dir path=usr/bin
35 $(i386_ONLY)dir path=usr/bin/$(ARCH32)
36 dir path=usr/bin/$(ARCH64)
37 dir path=usr/lib
38 dir path=usr/lib/$(ARCH64)
39 dir path=usr/lib/adb group=sys
40 dir path=usr/lib/adb/$(ARCH64) group=sys
41 dir path=usr/lib/fs group=sys
42 dir path=usr/lib/fs/pcfs group=sys
43 dir path=usr/lib/spell
44 dir path=usr/proc
45 dir path=usr/proc/bin
46 dir path=usr/sbin
47 dir path=usr/sbin/$(ARCH64)
48 dir path=usr/share
49 dir path=usr/share/lib
50 dir path=usr/share/lib/dict
51 dir path=usr/share/man/man1
52 dir path=usr/share/man/man1m
53 $(i386_ONLY)file path=usr/bin/$(ARCH32)/pargs mode=0555
54 $(i386_ONLY)file path=usr/bin/$(ARCH32)/pcred mode=0555
55 $(i386_ONLY)file path=usr/bin/$(ARCH32)/pfiles mode=0555
56 $(i386_ONLY)file path=usr/bin/$(ARCH32)/pflags mode=0555
57 $(i386_ONLY)file path=usr/bin/$(ARCH32)/pldd mode=0555
58 $(i386_ONLY)file path=usr/bin/$(ARCH32)/plgrp mode=0555

```

```

59 $(i386_ONLY)file path=usr/bin/$(ARCH32)/pmadvise mode=0555
60 $(i386_ONLY)file path=usr/bin/$(ARCH32)/pmap mode=0555
61 $(i386_ONLY)file path=usr/bin/$(ARCH32)/ppgsz mode=0555
62 $(i386_ONLY)file path=usr/bin/$(ARCH32)/ppriv mode=0555
63 $(i386_ONLY)file path=usr/bin/$(ARCH32)/preap mode=0555
64 $(i386_ONLY)file path=usr/bin/$(ARCH32)/prun mode=0555
65 $(i386_ONLY)file path=usr/bin/$(ARCH32)/psecflags mode=0555
66 #endif /* ! codereview */
67 $(i386_ONLY)file path=usr/bin/$(ARCH32)/psig mode=0555
68 $(i386_ONLY)file path=usr/bin/$(ARCH32)/pstack mode=0555
69 $(i386_ONLY)file path=usr/bin/$(ARCH32)/pstop mode=0555
70 $(i386_ONLY)file path=usr/bin/$(ARCH32)/ptime mode=0555
71 $(i386_ONLY)file path=usr/bin/$(ARCH32)/ptree mode=0555
72 $(i386_ONLY)file path=usr/bin/$(ARCH32)/pwait mode=0555
73 $(i386_ONLY)file path=usr/bin/$(ARCH32)/pwdx mode=0555
74 $(i386_ONLY)file path=usr/bin/$(ARCH32)/sort mode=0555
75 file path=usr/bin/$(ARCH64)/pargs mode=0555
76 file path=usr/bin/$(ARCH64)/pcred mode=0555
77 file path=usr/bin/$(ARCH64)/pfiles mode=0555
78 file path=usr/bin/$(ARCH64)/pflags mode=0555
79 file path=usr/bin/$(ARCH64)/pldd mode=0555
80 file path=usr/bin/$(ARCH64)/plgrp mode=0555
81 file path=usr/bin/$(ARCH64)/pmadvise mode=0555
82 file path=usr/bin/$(ARCH64)/pmap mode=0555
83 file path=usr/bin/$(ARCH64)/ppgsz mode=0555
84 file path=usr/bin/$(ARCH64)/ppriv mode=0555
85 file path=usr/bin/$(ARCH64)/preap mode=0555
86 file path=usr/bin/$(ARCH64)/prun mode=0555
87 file path=usr/bin/$(ARCH64)/psecflags mode=0555
88 #endif /* ! codereview */
89 file path=usr/bin/$(ARCH64)/psig mode=0555
90 file path=usr/bin/$(ARCH64)/pstack mode=0555
91 file path=usr/bin/$(ARCH64)/pstop mode=0555
92 file path=usr/bin/$(ARCH64)/ptime mode=0555
93 file path=usr/bin/$(ARCH64)/ptree mode=0555
94 file path=usr/bin/$(ARCH64)/pwait mode=0555
95 file path=usr/bin/$(ARCH64)/pwdx mode=0555
96 file path=usr/bin/$(ARCH64)/sort mode=0555
97 file path=usr/bin/asa mode=0555
98 file path=usr/bin/awk mode=0555
99 file path=usr/bin/banner mode=0555
100 file path=usr/bin/batch mode=0555
101 file path=usr/bin/bc mode=0555
102 file path=usr/bin/bdiff mode=0755
103 file path=usr/bin/cal mode=0555
104 file path=usr/bin/calendar mode=0555
105 file path=usr/bin/col mode=0555
106 file path=usr/bin/compress mode=0555
107 file path=usr/bin/csplite mode=0555
108 file path=usr/bin/dc mode=0555
109 file path=usr/bin/diff mode=0555
110 file path=usr/bin/diff3 mode=0555
111 file path=usr/bin/dircmp mode=0555
112 file path=usr/bin/dos2unix mode=0555
113 file path=usr/bin/expand mode=0555
114 file path=usr/bin/factor mode=0555
115 file path=usr/bin/kstat mode=0555
116 file path=usr/bin/last mode=0555
117 file path=usr/bin/lastcomm mode=0555
118 file path=usr/bin/lgrpinfo mode=0555
119 file path=usr/bin/look mode=0755
120 file path=usr/bin/mkfifo mode=0555
121 file path=usr/bin/nawk mode=0555
122 file path=usr/bin/newform mode=0555
123 file path=usr/bin/news mode=0555
124 file path=usr/bin/nl mode=0555

```

```

125 file path=usr/bin/pack mode=0555
126 file path=usr/bin/pginfo mode=0555
127 file path=usr/bin/pgstat mode=0555
128 file path=usr/bin/sdiff mode=0555
129 file path=usr/bin/spell mode=0555
130 file path=usr/bin/split mode=0555
131 file path=usr/bin/tcopy mode=0555
132 file path=usr/bin/unexpand mode=0555
133 file path=usr/bin/units mode=0555
134 file path=usr/bin/unix2dos mode=0555
135 file path=usr/bin/unpack mode=0555
136 file path=usr/bin/uudecode group=uucp mode=0555
137 file path=usr/bin/uencode group=uucp mode=0555
138 file path=usr/bin/yes mode=0555
139 file path=usr/lib/$(ARCH64)/madv.so.1
140 file path=usr/lib/$(ARCH64)/mpss.so.1
141 file path=usr/lib/adb/$(ARCH64)/adbsub.o group=sys
142 file path=usr/lib/adb/adbgen group=sys mode=0755
143 file path=usr/lib/adb/adbgen1 group=sys mode=0755
144 file path=usr/lib/adb/adbgen3 group=sys mode=0755
145 file path=usr/lib/adb/adbgen4 group=sys mode=0755
146 file path=usr/lib/adb/adbsub.o group=sys
147 file path=usr/lib/diff3prog mode=0555
148 file path=usr/lib/fs/pcfs/fsck mode=0555
149 file path=usr/lib/fs/pcfs/fstyp.so.1 mode=0555
150 file path=usr/lib/fs/pcfs/mkfs mode=0555
151 file path=usr/lib/fs/pcfs/mount mode=0555
152 file path=usr/lib/madv.so.1
153 file path=usr/lib/mpss.so.1
154 file path=usr/lib/spell/compress mode=0555
155 file path=usr/lib/spell/hashcheck mode=0555
156 file path=usr/lib/spell/hashmake mode=0555
157 file path=usr/lib/spell/hlستا
158 file path=usr/lib/spell/hlistb
159 file path=usr/lib/spell/hstop
160 file path=usr/lib/spell/spellin mode=0555
161 file path=usr/lib/spell/spellprog mode=0555
162 file path=usr/sbin/dmesg mode=0555
163 file path=usr/sbin/projadd group=sys mode=0555
164 file path=usr/sbin/projdel group=sys mode=0555
165 file path=usr/sbin/projmod group=sys mode=0555
166 file path=usr/share/lib/dict/words
167 file path=usr/share/man/man1/asa.1
168 file path=usr/share/man/man1/awk.1
169 file path=usr/share/man/man1/banner.1
170 file path=usr/share/man/man1/bc.1
171 file path=usr/share/man/man1/bdiff.1
172 file path=usr/share/man/man1/cal.1
173 file path=usr/share/man/man1/calendar.1
174 file path=usr/share/man/man1/col.1
175 file path=usr/share/man/man1/compress.1
176 file path=usr/share/man/man1/csplit.1
177 file path=usr/share/man/man1/dc.1
178 file path=usr/share/man/man1/diff.1
179 file path=usr/share/man/man1/diff3.1
180 file path=usr/share/man/man1/dircomp.1
181 file path=usr/share/man/man1/dos2unix.1
182 file path=usr/share/man/man1/expand.1
183 file path=usr/share/man/man1/factor.1
184 file path=usr/share/man/man1/last.1
185 file path=usr/share/man/man1/lastcomm.1
186 file path=usr/share/man/man1/lgrpinfo.1
187 file path=usr/share/man/man1/look.1
188 file path=usr/share/man/man1/madv.so.1.1
189 file path=usr/share/man/man1/mpss.so.1.1
190 file path=usr/share/man/man1/nawk.1

```

```

191 file path=usr/share/man/man1/newform.1
192 file path=usr/share/man/man1/news.1
193 file path=usr/share/man/man1/nl.1
194 file path=usr/share/man/man1/pack.1
195 file path=usr/share/man/man1/pargs.1
196 file path=usr/share/man/man1/plgrp.1
197 file path=usr/share/man/man1/pmadvise.1
198 file path=usr/share/man/man1/pmap.1
199 file path=usr/share/man/man1/ppgsz.1
200 file path=usr/share/man/man1/ppriv.1
201 file path=usr/share/man/man1/preap.1
202 file path=usr/share/man/man1/psecflags.1
203 #endif /* ! codereview */
204 file path=usr/share/man/man1/ptree.1
205 file path=usr/share/man/man1/sdiff.1
206 file path=usr/share/man/man1/sort.1
207 file path=usr/share/man/man1/spell.1
208 file path=usr/share/man/man1/split.1
209 file path=usr/share/man/man1/tcopy.1
210 file path=usr/share/man/man1/units.1
211 file path=usr/share/man/man1/unix2dos.1
212 file path=usr/share/man/man1/yes.1
213 file path=usr/share/man/man1m/adbgen.1m
214 file path=usr/share/man/man1m/dmesg.1m
215 file path=usr/share/man/man1m/fsck_pcfs.1m
216 file path=usr/share/man/man1m/kstat.1m
217 file path=usr/share/man/man1m/mkfifo.1m
218 file path=usr/share/man/man1m/mkfs_pcfs.1m
219 file path=usr/share/man/man1m/mount_pcfs.1m
220 file path=usr/share/man/man1m/projadd.1m
221 file path=usr/share/man/man1m/projdel.1m
222 file path=usr/share/man/man1m/projmod.1m
223 hardlink path=usr/bin/awk target=../usr/bin/awk
224 hardlink path=usr/bin/pargs target=../usr/lib/isaexec
225 hardlink path=usr/bin/pcred target=../usr/lib/isaexec
226 hardlink path=usr/bin/pfiles target=../usr/lib/isaexec
227 hardlink path=usr/bin/pflags target=../usr/lib/isaexec
228 hardlink path=usr/bin/pldd target=../usr/lib/isaexec
229 hardlink path=usr/bin/plgrp target=../usr/lib/isaexec
230 hardlink path=usr/bin/pmadvise target=../usr/lib/isaexec
231 hardlink path=usr/bin/pmap target=../usr/lib/isaexec
232 hardlink path=usr/bin/ppgsz target=../usr/lib/isaexec
233 hardlink path=usr/bin/ppriv target=../usr/lib/isaexec
234 hardlink path=usr/bin/preap target=../usr/lib/isaexec
235 hardlink path=usr/bin/prun target=../usr/lib/isaexec
236 hardlink path=usr/bin/psecflags target=../usr/lib/isaexec
237 #endif /* ! codereview */
238 hardlink path=usr/bin/psig target=../usr/lib/isaexec
239 hardlink path=usr/bin/pstack target=../usr/lib/isaexec
240 hardlink path=usr/bin/pstop target=../usr/lib/isaexec
241 hardlink path=usr/bin/ptime target=../usr/lib/isaexec
242 hardlink path=usr/bin/ptree target=../usr/lib/isaexec
243 hardlink path=usr/bin/pwait target=../usr/lib/isaexec
244 hardlink path=usr/bin/pwdx target=../usr/lib/isaexec
245 hardlink path=usr/bin/sort target=../usr/lib/isaexec
246 hardlink path=usr/bin/uncompress target=../usr/bin/compress
247 hardlink path=usr/bin/zcat target=../usr/bin/compress
248 hardlink path=usr/lib/fs/pcfs/fstyp target=../usr/sbin/fstyp
249 legacy pkg=SUNWesu \
250     desc="additional UNIX system utilities, including awk, bc, cal, compress, di
251     name="Extended System Utilities"
252 license cr_Sun license=cr_Sun
253 license lic_CDDL license=lic_CDDL
254 license usr/src/cmd/compress/THIRDPARTYLICENSE \
255     license=usr/src/cmd/compress/THIRDPARTYLICENSE
256 license usr/src/cmd/lastcomm/THIRDPARTYLICENSE \

```

```
257 license=usr/src/cmd/lastcomm/THIRDPARTYLICENSE
258 license usr/src/cmd/look/THIRDPARTYLICENSE \
259 license=usr/src/cmd/look/THIRDPARTYLICENSE
260 license usr/src/cmd/units/THIRDPARTYLICENSE \
261 license=usr/src/cmd/units/THIRDPARTYLICENSE
262 link path=usr/bin/dmesg target=../sbin/dmesg
263 link path=usr/bin/pcat target=../unpack
264 link path=usr/bin/strace target=../sbin/strace
265 link path=usr/dict target=../share/lib/dict
266 link path=usr/proc/bin/pcred target=../bin/pcred
267 link path=usr/proc/bin/pfiles target=../bin/pfiles
268 link path=usr/proc/bin/pflags target=../bin/pflags
269 link path=usr/proc/bin/pldd target=../bin/pldd
270 link path=usr/proc/bin/pmap target=../bin/pmap
271 link path=usr/proc/bin/prun target=../bin/prun
272 link path=usr/proc/bin/psig target=../bin/psig
273 link path=usr/proc/bin/pstack target=../bin/pstack
274 link path=usr/proc/bin/pstop target=../bin/pstop
275 link path=usr/proc/bin/ptime target=../bin/ptime
276 link path=usr/proc/bin/ptree target=../bin/ptree
277 link path=usr/proc/bin/pwait target=../bin/pwait
278 link path=usr/proc/bin/pwdx target=../bin/pwdx
279 link path=usr/share/man/man1/hashcheck.1 target=spell.1
280 link path=usr/share/man/man1/hashmake.1 target=spell.1
281 link path=usr/share/man/man1/pcat.1 target=pack.1
282 link path=usr/share/man/man1/spellin.1 target=spell.1
283 link path=usr/share/man/man1/uncompress.1 target=compress.1
284 link path=usr/share/man/man1/unexpand.1 target=expand.1
285 link path=usr/share/man/man1/unpack.1 target=pack.1
286 link path=usr/share/man/man1/zcat.1 target=compress.1
287 depend fmri=runtime/perl$(PERL_PKGVERS) type=require
```

new/usr/src/pkg/manifests/system-header.mf

1

```
*****
90536 Thu Jun 30 22:02:57 2016
new/usr/src/pkg/manifests/system-header.mf
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
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 #
23 # Copyright (c) 2010, Oracle and/or its affiliates. All rights reserved.
24 # Copyright (c) 2012 by Delphix. All rights reserved.
25 # Copyright 2013 Saso Kiselkov. All rights reserved.
26 # Copyright 2014 Garrett D'Amore <garrett@damore.org>
27 # Copyright 2016 Nexenta Systems, Inc.
28 #
29 #
30 set name=pkg.fmri value=pkg:/system/header@$(PKGVERS)
31 set name=pkg.description \
32   value="SunOS C/C++ header files for general development of software"
33 set name=pkg.summary value="SunOS Header Files"
34 set name=info.classification value=org.opensolaris.category.2008:System/Core
35 set name=variant.arch value=$(ARCH)
36 dir path=usr group=sys
37 dir path=usr/include
38 $(i386_ONLY)dir path=usr/include/$(ARCH64)
39 $(i386_ONLY)dir path=usr/include/$(ARCH64)/sys
40 dir path=usr/include/ads
41 dir path=usr/include/arpa
42 dir path=usr/include/asm
43 dir path=usr/include/ast
44 dir path=usr/include/bsm
45 dir path=usr/include/dat
46 dir path=usr/include/des
47 dir path=usr/include/gssapi
48 dir path=usr/include/hal
49 $(i386_ONLY)dir path=usr/include/ia32
50 $(i386_ONLY)dir path=usr/include/ia32/sys
51 dir path=usr/include/inet
52 dir path=usr/include/inet/kssl
53 dir path=usr/include/ipp
54 dir path=usr/include/ipp/ipgpc
55 dir path=usr/include/iso
56 dir path=usr/include/kerberosv5
57 dir path=usr/include/libpolkit
58 dir path=usr/include/net
```

new/usr/src/pkg/manifests/system-header.mf

2

```
59 dir path=usr/include/netinet
60 dir path=usr/include/nfs
61 dir path=usr/include/protocols
62 dir path=usr/include/rpc
63 dir path=usr/include/rpcsvc
64 dir path=usr/include/sasl
65 dir path=usr/include/scsi
66 dir path=usr/include/scsi/plugins
67 dir path=usr/include/scsi/plugins/ses
68 dir path=usr/include/scsi/plugins/ses/framework
69 dir path=usr/include/scsi/plugins/ses/vendor
70 dir path=usr/include/scsi/plugins/smp
71 dir path=usr/include/scsi/plugins/smp/engine
72 dir path=usr/include/scsi/plugins/smp/framework
73 dir path=usr/include/security
74 dir path=usr/include/sharefs
75 dir path=usr/include/sys
76 dir path=usr/include/sys/av
77 dir path=usr/include/sys/contract
78 dir path=usr/include/sys/crypto
79 dir path=usr/include/sys/dktp
80 dir path=usr/include/sys/fc4
81 dir path=usr/include/sys/fm
82 dir path=usr/include/sys/fm/cpu
83 dir path=usr/include/sys/fm/fs
84 dir path=usr/include/sys/fm/io
85 $(sparc_ONLY)dir path=usr/include/sys/fpu
86 dir path=usr/include/sys/fs
87 dir path=usr/include/sys/hotplug
88 dir path=usr/include/sys/hotplug/pci
89 dir path=usr/include/sys/ib
90 dir path=usr/include/sys/ib/adapters
91 dir path=usr/include/sys/ib/adapters/hermon
92 dir path=usr/include/sys/ib/adapters/tavor
93 dir path=usr/include/sys/ib/clients
94 dir path=usr/include/sys/ib/clients/ibd
95 dir path=usr/include/sys/ib/clients/of
96 dir path=usr/include/sys/ib/clients/of/rdma
97 dir path=usr/include/sys/ib/clients/of/sol_ofs
98 dir path=usr/include/sys/ib/clients/of/sol_ucma
99 dir path=usr/include/sys/ib/clients/of/sol_umad
100 dir path=usr/include/sys/ib/clients/of/sol_uverbs
101 dir path=usr/include/sys/ib/ibnex
102 dir path=usr/include/sys/ib/ibt1
103 dir path=usr/include/sys/ib/ibt1/impl
104 dir path=usr/include/sys/ib/mgt
105 dir path=usr/include/sys/ib/mgt/ibmf
106 dir path=usr/include/sys/iso
107 dir path=usr/include/sys/lvm
108 dir path=usr/include/sys/proc
109 dir path=usr/include/sys/rsm
110 $(i386_ONLY)dir path=usr/include/sys/sata group=sys
111 dir path=usr/include/sys/scsi
112 dir path=usr/include/sys/scsi/adapters
113 dir path=usr/include/sys/scsi/conf
114 dir path=usr/include/sys/scsi/generic
115 dir path=usr/include/sys/scsi/impl
116 dir path=usr/include/sys/scsi/targets
117 dir path=usr/include/sys/sysevent
118 dir path=usr/include/sys/tsol
119 dir path=usr/include/tsol
120 dir path=usr/include/uuid
121 $(sparc_ONLY)dir path=usr/include/v7
122 $(sparc_ONLY)dir path=usr/include/v7/sys
123 $(sparc_ONLY)dir path=usr/include/v9
124 $(sparc_ONLY)dir path=usr/include/v9/sys
```

```

125 dir path=usr/include/vm
126 dir path=usr/platform/group=sys
127 $(sparc_ONLY)dir path=usr/platform/SUNW,A70 group=sys
128 $(sparc_ONLY)dir path=usr/platform/SUNW,Netra-CP2300 group=sys
129 $(sparc_ONLY)dir path=usr/platform/SUNW,Netra-CP2300/include
130 $(sparc_ONLY)dir path=usr/platform/SUNW,Netra-CP3010 group=sys
131 $(sparc_ONLY)dir path=usr/platform/SUNW,Netra-CP3010/include
132 $(sparc_ONLY)dir path=usr/platform/SUNW,Netra-T12 group=sys
133 $(sparc_ONLY)dir path=usr/platform/SUNW,Netra-T4 group=sys
134 $(sparc_ONLY)dir path=usr/platform/SUNW,SPARC-Enterprise group=sys
135 $(sparc_ONLY)dir path=usr/platform/SUNW,Serverblade1 group=sys
136 $(sparc_ONLY)dir path=usr/platform/SUNW,Sun-Blade-100 group=sys
137 $(sparc_ONLY)dir path=usr/platform/SUNW,Sun-Blade-1000 group=sys
138 $(sparc_ONLY)dir path=usr/platform/SUNW,Sun-Blade-1500 group=sys
139 $(sparc_ONLY)dir path=usr/platform/SUNW,Sun-Blade-2500 group=sys
140 $(sparc_ONLY)dir path=usr/platform/SUNW,Sun-Fire group=sys
141 $(sparc_ONLY)dir path=usr/platform/SUNW,Sun-Fire-15000 group=sys
142 $(sparc_ONLY)dir path=usr/platform/SUNW,Sun-Fire-280R group=sys
143 $(sparc_ONLY)dir path=usr/platform/SUNW,Sun-Fire-480R group=sys
144 $(sparc_ONLY)dir path=usr/platform/SUNW,Sun-Fire-880 group=sys
145 $(sparc_ONLY)dir path=usr/platform/SUNW,Sun-Fire-V215 group=sys
146 $(sparc_ONLY)dir path=usr/platform/SUNW,Sun-Fire-V240 group=sys
147 $(sparc_ONLY)dir path=usr/platform/SUNW,Sun-Fire-V250 group=sys
148 $(sparc_ONLY)dir path=usr/platform/SUNW,Sun-Fire-V440 group=sys
149 $(sparc_ONLY)dir path=usr/platform/SUNW,Sun-Fire-V445 group=sys
150 $(sparc_ONLY)dir path=usr/platform/SUNW,Sun-Fire-V490 group=sys
151 $(sparc_ONLY)dir path=usr/platform/SUNW,Sun-Fire-V890 group=sys
152 $(sparc_ONLY)dir path=usr/platform/SUNW,Ultra-2 group=sys
153 $(sparc_ONLY)dir path=usr/platform/SUNW,Ultra-250 group=sys
154 $(sparc_ONLY)dir path=usr/platform/SUNW,Ultra-4 group=sys
155 $(sparc_ONLY)dir path=usr/platform/SUNW,Ultra-Enterprise group=sys
156 $(sparc_ONLY)dir path=usr/platform/SUNW,Ultra-Enterprise-10000 group=sys
157 $(sparc_ONLY)dir path=usr/platform/SUNW,UltraSPARC-IIe-NetraCT-40 group=sys
158 $(sparc_ONLY)dir path=usr/platform/SUNW,UltraSPARC-IIe-NetraCT-60 group=sys
159 $(sparc_ONLY)dir path=usr/platform/SUNW,UltraSPARC-IIi-Netract group=sys
160 $(i386_ONLY)dir path=usr/platform/i86pc group=sys
161 $(i386_ONLY)dir path=usr/platform/i86pc/include
162 $(i386_ONLY)dir path=usr/platform/i86pc/include/sys
163 $(i386_ONLY)dir path=usr/platform/i86pc/include/vm
164 $(i386_ONLY)dir path=usr/platform/i86xpv group=sys
165 $(i386_ONLY)dir path=usr/platform/i86xpv/include
166 $(i386_ONLY)dir path=usr/platform/i86xpv/include/sys
167 $(i386_ONLY)dir path=usr/platform/i86xpv/include/vm
168 $(sparc_ONLY)dir path=usr/platform/sun4u group=sys
169 $(sparc_ONLY)dir path=usr/platform/sun4u/include
170 $(sparc_ONLY)dir path=usr/platform/sun4u/include/sys
171 $(sparc_ONLY)dir path=usr/platform/sun4u/include/sys/i2c
172 $(sparc_ONLY)dir path=usr/platform/sun4u/include/sys/i2c/clients
173 $(sparc_ONLY)dir path=usr/platform/sun4u/include/sys/i2c/misc
174 $(sparc_ONLY)dir path=usr/platform/sun4u/include/vm
175 $(sparc_ONLY)dir path=usr/platform/sun4v group=sys
176 $(sparc_ONLY)dir path=usr/platform/sun4v/include
177 $(sparc_ONLY)dir path=usr/platform/sun4v/include/sys
178 $(sparc_ONLY)dir path=usr/platform/sun4v/include/vm
179 dir path=usr/share
180 dir path=usr/share/man
181 dir path=usr/share/man/man3head
182 dir path=usr/share/man/man4
183 dir path=usr/share/man/man5
184 dir path=usr/share/man/man7i
185 dir path=usr/share/src group=sys
186 dir path=usr/share/src/uts
187 $(i386_ONLY)dir path=usr/share/src/uts/i86pc
188 $(i386_ONLY)dir path=usr/share/src/uts/i86xpv
189 $(sparc_ONLY)dir path=usr/share/src/uts/sun4u
190 $(sparc_ONLY)dir path=usr/share/src/uts/sun4v

```

```

191 dir path=usr/xpg4
192 dir path=usr/xpg4/include
193 $(i386_ONLY)file path=usr/include/$(ARCH64)/sys/kdi_regs.h
194 $(i386_ONLY)file path=usr/include/$(ARCH64)/sys/privmregs.h
195 $(i386_ONLY)file path=usr/include/$(ARCH64)/sys/privregs.h
196 file path=usr/include/ads/dsgetdc.h
197 file path=usr/include/aio.h
198 file path=usr/include/alloca.h
199 file path=usr/include/apprtrace.h
200 file path=usr/include/apprtrace_impl.h
201 file path=usr/include/ar.h
202 file path=usr/include/archives.h
203 file path=usr/include/arpa/ftp.h
204 file path=usr/include/arpa/inet.h
205 file path=usr/include/arpa/nameser.h
206 file path=usr/include/arpa/nameser_compat.h
207 file path=usr/include/arpa/telnet.h
208 file path=usr/include/arpa/tftp.h
209 $(i386_ONLY)file path=usr/include/asm/atomic.h
210 $(i386_ONLY)file path=usr/include/asm/bitmap.h
211 $(i386_ONLY)file path=usr/include/asm/byteorder.h
212 $(i386_ONLY)file path=usr/include/asm/clock.h
213 $(i386_ONLY)file path=usr/include/asm/cpu.h
214 $(i386_ONLY)file path=usr/include/asm/cpuvar.h
215 $(sparc_ONLY)file path=usr/include/asm/flush.h
216 $(i386_ONLY)file path=usr/include/asm/htable.h
217 $(i386_ONLY)file path=usr/include/asm/mmu.h
218 file path=usr/include/asm/sunddi.h
219 file path=usr/include/asm/thread.h
220 file path=usr/include/assert.h
221 file path=usr/include/ast/align.h
222 file path=usr/include/ast/ast.h
223 file path=usr/include/ast/ast_botch.h
224 file path=usr/include/ast/ast_ccode.h
225 file path=usr/include/ast/ast_common.h
226 file path=usr/include/ast/ast_dir.h
227 file path=usr/include/ast/ast_dirent.h
228 file path=usr/include/ast/ast_fcntl.h
229 file path=usr/include/ast/ast_float.h
230 file path=usr/include/ast/ast_fs.h
231 file path=usr/include/ast/ast_getopt.h
232 file path=usr/include/ast/ast_iconv.h
233 file path=usr/include/ast/ast_lib.h
234 file path=usr/include/ast/ast_limits.h
235 file path=usr/include/ast/ast_map.h
236 file path=usr/include/ast/ast_mmap.h
237 file path=usr/include/ast/ast_mode.h
238 file path=usr/include/ast/ast_namval.h
239 file path=usr/include/ast/ast_ndbm.h
240 file path=usr/include/ast/ast_nl_types.h
241 file path=usr/include/ast/ast_param.h
242 file path=usr/include/ast/ast_standards.h
243 file path=usr/include/ast/ast_std.h
244 file path=usr/include/ast/ast_stdio.h
245 file path=usr/include/ast/ast_sys.h
246 file path=usr/include/ast/ast_time.h
247 file path=usr/include/ast/ast_tty.h
248 file path=usr/include/ast/ast_version.h
249 file path=usr/include/ast/ast_vfork.h
250 file path=usr/include/ast/ast_wait.h
251 file path=usr/include/ast/ast_wchar.h
252 file path=usr/include/ast/ast_windows.h
253 file path=usr/include/ast/bytsex.h
254 file path=usr/include/ast/ccode.h
255 file path=usr/include/ast/cdt.h
256 file path=usr/include/ast/cmd.h

```

257 file path=usr/include/ast/cmdext.h
258 file path=usr/include/ast/debug.h
259 file path=usr/include/ast/dirent.h
260 file path=usr/include/ast/dlldefs.h
261 file path=usr/include/ast/dt.h
262 file path=usr/include/ast/endian.h
263 file path=usr/include/ast/error.h
264 file path=usr/include/ast/find.h
265 file path=usr/include/ast/fnmatch.h
266 file path=usr/include/ast/fnv.h
267 file path=usr/include/ast/fs3d.h
268 file path=usr/include/ast/fts.h
269 file path=usr/include/ast/ftw.h
270 file path=usr/include/ast/ftwalk.h
271 file path=usr/include/ast/getopt.h
272 file path=usr/include/ast/glob.h
273 file path=usr/include/ast/hash.h
274 file path=usr/include/ast/hashkey.h
275 file path=usr/include/ast/hashpart.h
276 file path=usr/include/ast/history.h
277 file path=usr/include/ast/iconv.h
278 file path=usr/include/ast/ip6.h
279 file path=usr/include/ast/lc.h
280 file path=usr/include/ast/lsh.h
281 file path=usr/include/ast/magic.h
282 file path=usr/include/ast/magicid.h
283 file path=usr/include/ast/mc.h
284 file path=usr/include/ast/mime.h
285 file path=usr/include/ast/mnt.h
286 file path=usr/include/ast/modecanon.h
287 file path=usr/include/ast/modex.h
288 file path=usr/include/ast/namval.h
289 file path=usr/include/ast/nl_types.h
290 file path=usr/include/ast/nval.h
291 file path=usr/include/ast/option.h
292 file path=usr/include/ast/preroot.h
293 file path=usr/include/ast/proc.h
294 file path=usr/include/ast/prototyped.h
295 file path=usr/include/ast/re_comp.h
296 file path=usr/include/ast/recfmt.h
297 file path=usr/include/ast/regex.h
298 file path=usr/include/ast/regexp.h
299 file path=usr/include/ast/sfdisc.h
300 file path=usr/include/ast/sfio.h
301 file path=usr/include/ast/sfio_s.h
302 file path=usr/include/ast/sfio_t.h
303 file path=usr/include/ast/shcmd.h
304 file path=usr/include/ast/shell.h
305 file path=usr/include/ast/sig.h
306 file path=usr/include/ast/stack.h
307 file path=usr/include/ast/stak.h
308 file path=usr/include/ast/stdio.h
309 file path=usr/include/ast/stk.h
310 file path=usr/include/ast/sum.h
311 file path=usr/include/ast/swap.h
312 file path=usr/include/ast/tar.h
313 file path=usr/include/ast/times.h
314 file path=usr/include/ast/tm.h
315 file path=usr/include/ast/tmx.h
316 file path=usr/include/ast/tok.h
317 file path=usr/include/ast/tv.h
318 file path=usr/include/ast/usage.h
319 file path=usr/include/ast/vdb.h
320 file path=usr/include/ast/vecargs.h
321 file path=usr/include/ast/vmalloc.h
322 file path=usr/include/ast/wait.h

323 file path=usr/include/ast/wchar.h
324 file path=usr/include/ast/wordexp.h
325 file path=usr/include/atomic.h
326 file path=usr/include/attr.h
327 file path=usr/include/auth_attr.h
328 file path=usr/include/bsm/adt.h
329 file path=usr/include/bsm/adt_event.h
330 file path=usr/include/bsm/audit.h
331 file path=usr/include/bsm/audit_kernel.h
332 file path=usr/include/bsm/audit_kevents.h
333 file path=usr/include/bsm/audit_record.h
334 file path=usr/include/bsm/audit_uevents.h
335 file path=usr/include/bsm/devices.h
336 file path=usr/include/bsm/libbsm.h
337 file path=usr/include/config_admin.h
338 file path=usr/include/cpio.h
339 file path=usr/include/crypt.h
340 file path=usr/include/cryptoutil.h
341 file path=usr/include/ctype.h
342 file path=usr/include/curses.h
343 file path=usr/include/dat/dat.h
344 file path=usr/include/dat/dat_error.h
345 file path=usr/include/dat/dat_platform_specific.h
346 file path=usr/include/dat/dat_redirection.h
347 file path=usr/include/dat/dat_registry.h
348 file path=usr/include/dat/dat_vendor_specific.h
349 file path=usr/include/dat/udat.h
350 file path=usr/include/dat/udat_config.h
351 file path=usr/include/dat/udat_redirection.h
352 file path=usr/include/dat/udat_vendor_specific.h
353 file path=usr/include/deflt.h
354 file path=usr/include/des/des.h
355 file path=usr/include/des/desdata.h
356 file path=usr/include/des/softdes.h
357 file path=usr/include/device_info.h
358 file path=usr/include/devid.h
359 file path=usr/include/devmgmt.h
360 file path=usr/include/devpoll.h
361 file path=usr/include/dial.h
362 file path=usr/include/dirent.h
363 file path=usr/include/dlfcn.h
364 file path=usr/include/door.h
365 file path=usr/include/elf.h
366 file path=usr/include/endian.h
367 file path=usr/include/err.h
368 file path=usr/include/errno.h
369 file path=usr/include/eti.h
370 file path=usr/include/euc.h
371 file path=usr/include/exacct.h
372 file path=usr/include/exacct_impl.h
373 file path=usr/include/exec_attr.h
374 file path=usr/include/execinfo.h
375 file path=usr/include/fatal.h
376 file path=usr/include/fcntl.h
377 file path=usr/include/float.h
378 file path=usr/include/fmtmsg.h
379 file path=usr/include/fnmatch.h
380 file path=usr/include/form.h
381 file path=usr/include/ftw.h
382 file path=usr/include/gelf.h
383 file path=usr/include/getopt.h
384 file path=usr/include/getwidth.h
385 file path=usr/include/glob.h
386 file path=usr/include/grp.h
387 file path=usr/include/gssapi/gssapi.h
388 file path=usr/include/gssapi/gssapi_ext.h

```

389 file path=usr/include/hal/libhal-storage.h
390 file path=usr/include/hal/libhal.h
391 $(i386_ONLY)file path=usr/include/ia32/sys/asm_linkage.h
392 $(i386_ONLY)file path=usr/include/ia32/sys/kdi_regs.h
393 $(i386_ONLY)file path=usr/include/ia32/sys/machtypes.h
394 $(i386_ONLY)file path=usr/include/ia32/sys/privregs.h
395 $(i386_ONLY)file path=usr/include/ia32/sys/privregs.h
396 $(i386_ONLY)file path=usr/include/ia32/sys/psw.h
397 $(i386_ONLY)file path=usr/include/ia32/sys/pte.h
398 $(i386_ONLY)file path=usr/include/ia32/sys/reg.h
399 $(i386_ONLY)file path=usr/include/ia32/sys/stack.h
400 $(i386_ONLY)file path=usr/include/ia32/sys/trap.h
401 $(i386_ONLY)file path=usr/include/ia32/sys/traptrace.h
402 file path=usr/include/iconv.h
403 file path=usr/include/idmap.h
404 file path=usr/include/ieeefp.h
405 file path=usr/include/ifaddrs.h
406 file path=usr/include/inet/arp.h
407 file path=usr/include/inet/common.h
408 file path=usr/include/inet/ip.h
409 file path=usr/include/inet/ip6.h
410 file path=usr/include/inet/ip6_asp.h
411 file path=usr/include/inet/ip_arp.h
412 file path=usr/include/inet/ip_ftable.h
413 file path=usr/include/inet/ip_if.h
414 file path=usr/include/inet/ip_ire.h
415 file path=usr/include/inet/ip_multi.h
416 file path=usr/include/inet/ip_netinfo.h
417 file path=usr/include/inet/ip_rts.h
418 file path=usr/include/inet/ip_stack.h
419 file path=usr/include/inet/ipclassifier.h
420 file path=usr/include/inet/ipdrop.h
421 file path=usr/include/inet/ipnet.h
422 file path=usr/include/inet/ipp_common.h
423 file path=usr/include/inet/kssl/ksslapi.h
424 file path=usr/include/inet/led.h
425 file path=usr/include/inet/mi.h
426 file path=usr/include/inet/mib2.h
427 file path=usr/include/inet/nd.h
428 file path=usr/include/inet/optcom.h
429 file path=usr/include/inet/sctp_itf.h
430 file path=usr/include/inet/snmpcom.h
431 file path=usr/include/inet/tcp.h
432 file path=usr/include/inet/tcp_sack.h
433 file path=usr/include/inet/tcp_stack.h
434 file path=usr/include/inet/tcp_stats.h
435 file path=usr/include/inet/tunables.h
436 file path=usr/include/inet/wifi_ioctl.h
437 file path=usr/include/inttypes.h
438 file path=usr/include/ipmp.h
439 file path=usr/include/ipmp_admin.h
440 file path=usr/include/ipmp_mpathd.h
441 file path=usr/include/ipmp_query.h
442 file path=usr/include/ipp/ipgpc/ipgpc.h
443 file path=usr/include/ipp/ipp.h
444 file path=usr/include/ipp/ipp_config.h
445 file path=usr/include/ipp/ipp_impl.h
446 file path=usr/include/ipp/ippctl.h
447 file path=usr/include/iso/ctype_iso.h
448 file path=usr/include/iso/limits_iso.h
449 file path=usr/include/iso/locale_iso.h
450 file path=usr/include/iso/setjmp_iso.h
451 file path=usr/include/iso/signal_iso.h
452 file path=usr/include/iso/stdarg_c99.h
453 file path=usr/include/iso/stdarg_iso.h
454 file path=usr/include/iso/stddef_iso.h

```

```

455 file path=usr/include/iso/stdio_c99.h
456 file path=usr/include/iso/stdio_iso.h
457 file path=usr/include/iso/stdlib_c11.h
458 file path=usr/include/iso/stdlib_c99.h
459 file path=usr/include/iso/stdlib_iso.h
460 file path=usr/include/iso/string_iso.h
461 file path=usr/include/iso/time_iso.h
462 file path=usr/include/iso/wchar_c99.h
463 file path=usr/include/iso/wchar_iso.h
464 file path=usr/include/iso/wctype_iso.h
465 file path=usr/include/iso646.h
466 file path=usr/include/kerberosv5/com_err.h
467 file path=usr/include/kerberosv5/krb5.h
468 file path=usr/include/kerberosv5/locate_plugin.h
469 file path=usr/include/kerberosv5/mit-sipb-copyright.h
470 file path=usr/include/kerberosv5/mit_copyright.h
471 file path=usr/include/klpd.h
472 file path=usr/include/kmfapi.h
473 file path=usr/include/kmftypes.h
474 file path=usr/include/kstat.h
475 file path=usr/include/kvm.h
476 file path=usr/include/langinfo.h
477 file path=usr/include/lastlog.h
478 file path=usr/include/lber.h
479 file path=usr/include/ldap.h
480 file path=usr/include/libcontract.h
481 file path=usr/include/libctf.h
482 file path=usr/include/libdevice.h
483 file path=usr/include/libdevinfo.h
484 file path=usr/include/libdladm.h
485 file path=usr/include/libdlbridge.h
486 file path=usr/include/libdlib.h
487 file path=usr/include/libdllink.h
488 file path=usr/include/libdmpi.h
489 file path=usr/include/libdvlvlan.h
490 file path=usr/include/libelf.h
491 $(i386_ONLY)file path=usr/include/libfdisk.h
492 file path=usr/include/libfstyp.h
493 file path=usr/include/libfstyp_module.h
494 file path=usr/include/libgen.h
495 file path=usr/include/libgrubmgmt.h
496 file path=usr/include/libintl.h
497 file path=usr/include/libipmi.h
498 file path=usr/include/libipp.h
499 file path=usr/include/libnvpair.h
500 file path=usr/include/libnwam.h
501 file path=usr/include/libpolkit/libpolkit.h
502 file path=usr/include/libproc.h
503 file path=usr/include/librcm.h
504 file path=usr/include/libscf.h
505 file path=usr/include/libscf_priv.h
506 file path=usr/include/libshare.h
507 file path=usr/include/libsvm.h
508 file path=usr/include/libsysevent.h
509 file path=usr/include/libsysevent_impl.h
510 file path=usr/include/libtsnet.h
511 $(sparc_ONLY)file path=usr/include/libv12n.h
512 file path=usr/include/libw.h
513 file path=usr/include/libzfs.h
514 file path=usr/include/libzfs_core.h
515 file path=usr/include/libzoneinfo.h
516 file path=usr/include/limits.h
517 file path=usr/include/linenum.h
518 file path=usr/include/link.h
519 file path=usr/include/listen.h
520 file path=usr/include/locale.h

```

```

521 file path=usr/include/macros.h
522 file path=usr/include/maillock.h
523 file path=usr/include/malloc.h
524 file path=usr/include/md4.h
525 file path=usr/include/md5.h
526 file path=usr/include/mdiox.h
527 file path=usr/include/mdmm_changelog.h
528 file path=usr/include/memory.h
529 file path=usr/include/menu.h
530 file path=usr/include/meta.h
531 file path=usr/include/meta_basic.h
532 file path=usr/include/meta_runtime.h
533 file path=usr/include/metacl.h
534 file path=usr/include/metad.h
535 file path=usr/include/metadyn.h
536 file path=usr/include/metamed.h
537 file path=usr/include/metamhd.h
538 file path=usr/include/mhdx.h
539 file path=usr/include/mon.h
540 file path=usr/include/monetary.h
541 file path=usr/include/mp.h
542 file path=usr/include/mqueue.h
543 file path=usr/include/mtmalloc.h
544 file path=usr/include/nan.h
545 file path=usr/include/ndbm.h
546 file path=usr/include/ndpd.h
547 file path=usr/include/net/af.h
548 file path=usr/include/net/bridge.h
549 file path=usr/include/net/if.h
550 file path=usr/include/net/if_arp.h
551 file path=usr/include/net/if_dl.h
552 file path=usr/include/net/if_types.h
553 file path=usr/include/net/pfkeyv2.h
554 file path=usr/include/net/pfpolicy.h
555 file path=usr/include/net/ppp-comp.h
556 file path=usr/include/net/ppp_defs.h
557 file path=usr/include/net/pppio.h
558 file path=usr/include/net/radix.h
559 file path=usr/include/net/route.h
560 file path=usr/include/net/trill.h
561 file path=usr/include/net/vjcompress.h
562 file path=usr/include/netconfig.h
563 file path=usr/include/netdb.h
564 file path=usr/include/netdir.h
565 file path=usr/include/netinet/arp.h
566 file path=usr/include/netinet/dhcp.h
567 file path=usr/include/netinet/dhcp6.h
568 file path=usr/include/netinet/icmp6.h
569 file path=usr/include/netinet/icmp_var.h
570 file path=usr/include/netinet/if_ether.h
571 file path=usr/include/netinet/igmp.h
572 file path=usr/include/netinet/igmp_var.h
573 file path=usr/include/netinet/in.h
574 file path=usr/include/netinet/in_pcb.h
575 file path=usr/include/netinet/in_system.h
576 file path=usr/include/netinet/in_var.h
577 file path=usr/include/netinet/ip.h
578 file path=usr/include/netinet/ip6.h
579 file path=usr/include/netinet/ip_icmp.h
580 file path=usr/include/netinet/ip_mroute.h
581 file path=usr/include/netinet/ip_var.h
582 file path=usr/include/netinet/pim.h
583 file path=usr/include/netinet/sctp.h
584 file path=usr/include/netinet/tcp.h
585 file path=usr/include/netinet/tcp_debug.h
586 file path=usr/include/netinet/tcp_fsm.h

```

```

587 file path=usr/include/netinet/tcp_seq.h
588 file path=usr/include/netinet/tcp_timer.h
589 file path=usr/include/netinet/tcp_var.h
590 file path=usr/include/netinet/tcpip.h
591 file path=usr/include/netinet/udp.h
592 file path=usr/include/netinet/udp_var.h
593 file path=usr/include/netinet/vrrp.h
594 file path=usr/include/nfs/auth.h
595 file path=usr/include/nfs/export.h
596 file path=usr/include/nfs/lm.h
597 file path=usr/include/nfs/mapid.h
598 file path=usr/include/nfs/mount.h
599 file path=usr/include/nfs/nfs.h
600 file path=usr/include/nfs/nfs4.h
601 file path=usr/include/nfs/nfs4_attr.h
602 file path=usr/include/nfs/nfs4_clnt.h
603 file path=usr/include/nfs/nfs4_db_impl.h
604 file path=usr/include/nfs/nfs4_idmap_impl.h
605 file path=usr/include/nfs/nfs4_kprot.h
606 file path=usr/include/nfs/nfs_acl.h
607 file path=usr/include/nfs/nfs_clnt.h
608 file path=usr/include/nfs/nfs_cmd.h
609 file path=usr/include/nfs/nfs_log.h
610 file path=usr/include/nfs/nfs_sec.h
611 file path=usr/include/nfs/nfsid_map.h
612 file path=usr/include/nfs/nfssys.h
613 file path=usr/include/nfs/rnode.h
614 file path=usr/include/nfs/rnode4.h
615 file path=usr/include/nl_types.h
616 file path=usr/include/nlist.h
617 file path=usr/include/note.h
618 file path=usr/include/nss_common.h
619 file path=usr/include/nss_dbdefs.h
620 file path=usr/include/nss_netdir.h
621 file path=usr/include/nsswitch.h
622 file path=usr/include/panel.h
623 file path=usr/include/paths.h
624 file path=usr/include/pcsample.h
625 file path=usr/include/pfmt.h
626 file path=usr/include/pkgdev.h
627 file path=usr/include/pkginfo.h
628 file path=usr/include/pkglocs.h
629 file path=usr/include/pkgstrct.h
630 file path=usr/include/pkgtrans.h
631 file path=usr/include/poll.h
632 file path=usr/include/port.h
633 file path=usr/include/priv.h
634 file path=usr/include/proc_service.h
635 file path=usr/include/procfs.h
636 file path=usr/include/prof.h
637 file path=usr/include/prof_attr.h
638 file path=usr/include/project.h
639 file path=usr/include/protocols/dumppstore.h
640 file path=usr/include/protocols/routed.h
641 file path=usr/include/protocols/rwhod.h
642 file path=usr/include/protocols/timed.h
643 file path=usr/include/pthread.h
644 file path=usr/include/pw.h
645 file path=usr/include/pwd.h
646 file path=usr/include/rcm_module.h
647 file path=usr/include/rctl.h
648 file path=usr/include/re_comp.h
649 file path=usr/include/regex.h
650 file path=usr/include/regex.h
651 file path=usr/include/regexr.h
652 file path=usr/include/resolv.h

```



```

653 file path=usr/include/rje.h
654 file path=usr/include/rp_plugin.h
655 file path=usr/include/rpc/auth.h
656 file path=usr/include/rpc/auth_des.h
657 file path=usr/include/rpc/auth_sys.h
658 file path=usr/include/rpc/auth_unix.h
659 file path=usr/include/rpc/bootparam.h
660 file path=usr/include/rpc/clnt.h
661 file path=usr/include/rpc/clnt_soc.h
662 file path=usr/include/rpc/clnt_stat.h
663 file path=usr/include/rpc/des_crypt.h
664 $(sparc_ONLY)file path=usr/include/rpc/ib.h
665 file path=usr/include/rpc/key_prot.h
666 file path=usr/include/rpc/nettype.h
667 file path=usr/include/rpc/pmap_clnt.h
668 file path=usr/include/rpc/pmap_prot.h
669 file path=usr/include/rpc/pmap_prot.x
670 file path=usr/include/rpc/pmap_rmt.h
671 file path=usr/include/rpc/raw.h
672 file path=usr/include/rpc/rpc.h
673 file path=usr/include/rpc/rpc_com.h
674 file path=usr/include/rpc/rpc_msg.h
675 file path=usr/include/rpc/rpc_rdma.h
676 file path=usr/include/rpc/rpc_sztypes.h
677 file path=usr/include/rpc/rpcb_clnt.h
678 file path=usr/include/rpc/rpcb_prot.h
679 file path=usr/include/rpc/rpcb_prot.x
680 file path=usr/include/rpc/rpcent.h
681 file path=usr/include/rpc/rpcsec_gss.h
682 file path=usr/include/rpc/rpcsys.h
683 file path=usr/include/rpc/svc.h
684 file path=usr/include/rpc/svc_auth.h
685 file path=usr/include/rpc/svc_mt.h
686 file path=usr/include/rpc/svc_soc.h
687 file path=usr/include/rpc/types.h
688 file path=usr/include/rpc/xdr.h
689 file path=usr/include/rpcsvc/autofs_prot.h
690 file path=usr/include/rpcsvc/autofs_prot.x
691 file path=usr/include/rpcsvc/bootparam.h
692 file path=usr/include/rpcsvc/bootparam_prot.h
693 file path=usr/include/rpcsvc/bootparam_prot.x
694 file path=usr/include/rpcsvc/dbm.h
695 file path=usr/include/rpcsvc/key_prot.x
696 file path=usr/include/rpcsvc/mount.h
697 file path=usr/include/rpcsvc/mount.x
698 file path=usr/include/rpcsvc/nfs4_prot.h
699 file path=usr/include/rpcsvc/nfs4_prot.x
700 file path=usr/include/rpcsvc/nfs_acl.h
701 file path=usr/include/rpcsvc/nfs_acl.x
702 file path=usr/include/rpcsvc/nfs_prot.h
703 file path=usr/include/rpcsvc/nfs_prot.x
704 file path=usr/include/rpcsvc/nis.h
705 file path=usr/include/rpcsvc/nis.x
706 file path=usr/include/rpcsvc/nis_db.h
707 file path=usr/include/rpcsvc/nis_object.x
708 file path=usr/include/rpcsvc/nislib.h
709 file path=usr/include/rpcsvc/nlm_prot.h
710 file path=usr/include/rpcsvc/nlm_prot.x
711 file path=usr/include/rpcsvc/nsm_addr.h
712 file path=usr/include/rpcsvc/nsm_addr.x
713 file path=usr/include/rpcsvc/rex.h
714 file path=usr/include/rpcsvc/rex.x
715 file path=usr/include/rpcsvc/rpc_sztypes.h
716 file path=usr/include/rpcsvc/rpc_sztypes.x
717 file path=usr/include/rpcsvc/rquota.h
718 file path=usr/include/rpcsvc/rquota.x

```

```

719 file path=usr/include/rpcsvc/rstat.h
720 file path=usr/include/rpcsvc/rstat.x
721 file path=usr/include/rpcsvc/rusers.h
722 file path=usr/include/rpcsvc/rusers.x
723 file path=usr/include/rpcsvc/rwall.h
724 file path=usr/include/rpcsvc/rwall.x
725 file path=usr/include/rpcsvc/sm_inter.h
726 file path=usr/include/rpcsvc/sm_inter.x
727 file path=usr/include/rpcsvc/spray.h
728 file path=usr/include/rpcsvc/spray.x
729 file path=usr/include/rpcsvc/ufs_prot.h
730 file path=usr/include/rpcsvc/ufs_prot.x
731 file path=usr/include/rpcsvc/yp.x
732 file path=usr/include/rpcsvc/yp_prot.h
733 file path=usr/include/rpcsvc/ypclnt.h
734 file path=usr/include/rpcsvc/yppasswd.h
735 file path=usr/include/rpcsvc/ypupd.h
736 file path=usr/include/rsmapi.h
737 file path=usr/include/rtdb_db.h
738 file path=usr/include/sac.h
739 file path=usr/include/sasl/prop.h
740 file path=usr/include/sasl/sasl.h
741 file path=usr/include/sasl/saslplug.h
742 file path=usr/include/sasl/saslutil.h
743 file path=usr/include/sched.h
744 file path=usr/include/schedctl.h
745 file path=usr/include/scsi/libscsi.h
746 file path=usr/include/scsi/libses.h
747 file path=usr/include/scsi/libses_plugin.h
748 file path=usr/include/scsi/libsmpl.h
749 file path=usr/include/scsi/libsmpl_plugin.h
750 file path=usr/include/scsi/plugins/ses/framework/libses.h
751 file path=usr/include/scsi/plugins/ses/framework/ses2.h
752 file path=usr/include/scsi/plugins/ses/framework/ses2_impl.h
753 file path=usr/include/scsi/plugins/ses/vendor/sun.h
754 file path=usr/include/sdp.h
755 file path=usr/include/search.h
756 file path=usr/include/secdb.h
757 file path=usr/include/security/auditd.h
758 file path=usr/include/security/cryptoki.h
759 file path=usr/include/security/pam_appl.h
760 file path=usr/include/security/pam_modules.h
761 file path=usr/include/security/pkcs11.h
762 file path=usr/include/security/pkcs11f.h
763 file path=usr/include/security/pkcs11t.h
764 file path=usr/include/semaphore.h
765 file path=usr/include/setjmp.h
766 file path=usr/include/sgtty.h
767 file path=usr/include/shal.h
768 file path=usr/include/sha2.h
769 file path=usr/include/shadow.h
770 file path=usr/include/sharefs/share.h
771 file path=usr/include/sharefs/sharefs.h
772 file path=usr/include/sharefs/sharetab.h
773 file path=usr/include/siginfo.h
774 file path=usr/include/signal.h
775 file path=usr/include/sip.h
776 file path=usr/include/skein.h
777 file path=usr/include/smbios.h
778 file path=usr/include/spawn.h
779 $(i386_ONLY)file path=usr/include/stack_unwind.h
780 file path=usr/include/stdalign.h
781 file path=usr/include/stdarg.h
782 file path=usr/include/stdbool.h
783 file path=usr/include/stddef.h
784 file path=usr/include/stdint.h

```

```

785 file path=usr/include/stdio.h
786 file path=usr/include/stdio_ext.h
787 file path=usr/include/stdio_impl.h
788 file path=usr/include/stdio_tag.h
789 file path=usr/include/stdlib.h
790 file path=usr/include/stdnoreturn.h
791 file path=usr/include/storclass.h
792 file path=usr/include/string.h
793 file path=usr/include/strings.h
794 file path=usr/include/stropts.h
795 file path=usr/include/syms.h
796 file path=usr/include/synch.h
797 file path=usr/include/sys/acct.h
798 file path=usr/include/sys/acctctl.h
799 file path=usr/include/sys/acl.h
800 file path=usr/include/sys/acl_impl.h
801 file path=usr/include/sys/acpi_drv.h
802 file path=usr/include/sys/aio.h
803 file path=usr/include/sys/aio_impl.h
804 file path=usr/include/sys/aio_req.h
805 file path=usr/include/sys/aioch.h
806 file path=usr/include/sys/archsystem.h
807 file path=usr/include/sys/ascii.h
808 file path=usr/include/sys/asm_linkage.h
809 file path=usr/include/sys/asynch.h
810 file path=usr/include/sys/atomic.h
811 file path=usr/include/sys/attr.h
812 file path=usr/include/sys/autoconf.h
813 file path=usr/include/sys/auxv.h
814 file path=usr/include/sys/auxv_386.h
815 file path=usr/include/sys/auxv_SPARC.h
816 file path=usr/include/sys/av/iec61883.h
817 file path=usr/include/sys/avintr.h
818 file path=usr/include/sys/avl.h
819 file path=usr/include/sys/avl_impl.h
820 file path=usr/include/sys/bitmap.h
821 file path=usr/include/sys/bitset.h
822 file path=usr/include/sys/bl.h
823 file path=usr/include/sys/blkdev.h
824 file path=usr/include/sys/bofi.h
825 file path=usr/include/sys/bofi_impl.h
826 file path=usr/include/sys/bootconf.h
827 $(i386_ONLY)file path=usr/include/sys/bootregs.h
828 file path=usr/include/sys/bootstat.h
829 $(i386_ONLY)file path=usr/include/sys/bootsvcs.h
830 file path=usr/include/sys/bpp_io.h
831 file path=usr/include/sys/brand.h
832 file path=usr/include/sys/buf.h
833 file path=usr/include/sys/bufmod.h
834 file path=usr/include/sys/bustypes.h
835 file path=usr/include/sys/byteorder.h
836 file path=usr/include/sys/callb.h
837 file path=usr/include/sys/callo.h
838 file path=usr/include/sys/cap_util.h
839 file path=usr/include/sys/ccompile.h
840 file path=usr/include/sys/cdio.h
841 file path=usr/include/sys/cis.h
842 file path=usr/include/sys/cis_handlers.h
843 file path=usr/include/sys/cis_protos.h
844 file path=usr/include/sys/cladm.h
845 file path=usr/include/sys/class.h
846 file path=usr/include/sys/clconf.h
847 file path=usr/include/sys/cmlb.h
848 file path=usr/include/sys/cmn_err.h
849 $(sparc_ONLY)file path=usr/include/sys/cmpregs.h
850 file path=usr/include/sys/compress.h

```

```

851 file path=usr/include/sys/condvar.h
852 file path=usr/include/sys/condvar_impl.h
853 file path=usr/include/sys/conf.h
854 file path=usr/include/sys/consdev.h
855 file path=usr/include/sys/console.h
856 file path=usr/include/sys/consplat.h
857 file path=usr/include/sys/contract.h
858 file path=usr/include/sys/contract/device.h
859 file path=usr/include/sys/contract/device_impl.h
860 file path=usr/include/sys/contract/process.h
861 file path=usr/include/sys/contract/process_impl.h
862 file path=usr/include/sys/contract_impl.h
863 $(i386_ONLY)file path=usr/include/sys/controlregs.h
864 file path=usr/include/sys/copyops.h
865 file path=usr/include/sys/core.h
866 file path=usr/include/sys/corectl.h
867 file path=usr/include/sys/cpc_impl.h
868 file path=usr/include/sys/cpc_pcbe.h
869 file path=usr/include/sys/cpr.h
870 file path=usr/include/sys/cpu.h
871 file path=usr/include/sys/cpucaps.h
872 file path=usr/include/sys/cpucaps_impl.h
873 file path=usr/include/sys/cpupart.h
874 file path=usr/include/sys/cpuvar.h
875 file path=usr/include/sys/crc32.h
876 file path=usr/include/sys/cred.h
877 file path=usr/include/sys/cred_impl.h
878 file path=usr/include/sys/crtctl.h
879 file path=usr/include/sys/crypto/api.h
880 file path=usr/include/sys/crypto/common.h
881 file path=usr/include/sys/crypto/ioctl.h
882 file path=usr/include/sys/crypto/ioctladmin.h
883 file path=usr/include/sys/crypto/spi.h
884 file path=usr/include/sys/cs.h
885 file path=usr/include/sys/cs_priv.h
886 file path=usr/include/sys/cs_strings.h
887 file path=usr/include/sys/cs_stubs.h
888 file path=usr/include/sys/cs_types.h
889 file path=usr/include/sys/csiioctl.h
890 file path=usr/include/sys/ctf.h
891 file path=usr/include/sys/ctf_api.h
892 file path=usr/include/sys/ctfs.h
893 file path=usr/include/sys/ctfs_impl.h
894 file path=usr/include/sys/ctype.h
895 file path=usr/include/sys/cyclic.h
896 file path=usr/include/sys/cyclic_impl.h
897 file path=usr/include/sys/dacf.h
898 file path=usr/include/sys/dacf_impl.h
899 file path=usr/include/sys/damap.h
900 file path=usr/include/sys/damap_impl.h
901 file path=usr/include/sys/dc_ki.h
902 file path=usr/include/sys/ddi.h
903 file path=usr/include/sys/ddi_hp.h
904 file path=usr/include/sys/ddi_hp_impl.h
905 file path=usr/include/sys/ddi_impldefs.h
906 file path=usr/include/sys/ddi_implfuncs.h
907 file path=usr/include/sys/ddi_intr.h
908 file path=usr/include/sys/ddi_intr_impl.h
909 file path=usr/include/sys/ddi_isa.h
910 file path=usr/include/sys/ddi_obsolete.h
911 file path=usr/include/sys/ddi_periodic.h
912 file path=usr/include/sys/ddidevmap.h
913 file path=usr/include/sys/ddidmareq.h
914 file path=usr/include/sys/ddifm.h
915 file path=usr/include/sys/ddifm_impl.h
916 file path=usr/include/sys/ddimapreq.h

```

917 file path=usr/include/sys/ddipropdefs.h
 918 file path=usr/include/sys/dditypes.h
 919 file path=usr/include/sys/debug.h
 920 \$(i386_ONLY)file path=usr/include/sys/debugreg.h
 921 file path=usr/include/sys/des.h
 922 file path=usr/include/sys/devcache.h
 923 file path=usr/include/sys/devcache_impl.h
 924 file path=usr/include/sys/devctl.h
 925 file path=usr/include/sys/devfm.h
 926 file path=usr/include/sys/devid_cache.h
 927 file path=usr/include/sys/devinfo_impl.h
 928 file path=usr/include/sys/devops.h
 929 file path=usr/include/sys/devpolicy.h
 930 file path=usr/include/sys/devpoll.h
 931 file path=usr/include/sys/dirent.h
 932 file path=usr/include/sys/disp.h
 933 file path=usr/include/sys/dkbad.h
 934 file path=usr/include/sys/dkio.h
 935 file path=usr/include/sys/dklabel.h
 936 \$(sparc_ONLY)file path=usr/include/sys/dkmpio.h
 937 \$(i386_ONLY)file path=usr/include/sys/dktp/altctr.h
 938 \$(i386_ONLY)file path=usr/include/sys/dktp/cmpkt.h
 939 file path=usr/include/sys/dktp/dadkio.h
 940 file path=usr/include/sys/dktp/fdisk.h
 941 file path=usr/include/sys/dl.h
 942 file path=usr/include/sys/dld.h
 943 file path=usr/include/sys/dlpi.h
 944 file path=usr/include/sys/dls_mgmt.h
 945 \$(i386_ONLY)file path=usr/include/sys/dma_engine.h
 946 file path=usr/include/sys/dma_i8237A.h
 947 file path=usr/include/sys/dnlc.h
 948 file path=usr/include/sys/door.h
 949 file path=usr/include/sys/door_data.h
 950 file path=usr/include/sys/door_impl.h
 951 file path=usr/include/sys/dumphdr.h
 952 file path=usr/include/sys/ecppio.h
 953 file path=usr/include/sys/ecppreg.h
 954 file path=usr/include/sys/ecpps.h
 955 file path=usr/include/sys/ecppvar.h
 956 file path=usr/include/sys/edonr.h
 957 file path=usr/include/sys/efi_partition.h
 958 file path=usr/include/sys/elf.h
 959 file path=usr/include/sys/elf_386.h
 960 file path=usr/include/sys/elf_SPARC.h
 961 file path=usr/include/sys/elf_amd64.h
 962 file path=usr/include/sys/elf_notes.h
 963 file path=usr/include/sys/elftypes.h
 964 file path=usr/include/sys/epm.h
 965 file path=usr/include/sys/epoll.h
 966 file path=usr/include/sys/errno.h
 967 file path=usr/include/sys/errorq.h
 968 file path=usr/include/sys/errorq_impl.h
 969 file path=usr/include/sys/esunddi.h
 970 file path=usr/include/sys/ethernet.h
 971 file path=usr/include/sys/euc.h
 972 file path=usr/include/sys/eucioctl.h
 973 file path=usr/include/sys/eventfd.h
 974 file path=usr/include/sys/exacct.h
 975 file path=usr/include/sys/exacct_catalog.h
 976 file path=usr/include/sys/exacct_impl.h
 977 file path=usr/include/sys/exec.h
 978 file path=usr/include/sys/exechdr.h
 979 file path=usr/include/sys/fault.h
 980 file path=usr/include/sys/fbio.h
 981 file path=usr/include/sys/fbuf.h
 982 file path=usr/include/sys/fc4/fc.h

983 file path=usr/include/sys/fc4/fc_transport.h
 984 file path=usr/include/sys/fc4/fcal.h
 985 file path=usr/include/sys/fc4/fcal_linkapp.h
 986 file path=usr/include/sys/fc4/fcal_transport.h
 987 file path=usr/include/sys/fc4/fcio.h
 988 file path=usr/include/sys/fc4/fcp.h
 989 file path=usr/include/sys/fc4/linkapp.h
 990 file path=usr/include/sys/fcntl.h
 991 file path=usr/include/sys/fdbuffer.h
 992 file path=usr/include/sys/fdio.h
 993 \$(sparc_ONLY)file path=usr/include/sys/fdreg.h
 994 \$(sparc_ONLY)file path=usr/include/sys/fdvar.h
 995 file path=usr/include/sys/feature_tests.h
 996 file path=usr/include/sys/fem.h
 997 file path=usr/include/sys/file.h
 998 file path=usr/include/sys/filio.h
 999 file path=usr/include/sys/flock.h
 1000 file path=usr/include/sys/flock_impl.h
 1001 \$(sparc_ONLY)file path=usr/include/sys/fm/cpu/SPARC64-VI.h
 1002 \$(sparc_ONLY)file path=usr/include/sys/fm/cpu/UltraSPARC-II.h
 1003 \$(sparc_ONLY)file path=usr/include/sys/fm/cpu/UltraSPARC-III.h
 1004 \$(sparc_ONLY)file path=usr/include/sys/fm/cpu/UltraSPARC-T1.h
 1005 file path=usr/include/sys/fm/fs/zfs.h
 1006 file path=usr/include/sys/fm/io/ddi.h
 1007 file path=usr/include/sys/fm/io/disk.h
 1008 file path=usr/include/sys/fm/io/opl_mc_fm.h
 1009 file path=usr/include/sys/fm/io/pci.h
 1010 file path=usr/include/sys/fm/io/scsi.h
 1011 file path=usr/include/sys/fm/io/sun4upci.h
 1012 file path=usr/include/sys/fm/protocol.h
 1013 file path=usr/include/sys/fm/util.h
 1014 file path=usr/include/sys/fork.h
 1015 \$(i386_ONLY)file path=usr/include/sys/fp.h
 1016 \$(sparc_ONLY)file path=usr/include/sys/fpu/fpu_simulator.h
 1017 \$(sparc_ONLY)file path=usr/include/sys/fpu/fpusystem.h
 1018 \$(sparc_ONLY)file path=usr/include/sys/fpu/globals.h
 1019 \$(sparc_ONLY)file path=usr/include/sys/fpu/ieee.h
 1020 file path=usr/include/sys/frame.h
 1021 file path=usr/include/sys/fs/autofs.h
 1022 file path=usr/include/sys/fs/decomp.h
 1023 file path=usr/include/sys/fs/dv_node.h
 1024 file path=usr/include/sys/fs/fifonode.h
 1025 file path=usr/include/sys/fs/hsfs_isospec.h
 1026 file path=usr/include/sys/fs/hsfs_node.h
 1027 file path=usr/include/sys/fs/hsfs_rrip.h
 1028 file path=usr/include/sys/fs/hsfs_spec.h
 1029 file path=usr/include/sys/fs/hsfs_susp.h
 1030 file path=usr/include/sys/fs/lofs_info.h
 1031 file path=usr/include/sys/fs/lofs_node.h
 1032 file path=usr/include/sys/fs/mntdata.h
 1033 file path=usr/include/sys/fs/namenode.h
 1034 file path=usr/include/sys/fs/pc_dir.h
 1035 file path=usr/include/sys/fs/pc_fs.h
 1036 file path=usr/include/sys/fs/pc_label.h
 1037 file path=usr/include/sys/fs/pc_node.h
 1038 file path=usr/include/sys/fs/pxfs_ki.h
 1039 file path=usr/include/sys/fs/sdev_impl.h
 1040 file path=usr/include/sys/fs/snode.h
 1041 file path=usr/include/sys/fs/swapnode.h
 1042 file path=usr/include/sys/fs/tmp.h
 1043 file path=usr/include/sys/fs/tmpnode.h
 1044 file path=usr/include/sys/fs/udf_inode.h
 1045 file path=usr/include/sys/fs/udf_volume.h
 1046 file path=usr/include/sys/fs/ufs_acl.h
 1047 file path=usr/include/sys/fs/ufs_bio.h
 1048 file path=usr/include/sys/fs/ufs_filio.h

1049 file path=usr/include/sys/fs/ufs_fs.h
 1050 file path=usr/include/sys/fs/ufs_fsdir.h
 1051 file path=usr/include/sys/fs/ufs_inode.h
 1052 file path=usr/include/sys/fs/ufs_lockfs.h
 1053 file path=usr/include/sys/fs/ufs_log.h
 1054 file path=usr/include/sys/fs/ufs_mount.h
 1055 file path=usr/include/sys/fs/ufs_panic.h
 1056 file path=usr/include/sys/fs/ufs_prot.h
 1057 file path=usr/include/sys/fs/ufs_quota.h
 1058 file path=usr/include/sys/fs/ufs_snap.h
 1059 file path=usr/include/sys/fs/ufs_trans.h
 1060 file path=usr/include/sys/fs/zfs.h
 1061 file path=usr/include/sys/fs_reparse.h
 1062 file path=usr/include/sys/fs_subr.h
 1063 file path=usr/include/sys/fsid.h
 1064 \$(sparc_ONLY)file path=usr/include/sys/fsr.h
 1065 file path=usr/include/sys/fss.h
 1066 file path=usr/include/sys/fssnap.h
 1067 file path=usr/include/sys/fssnap_if.h
 1068 file path=usr/include/sys/fssprioctl.h
 1069 file path=usr/include/sys/fstyp.h
 1070 file path=usr/include/sys/fttrace.h
 1071 file path=usr/include/sys/fx.h
 1072 file path=usr/include/sys/fxpriocntl.h
 1073 file path=usr/include/sys/gfs.h
 1074 file path=usr/include/sys/gld.h
 1075 file path=usr/include/sys/gldpriv.h
 1076 file path=usr/include/sys/group.h
 1077 file path=usr/include/sys/hdio.h
 1078 file path=usr/include/sys/hook.h
 1079 file path=usr/include/sys/hook_event.h
 1080 file path=usr/include/sys/hook_impl.h
 1081 file path=usr/include/sys/hotplug/hpcsvc.h
 1082 file path=usr/include/sys/hotplug/hpctrl.h
 1083 file path=usr/include/sys/hotplug/pci/pcicfg.h
 1084 file path=usr/include/sys/hotplug/pci/pcihp.h
 1085 file path=usr/include/sys/hwconf.h
 1086 \$(i386_ONLY)file path=usr/include/sys/hypervisor.h
 1087 \$(i386_ONLY)file path=usr/include/sys/i8272A.h
 1088 file path=usr/include/sys/ia.h
 1089 file path=usr/include/sys/iapriocntl.h
 1090 file path=usr/include/sys/ib/adapters/hermon/hermon_ioctl.h
 1091 file path=usr/include/sys/ib/adapters/mlnx_umap.h
 1092 file path=usr/include/sys/ib/adapters/tavor/tavor_ioctl.h
 1093 file path=usr/include/sys/ib/clients/ibd/ibd.h
 1094 file path=usr/include/sys/ib/clients/of/ofa_solaris.h
 1095 file path=usr/include/sys/ib/clients/of/ofed_kernel.h
 1096 file path=usr/include/sys/ib/clients/of/rdma/ib_addr.h
 1097 file path=usr/include/sys/ib/clients/of/rdma/ib_user_mad.h
 1098 file path=usr/include/sys/ib/clients/of/rdma/ib_user_sa.h
 1099 file path=usr/include/sys/ib/clients/of/rdma/ib_user_verbs.h
 1100 file path=usr/include/sys/ib/clients/of/rdma/ib_verbs.h
 1101 file path=usr/include/sys/ib/clients/of/rdma/rdma_cm.h
 1102 file path=usr/include/sys/ib/clients/of/rdma/rdma_user_cm.h
 1103 file path=usr/include/sys/ib/clients/of/sol_ofs/sol_cma.h
 1104 file path=usr/include/sys/ib/clients/of/sol_ofs/sol_ib_cma.h
 1105 file path=usr/include/sys/ib/clients/of/sol_ofs/sol_kverb_impl.h
 1106 file path=usr/include/sys/ib/clients/of/sol_ofs/sol_ofs_common.h
 1107 file path=usr/include/sys/ib/clients/of/sol_ucma/sol_rdma_user_cm.h
 1108 file path=usr/include/sys/ib/clients/of/sol_ucma/sol_ucma.h
 1109 file path=usr/include/sys/ib/clients/of/sol_umad/sol_umad.h
 1110 file path=usr/include/sys/ib/clients/of/sol_uverbs/sol_uverbs.h
 1111 file path=usr/include/sys/ib/clients/of/sol_uverbs/sol_uverbs2ucma.h
 1112 file path=usr/include/sys/ib/clients/of/sol_uverbs/sol_uverbs_comp.h
 1113 file path=usr/include/sys/ib/clients/of/sol_uverbs/sol_uverbs_event.h
 1114 file path=usr/include/sys/ib/clients/of/sol_uverbs/sol_uverbs_hca.h

1115 file path=usr/include/sys/ib/clients/of/sol_uverbs/sol_uverbs_qp.h
 1116 file path=usr/include/sys/ib/ib_pkt_hdrs.h
 1117 file path=usr/include/sys/ib/ib_types.h
 1118 file path=usr/include/sys/ib/ibnex/ibnex_devctl.h
 1119 file path=usr/include/sys/ib/ibtl/ibci.h
 1120 file path=usr/include/sys/ib/ibtl/ibti.h
 1121 file path=usr/include/sys/ib/ibtl/ibti_cm.h
 1122 file path=usr/include/sys/ib/ibtl/ibti_common.h
 1123 file path=usr/include/sys/ib/ibtl/ibtl_ci_types.h
 1124 file path=usr/include/sys/ib/ibtl/ibtl_status.h
 1125 file path=usr/include/sys/ib/ibtl/ibtl_types.h
 1126 file path=usr/include/sys/ib/ibtl/ibvti.h
 1127 file path=usr/include/sys/ib/ibtl/impl/ibtl_util.h
 1128 file path=usr/include/sys/ib/mgt/ib_dm_attr.h
 1129 file path=usr/include/sys/ib/mgt/ib_mad.h
 1130 file path=usr/include/sys/ib/mgt/ibmf/ibmf.h
 1131 file path=usr/include/sys/ib/mgt/ibmf/ibmf_msg.h
 1132 file path=usr/include/sys/ib/mgt/ibmf/ibmf_saa.h
 1133 file path=usr/include/sys/ib/mgt/ibmf/ibmf_utils.h
 1134 file path=usr/include/sys/ib/mgt/sa_recs.h
 1135 file path=usr/include/sys/ib/mgt/sm_attr.h
 1136 file path=usr/include/sys/ibpart.h
 1137 file path=usr/include/sys/id32.h
 1138 file path=usr/include/sys/id_space.h
 1139 file path=usr/include/sys/idmap.h
 1140 file path=usr/include/sys/inline.h
 1141 file path=usr/include/sys/instance.h
 1142 file path=usr/include/sys/int_const.h
 1143 file path=usr/include/sys/int_fmtio.h
 1144 file path=usr/include/sys/int_limits.h
 1145 file path=usr/include/sys/int_types.h
 1146 file path=usr/include/sys/inttypes.h
 1147 file path=usr/include/sys/ioccom.h
 1148 file path=usr/include/sys/ioctl.h
 1149 \$(i386_ONLY)file path=usr/include/sys/iommuib.h
 1150 file path=usr/include/sys/ipc.h
 1151 file path=usr/include/sys/ipc_impl.h
 1152 file path=usr/include/sys/ipc_rctl.h
 1153 file path=usr/include/sys/isa_defs.h
 1154 file path=usr/include/sys/iso/signal_iso.h
 1155 file path=usr/include/sys/jioctl.h
 1156 file path=usr/include/sys/kbd.h
 1157 file path=usr/include/sys/kbdreg.h
 1158 file path=usr/include/sys/kbio.h
 1159 file path=usr/include/sys/kcpc.h
 1160 file path=usr/include/sys/kd.h
 1161 file path=usr/include/sys/kdi.h
 1162 file path=usr/include/sys/kdi_impl.h
 1163 file path=usr/include/sys/kdi_machimpl.h
 1164 \$(i386_ONLY)file path=usr/include/sys/kdi_regs.h
 1165 file path=usr/include/sys/kiconv.h
 1166 file path=usr/include/sys/kidmap.h
 1167 file path=usr/include/sys/klpd.h
 1168 file path=usr/include/sys/klwp.h
 1169 file path=usr/include/sys/kmem.h
 1170 file path=usr/include/sys/kmem_impl.h
 1171 file path=usr/include/sys/kobj.h
 1172 file path=usr/include/sys/kobj_impl.h
 1173 file path=usr/include/sys/ksocket.h
 1174 file path=usr/include/sys/kstat.h
 1175 file path=usr/include/sys/kstr.h
 1176 file path=usr/include/sys/ksyms.h
 1177 file path=usr/include/sys/ksynch.h
 1178 file path=usr/include/sys/lc_core.h
 1179 file path=usr/include/sys/ldterm.h
 1180 file path=usr/include/sys/lgrp.h

1181 file path=usr/include/sys/lgrp_user.h
 1182 file path=usr/include/sys/link.h
 1183 file path=usr/include/sys/list.h
 1184 file path=usr/include/sys/list_impl.h
 1185 file path=usr/include/sys/llcl.h
 1186 file path=usr/include/sys/loadavg.h
 1187 file path=usr/include/sys/localedef.h
 1188 file path=usr/include/sys/lock.h
 1189 file path=usr/include/sys/lockfs.h
 1190 file path=usr/include/sys/lofi.h
 1191 file path=usr/include/sys/log.h
 1192 file path=usr/include/sys/logindmux.h
 1193 file path=usr/include/sys/lvm/md_basic.h
 1194 file path=usr/include/sys/lvm/md_convert.h
 1195 file path=usr/include/sys/lvm/md_crc.h
 1196 file path=usr/include/sys/lvm/md_hotspares.h
 1197 file path=usr/include/sys/lvm/md_mddb.h
 1198 file path=usr/include/sys/lvm/md_mdiox.h
 1199 file path=usr/include/sys/lvm/md_mhdx.h
 1200 file path=usr/include/sys/lvm/md_mirror.h
 1201 file path=usr/include/sys/lvm/md_mirror_shared.h
 1202 file path=usr/include/sys/lvm/md_names.h
 1203 file path=usr/include/sys/lvm/md_notify.h
 1204 file path=usr/include/sys/lvm/md_raid.h
 1205 file path=usr/include/sys/lvm/md_rename.h
 1206 file path=usr/include/sys/lvm/md_sp.h
 1207 file path=usr/include/sys/lvm/md_stripe.h
 1208 file path=usr/include/sys/lvm/md_trans.h
 1209 file path=usr/include/sys/lvm/mdio.h
 1210 file path=usr/include/sys/lvm/mdmed.h
 1211 file path=usr/include/sys/lvm/mdmn_commd.h
 1212 file path=usr/include/sys/lvm/mdvar.h
 1213 file path=usr/include/sys/lwp.h
 1214 file path=usr/include/sys/lwp_timer_impl.h
 1215 file path=usr/include/sys/lwp_upimutex_impl.h
 1216 file path=usr/include/sys/mac.h
 1217 file path=usr/include/sys/mac_ether.h
 1218 file path=usr/include/sys/mac_flow.h
 1219 file path=usr/include/sys/mac_provider.h
 1220 file path=usr/include/sys/machelf.h
 1221 file path=usr/include/sys/machlock.h
 1222 file path=usr/include/sys/machsig.h
 1223 file path=usr/include/sys/machtypes.h
 1224 file path=usr/include/sys/map.h
 1225 \$(i386_ONLY)file path=usr/include/sys/mc.h
 1226 \$(i386_ONLY)file path=usr/include/sys/mc_amd.h
 1227 \$(i386_ONLY)file path=usr/include/sys/mc_intel.h
 1228 \$(i386_ONLY)file path=usr/include/sys/mca_amd.h
 1229 \$(i386_ONLY)file path=usr/include/sys/mca_x86.h
 1230 file path=usr/include/sys/mcontext.h
 1231 file path=usr/include/sys/md4.h
 1232 file path=usr/include/sys/md5.h
 1233 file path=usr/include/sys/md5_consts.h
 1234 file path=usr/include/sys/mdi_impldefs.h
 1235 file path=usr/include/sys/mem.h
 1236 file path=usr/include/sys/mem_config.h
 1237 file path=usr/include/sys/memlist.h
 1238 file path=usr/include/sys/mhd.h
 1239 file path=usr/include/sys/mii.h
 1240 file path=usr/include/sys/miiregs.h
 1241 file path=usr/include/sys/mkdev.h
 1242 file path=usr/include/sys/mman.h
 1243 file path=usr/include/sys/mmapobj.h
 1244 file path=usr/include/sys/mntent.h
 1245 file path=usr/include/sys/mntio.h
 1246 file path=usr/include/sys/mnttab.h

1247 file path=usr/include/sys/modctl.h
 1248 file path=usr/include/sys/mode.h
 1249 file path=usr/include/sys/model.h
 1250 file path=usr/include/sys/modhash.h
 1251 file path=usr/include/sys/modhash_impl.h
 1252 file path=usr/include/sys/mount.h
 1253 file path=usr/include/sys/mouse.h
 1254 file path=usr/include/sys/msacct.h
 1255 file path=usr/include/sys/msg.h
 1256 file path=usr/include/sys/msg_impl.h
 1257 file path=usr/include/sys/msio.h
 1258 file path=usr/include/sys/msreg.h
 1259 file path=usr/include/sys/mtio.h
 1260 file path=usr/include/sys/multidata.h
 1261 file path=usr/include/sys/mutex.h
 1262 \$(i386_ONLY)file path=usr/include/sys/mutex_impl.h
 1263 file path=usr/include/sys/nbmlck.h
 1264 file path=usr/include/sys/ndi_impldefs.h
 1265 file path=usr/include/sys/ndifm.h
 1266 file path=usr/include/sys/netconfig.h
 1267 file path=usr/include/sys/neti.h
 1268 file path=usr/include/sys/netstack.h
 1269 file path=usr/include/sys/nexusdefs.h
 1270 file path=usr/include/sys/note.h
 1271 file path=usr/include/sys/null.h
 1272 file path=usr/include/sys/nvpair.h
 1273 file path=usr/include/sys/nvpair_impl.h
 1274 file path=usr/include/sys/objfs.h
 1275 file path=usr/include/sys/objfs_impl.h
 1276 file path=usr/include/sys/obpdefs.h
 1277 file path=usr/include/sys/old_procfs.h
 1278 file path=usr/include/sys/open.h
 1279 file path=usr/include/sys/openpromio.h
 1280 file path=usr/include/sys/panic.h
 1281 file path=usr/include/sys/param.h
 1282 file path=usr/include/sys/pathconf.h
 1283 file path=usr/include/sys/pathname.h
 1284 file path=usr/include/sys/pattr.h
 1285 file path=usr/include/sys/pbio.h
 1286 file path=usr/include/sys/pcb.h
 1287 file path=usr/include/sys/pccard.h
 1288 file path=usr/include/sys/pci.h
 1289 \$(i386_ONLY)file path=usr/include/sys/pcic_reg.h
 1290 \$(i386_ONLY)file path=usr/include/sys/pcic_var.h
 1291 file path=usr/include/sys/pcie.h
 1292 file path=usr/include/sys/pcmcia.h
 1293 file path=usr/include/sys/pctypes.h
 1294 file path=usr/include/sys/pfmod.h
 1295 file path=usr/include/sys/pg.h
 1296 file path=usr/include/sys/pghw.h
 1297 file path=usr/include/sys/phymem.h
 1298 \$(i386_ONLY)file path=usr/include/sys/pic.h
 1299 \$(i386_ONLY)file path=usr/include/sys/pit.h
 1300 file path=usr/include/sys/pkp_hash.h
 1301 file path=usr/include/sys/pm.h
 1302 \$(i386_ONLY)file path=usr/include/sys/pmem.h
 1303 file path=usr/include/sys/policy.h
 1304 file path=usr/include/sys/poll.h
 1305 file path=usr/include/sys/poll_impl.h
 1306 file path=usr/include/sys/pool.h
 1307 file path=usr/include/sys/pool_impl.h
 1308 file path=usr/include/sys/pool_pset.h
 1309 file path=usr/include/sys/port.h
 1310 file path=usr/include/sys/port_impl.h
 1311 file path=usr/include/sys/port_kernel.h
 1312 file path=usr/include/sys/ppmio.h

```

1313 file path=usr/include/sys/priocntl.h
1314 file path=usr/include/sys/priv.h
1315 file path=usr/include/sys/priv_const.h
1316 file path=usr/include/sys/priv_impl.h
1317 file path=usr/include/sys/priv_names.h
1318 $(i386_ONLY)file path=usr/include/sys/privmregs.h
1319 $(i386_ONLY)file path=usr/include/sys/privregs.h
1320 file path=usr/include/sys/prnio.h
1321 file path=usr/include/sys/proc.h
1322 file path=usr/include/sys/proc/prdata.h
1323 file path=usr/include/sys/processor.h
1324 file path=usr/include/sys/procfs.h
1325 file path=usr/include/sys/procfs_isa.h
1326 file path=usr/include/sys/procset.h
1327 file path=usr/include/sys/project.h
1328 $(i386_ONLY)file path=usr/include/sys/prom_emul.h
1329 $(i386_ONLY)file path=usr/include/sys/prom_isa.h
1330 $(i386_ONLY)file path=usr/include/sys/prom_plat.h
1331 file path=usr/include/sys/promif.h
1332 file path=usr/include/sys/promimpl.h
1333 file path=usr/include/sys/protosw.h
1334 file path=usr/include/sys/prsystem.h
1335 file path=usr/include/sys/pset.h
1336 file path=usr/include/sys/psw.h
1337 $(i386_ONLY)file path=usr/include/sys/pte.h
1338 file path=usr/include/sys/ptem.h
1339 file path=usr/include/sys/ptms.h
1340 file path=usr/include/sys/ptyvar.h
1341 file path=usr/include/sys/queue.h
1342 file path=usr/include/sys/raidioctl.h
1343 file path=usr/include/sys/ramdisk.h
1344 file path=usr/include/sys/random.h
1345 file path=usr/include/sys/rctl.h
1346 file path=usr/include/sys/rctl_impl.h
1347 file path=usr/include/sys/rds.h
1348 file path=usr/include/sys/reboot.h
1349 file path=usr/include/sys/refstr.h
1350 file path=usr/include/sys/refstr_impl.h
1351 file path=usr/include/sys/reg.h
1352 file path=usr/include/sys/regset.h
1353 file path=usr/include/sys/resource.h
1354 file path=usr/include/sys/rliocntl.h
1355 file path=usr/include/sys/rsm/rsm.h
1356 file path=usr/include/sys/rsm/rsm_common.h
1357 file path=usr/include/sys/rsm/rsmapi_common.h
1358 file path=usr/include/sys/rsm/rsmka_path_int.h
1359 file path=usr/include/sys/rsm/rsmmdi.h
1360 file path=usr/include/sys/rsm/rsmmpi.h
1361 file path=usr/include/sys/rsm/rsmmpi_driver.h
1362 file path=usr/include/sys/rt.h
1363 $(i386_ONLY)file path=usr/include/sys/rtc.h
1364 file path=usr/include/sys/rtpriocntl.h
1365 file path=usr/include/sys/rwlock.h
1366 file path=usr/include/sys/rwlock_impl.h
1367 file path=usr/include/sys/rwstlock.h
1368 file path=usr/include/sys/sad.h
1369 $(i386_ONLY)file path=usr/include/sys/sata/sata_defs.h
1370 $(i386_ONLY)file path=usr/include/sys/sata/sata_hba.h
1371 file path=usr/include/sys/schedctl.h
1372 $(sparc_ONLY)file path=usr/include/sys/scsi/adapters/ifpio.h
1373 file path=usr/include/sys/scsi/adapters/scsi_vhci.h
1374 $(sparc_ONLY)file path=usr/include/sys/scsi/adapters/sfvar.h
1375 file path=usr/include/sys/scsi/conf/autoconf.h
1376 file path=usr/include/sys/scsi/conf/device.h
1377 file path=usr/include/sys/scsi/generic/commands.h
1378 file path=usr/include/sys/scsi/generic/dad_mode.h

```

```

1379 file path=usr/include/sys/scsi/generic/inquiry.h
1380 file path=usr/include/sys/scsi/generic/message.h
1381 file path=usr/include/sys/scsi/generic/mode.h
1382 file path=usr/include/sys/scsi/generic/persist.h
1383 file path=usr/include/sys/scsi/generic/sense.h
1384 file path=usr/include/sys/scsi/generic/sff_frames.h
1385 file path=usr/include/sys/scsi/generic/smp_frames.h
1386 file path=usr/include/sys/scsi/generic/status.h
1387 file path=usr/include/sys/scsi/impl/commands.h
1388 file path=usr/include/sys/scsi/impl/inquiry.h
1389 file path=usr/include/sys/scsi/impl/mode.h
1390 file path=usr/include/sys/scsi/impl/scsi_reset_notify.h
1391 file path=usr/include/sys/scsi/impl/scsi_sas.h
1392 file path=usr/include/sys/scsi/impl/sense.h
1393 file path=usr/include/sys/scsi/impl/services.h
1394 file path=usr/include/sys/scsi/impl/smp_transport.h
1395 file path=usr/include/sys/scsi/impl/spc3_types.h
1396 file path=usr/include/sys/scsi/impl/status.h
1397 file path=usr/include/sys/scsi/impl/transport.h
1398 file path=usr/include/sys/scsi/impl/types.h
1399 file path=usr/include/sys/scsi/impl/uscsi.h
1400 file path=usr/include/sys/scsi/impl/usmp.h
1401 file path=usr/include/sys/scsi/scsi.h
1402 file path=usr/include/sys/scsi/scsi_address.h
1403 file path=usr/include/sys/scsi/scsi_ctl.h
1404 file path=usr/include/sys/scsi/scsi_fm.h
1405 file path=usr/include/sys/scsi/scsi_names.h
1406 file path=usr/include/sys/scsi/scsi_params.h
1407 file path=usr/include/sys/scsi/scsi_pkt.h
1408 file path=usr/include/sys/scsi/scsi_resource.h
1409 file path=usr/include/sys/scsi/scsi_types.h
1410 file path=usr/include/sys/scsi/scsi_watch.h
1411 file path=usr/include/sys/scsi/targets/sddef.h
1412 file path=usr/include/sys/scsi/targets/ses.h
1413 file path=usr/include/sys/scsi/targets/sesio.h
1414 file path=usr/include/sys/scsi/targets/sgendef.h
1415 file path=usr/include/sys/scsi/targets/smp.h
1416 $(sparc_ONLY)file path=usr/include/sys/scsi/targets/ssddef.h
1417 file path=usr/include/sys/scsi/targets/stdef.h
1418 file path=usr/include/sys/secflags.h
1419 #endif /* ! codereview */
1420 $(i386_ONLY)file path=usr/include/sys/segment.h
1421 $(i386_ONLY)file path=usr/include/sys/segments.h
1422 file path=usr/include/sys/select.h
1423 file path=usr/include/sys/sem.h
1424 file path=usr/include/sys/sem_impl.h
1425 file path=usr/include/sys/semaphore.h
1426 file path=usr/include/sys/semaphore.h
1427 file path=usr/include/sys/sendfile.h
1428 $(sparc_ONLY)file path=usr/include/sys/ser_async.h
1429 file path=usr/include/sys/ser_sync.h
1430 $(sparc_ONLY)file path=usr/include/sys/ser_zscc.h
1431 file path=usr/include/sys/serializer.h
1432 file path=usr/include/sys/session.h
1433 file path=usr/include/sys/sha1.h
1434 file path=usr/include/sys/sha2.h
1435 file path=usr/include/sys/share.h
1436 file path=usr/include/sys/shm.h
1437 file path=usr/include/sys/shm_impl.h
1438 file path=usr/include/sys/sid.h
1439 file path=usr/include/sys/siginfo.h
1440 file path=usr/include/sys/signal.h
1441 file path=usr/include/sys/signalfd.h
1442 file path=usr/include/sys/skein.h
1443 file path=usr/include/sys/sleepq.h
1444 file path=usr/include/sys/smbios.h

```

```

1445 file path=usr/include/sys/smbios_impl.h
1446 file path=usr/include/sys/smedia.h
1447 file path=usr/include/sys/sobject.h
1448 $(sparc_ONLY)file path=usr/include/sys/socal_cq_defs.h
1449 $(sparc_ONLY)file path=usr/include/sys/socalio.h
1450 $(sparc_ONLY)file path=usr/include/sys/socalmap.h
1451 $(sparc_ONLY)file path=usr/include/sys/socalreg.h
1452 $(sparc_ONLY)file path=usr/include/sys/socalvar.h
1453 file path=usr/include/sys/socket.h
1454 file path=usr/include/sys/socket_impl.h
1455 file path=usr/include/sys/socket_proto.h
1456 file path=usr/include/sys/socketvar.h
1457 file path=usr/include/sys/sockio.h
1458 file path=usr/include/sys/spl.h
1459 file path=usr/include/sys/queue.h
1460 file path=usr/include/sys/queue_impl.h
1461 file path=usr/include/sys/sservice.h
1462 file path=usr/include/sys/stack.h
1463 file path=usr/include/sys/stat.h
1464 file path=usr/include/sys/stat_impl.h
1465 file path=usr/include/sys/statfs.h
1466 file path=usr/include/sys/statvfs.h
1467 file path=usr/include/sys/stdbool.h
1468 file path=usr/include/sys/stdint.h
1469 file path=usr/include/sys/stermio.h
1470 file path=usr/include/sys/stream.h
1471 file path=usr/include/sys/strft.h
1472 file path=usr/include/sys/strlog.h
1473 file path=usr/include/sys/strmdep.h
1474 file path=usr/include/sys/stropts.h
1475 file path=usr/include/sys/strredir.h
1476 file path=usr/include/sys/strstat.h
1477 file path=usr/include/sys/strsubr.h
1478 file path=usr/include/sys/strsun.h
1479 file path=usr/include/sys/strtty.h
1480 file path=usr/include/sys/sunddi.h
1481 file path=usr/include/sys/sunldi.h
1482 file path=usr/include/sys/sunldi_impl.h
1483 file path=usr/include/sys/sunmdi.h
1484 file path=usr/include/sys/sunndi.h
1485 file path=usr/include/sys/sunpm.h
1486 file path=usr/include/sys/suntpi.h
1487 file path=usr/include/sys/suntty.h
1488 file path=usr/include/sys/swap.h
1489 file path=usr/include/sys/synch.h
1490 file path=usr/include/sys/syscall.h
1491 file path=usr/include/sys/sysconf.h
1492 file path=usr/include/sys/sysconfig.h
1493 file path=usr/include/sys/sysconfig_impl.h
1494 file path=usr/include/sys/sysdc.h
1495 file path=usr/include/sys/sysdc_impl.h
1496 file path=usr/include/sys/sysevent.h
1497 file path=usr/include/sys/sysevent/ap_driver.h
1498 file path=usr/include/sys/sysevent/dev.h
1499 file path=usr/include/sys/sysevent/domain.h
1500 file path=usr/include/sys/sysevent/dr.h
1501 file path=usr/include/sys/sysevent/env.h
1502 file path=usr/include/sys/sysevent/eventdefs.h
1503 file path=usr/include/sys/sysevent/ipmp.h
1504 file path=usr/include/sys/sysevent/pwrctl.h
1505 file path=usr/include/sys/sysevent/svm.h
1506 file path=usr/include/sys/sysevent/vrrp.h
1507 file path=usr/include/sys/sysevent_impl.h
1508 $(i386_ONLY)file path=usr/include/sys/sysi86.h
1509 file path=usr/include/sys/sysinfo.h
1510 file path=usr/include/sys/syslog.h

```

```

1511 file path=usr/include/sys/sysmacros.h
1512 file path=usr/include/sys/systeminfo.h
1513 file path=usr/include/sys/system.h
1514 file path=usr/include/sys/t_kuser.h
1515 file path=usr/include/sys/t_lock.h
1516 file path=usr/include/sys/task.h
1517 file path=usr/include/sys/taskq.h
1518 file path=usr/include/sys/taskq_impl.h
1519 file path=usr/include/sys/teliocctl.h
1520 file path=usr/include/sys/termio.h
1521 file path=usr/include/sys/termios.h
1522 file path=usr/include/sys/termiox.h
1523 file path=usr/include/sys/thread.h
1524 file path=usr/include/sys/ticlts.h
1525 file path=usr/include/sys/ticots.h
1526 file path=usr/include/sys/ticotsord.h
1527 file path=usr/include/sys/tihdr.h
1528 file path=usr/include/sys/time.h
1529 file path=usr/include/sys/time_impl.h
1530 file path=usr/include/sys/time_std_impl.h
1531 file path=usr/include/sys/timeb.h
1532 file path=usr/include/sys/timer.h
1533 file path=usr/include/sys/timerfd.h
1534 file path=usr/include/sys/times.h
1535 file path=usr/include/sys/timex.h
1536 file path=usr/include/sys/timod.h
1537 file path=usr/include/sys/tirdwr.h
1538 file path=usr/include/sys/tiuser.h
1539 file path=usr/include/sys/tl.h
1540 file path=usr/include/sys/tnf.h
1541 file path=usr/include/sys/tnf_com.h
1542 file path=usr/include/sys/tnf_probe.h
1543 file path=usr/include/sys/tnf_writer.h
1544 file path=usr/include/sys/todio.h
1545 file path=usr/include/sys/tpicommon.h
1546 file path=usr/include/sys/trap.h
1547 $(i386_ONLY)file path=usr/include/sys/traptrace.h
1548 file path=usr/include/sys/ts.h
1549 file path=usr/include/sys/tsol/label.h
1550 file path=usr/include/sys/tsol/label_macro.h
1551 file path=usr/include/sys/tsol/priv.h
1552 file path=usr/include/sys/tsol/tndb.h
1553 file path=usr/include/sys/tsol/tsyscall.h
1554 file path=usr/include/sys/tspricocntl.h
1555 $(i386_ONLY)file path=usr/include/sys/tss.h
1556 file path=usr/include/sys/ttcompat.h
1557 file path=usr/include/sys/ttold.h
1558 file path=usr/include/sys/tty.h
1559 file path=usr/include/sys/ttychars.h
1560 file path=usr/include/sys/ttydev.h
1561 $(sparc_ONLY)file path=usr/include/sys/ttymux.h
1562 $(sparc_ONLY)file path=usr/include/sys/ttymuxuser.h
1563 file path=usr/include/sys/tuneable.h
1564 file path=usr/include/sys/turnstile.h
1565 file path=usr/include/sys/types.h
1566 file path=usr/include/sys/types32.h
1567 file path=usr/include/sys/tzfile.h
1568 file path=usr/include/sys/u8_textprep.h
1569 file path=usr/include/sys/uadmin.h
1570 $(i386_ONLY)file path=usr/include/sys/ucode.h
1571 file path=usr/include/sys/ucontext.h
1572 file path=usr/include/sys/uio.h
1573 file path=usr/include/sys/ulimit.h
1574 file path=usr/include/sys/un.h
1575 file path=usr/include/sys/unistd.h
1576 file path=usr/include/sys/user.h

```

```
1577 file path=usr/include/sys/ustat.h
1578 file path=usr/include/sys/utime.h
1579 file path=usr/include/sys/utrap.h
1580 file path=usr/include/sys/utsname.h
1581 file path=usr/include/sys/utssys.h
1582 file path=usr/include/sys/uuid.h
1583 file path=usr/include/sys/va_impl.h
1584 file path=usr/include/sys/va_list.h
1585 file path=usr/include/sys/var.h
1586 file path=usr/include/sys/varargs.h
1587 file path=usr/include/sys/vfs.h
1588 file path=usr/include/sys/vfs_opreg.h
1589 file path=usr/include/sys/vfstab.h
1590 file path=usr/include/sys/videodev2.h
1591 file path=usr/include/sys/visual_io.h
1592 file path=usr/include/sys/vm.h
1593 file path=usr/include/sys/vm_usage.h
1594 file path=usr/include/sys/vmem.h
1595 file path=usr/include/sys/vmem_impl.h
1596 file path=usr/include/sys/vmem_impl_user.h
1597 file path=usr/include/sys/vmparam.h
1598 file path=usr/include/sys/vmsystem.h
1599 file path=usr/include/sys/vnode.h
1600 file path=usr/include/sys/vt.h
1601 file path=usr/include/sys/vtdaemon.h
1602 file path=usr/include/sys/vtoc.h
1603 file path=usr/include/sys/vtrace.h
1604 file path=usr/include/sys/vuid_event.h
1605 file path=usr/include/sys/vuid_queue.h
1606 file path=usr/include/sys/vuid_state.h
1607 file path=usr/include/sys/vuid_store.h
1608 file path=usr/include/sys/vuid_wheel.h
1609 file path=usr/include/sys/wait.h
1610 file path=usr/include/sys/waitq.h
1611 file path=usr/include/sys/watchpoint.h
1612 $(i386_ONLY)file path=usr/include/sys/x86_archext.h
1613 $(i386_ONLY)file path=usr/include/sys/xen_errno.h
1614 file path=usr/include/sys/xti_inet.h
1615 file path=usr/include/sys/xti_osi.h
1616 file path=usr/include/sys/xti_xtiopt.h
1617 file path=usr/include/sys/zcons.h
1618 file path=usr/include/sys/zmod.h
1619 file path=usr/include/sys/zone.h
1620 $(sparc_ONLY)file path=usr/include/sys/zsdev.h
1621 file path=usr/include/sysexits.h
1622 file path=usr/include/syslog.h
1623 file path=usr/include/tar.h
1624 file path=usr/include/tcpd.h
1625 file path=usr/include/term.h
1626 file path=usr/include/termcap.h
1627 file path=usr/include/termio.h
1628 file path=usr/include/termios.h
1629 file path=usr/include/thread.h
1630 file path=usr/include/thread_db.h
1631 file path=usr/include/threads.h
1632 file path=usr/include/time.h
1633 file path=usr/include/tiuser.h
1634 file path=usr/include/tsol/label.h
1635 file path=usr/include/tzfile.h
1636 file path=usr/include/ucontext.h
1637 file path=usr/include/ucrd.h
1638 file path=usr/include/uid_stp.h
1639 file path=usr/include/ulimit.h
1640 file path=usr/include/umem.h
1641 file path=usr/include/umem_impl.h
1642 file path=usr/include/unctrl.h
```

```
1643 file path=usr/include/unistd.h
1644 file path=usr/include/user_attr.h
1645 file path=usr/include/userdefs.h
1646 file path=usr/include/ustat.h
1647 file path=usr/include/utility.h
1648 file path=usr/include/utime.h
1649 file path=usr/include/utmp.h
1650 file path=usr/include/utmpx.h
1651 file path=usr/include/uuid/uuid.h
1652 $(sparc_ONLY)file path=usr/include/v7/sys/machpcb.h
1653 $(sparc_ONLY)file path=usr/include/v7/sys/machtrap.h
1654 $(sparc_ONLY)file path=usr/include/v7/sys/mutex_impl.h
1655 $(sparc_ONLY)file path=usr/include/v7/sys/privregs.h
1656 $(sparc_ONLY)file path=usr/include/v7/sys/prom_isa.h
1657 $(sparc_ONLY)file path=usr/include/v7/sys/psr.h
1658 $(sparc_ONLY)file path=usr/include/v7/sys/traptrace.h
1659 $(sparc_ONLY)file path=usr/include/v9/sys/asi.h
1660 $(sparc_ONLY)file path=usr/include/v9/sys/machpcb.h
1661 $(sparc_ONLY)file path=usr/include/v9/sys/machtrap.h
1662 $(sparc_ONLY)file path=usr/include/v9/sys/membar.h
1663 $(sparc_ONLY)file path=usr/include/v9/sys/mutex_impl.h
1664 $(sparc_ONLY)file path=usr/include/v9/sys/privregs.h
1665 $(sparc_ONLY)file path=usr/include/v9/sys/prom_isa.h
1666 $(sparc_ONLY)file path=usr/include/v9/sys/psr_compat.h
1667 $(sparc_ONLY)file path=usr/include/v9/sys/vis_simulator.h
1668 file path=usr/include/valtools.h
1669 file path=usr/include/values.h
1670 file path=usr/include/varargs.h
1671 file path=usr/include/vm/anon.h
1672 file path=usr/include/vm/as.h
1673 file path=usr/include/vm/faultcode.h
1674 file path=usr/include/vm/hat.h
1675 file path=usr/include/vm/kpm.h
1676 file path=usr/include/vm/page.h
1677 file path=usr/include/vm/pvn.h
1678 file path=usr/include/vm/rm.h
1679 file path=usr/include/vm/seg.h
1680 file path=usr/include/vm/seg_dev.h
1681 file path=usr/include/vm/seg_enum.h
1682 file path=usr/include/vm/seg_kmem.h
1683 file path=usr/include/vm/seg_kp.h
1684 file path=usr/include/vm/seg_kpm.h
1685 file path=usr/include/vm/seg_map.h
1686 file path=usr/include/vm/seg_spt.h
1687 file path=usr/include/vm/seg_vn.h
1688 file path=usr/include/vm/vpage.h
1689 file path=usr/include/vm/vpm.h
1690 file path=usr/include/volmgt.h
1691 file path=usr/include/wait.h
1692 file path=usr/include/wchar.h
1693 file path=usr/include/wchar_impl.h
1694 file path=usr/include/wctype.h
1695 file path=usr/include/widex.h
1696 file path=usr/include/wordexp.h
1697 file path=usr/include/xlocale.h
1698 file path=usr/include/xti.h
1699 file path=usr/include/xti_inet.h
1700 file path=usr/include/zone.h
1701 file path=usr/include/zonestat.h
1702 $(i386_ONLY)file path=usr/platform/i86pc/include/sys/acpidev.h
1703 $(i386_ONLY)file path=usr/platform/i86pc/include/sys/amd_iommu.h
1704 $(i386_ONLY)file path=usr/platform/i86pc/include/sys/asm_misc.h
1705 $(i386_ONLY)file path=usr/platform/i86pc/include/sys/clock.h
1706 $(i386_ONLY)file path=usr/platform/i86pc/include/sys/cram.h
1707 $(i386_ONLY)file path=usr/platform/i86pc/include/sys/ddi_subrdefs.h
1708 $(i386_ONLY)file path=usr/platform/i86pc/include/sys/debug_info.h
```



```

1709 $(i386_ONLY)file path=usr/platform/i86pc/include/sys/fastboot.h
1710 $(i386_ONLY)file path=usr/platform/i86pc/include/sys/mach_mmu.h
1711 $(i386_ONLY)file path=usr/platform/i86pc/include/sys/machclock.h
1712 $(i386_ONLY)file path=usr/platform/i86pc/include/sys/machcpuvar.h
1713 $(i386_ONLY)file path=usr/platform/i86pc/include/sys/machparam.h
1714 $(i386_ONLY)file path=usr/platform/i86pc/include/sys/machprivregs.h
1715 $(i386_ONLY)file path=usr/platform/i86pc/include/sys/machsystem.h
1716 $(i386_ONLY)file path=usr/platform/i86pc/include/sys/machthread.h
1717 $(i386_ONLY)file path=usr/platform/i86pc/include/sys/memnode.h
1718 $(i386_ONLY)file path=usr/platform/i86pc/include/sys/pc_mmu.h
1719 $(i386_ONLY)file path=usr/platform/i86pc/include/sys/psm.h
1720 $(i386_ONLY)file path=usr/platform/i86pc/include/sys/psm_defs.h
1721 $(i386_ONLY)file path=usr/platform/i86pc/include/sys/psm_modctl.h
1722 $(i386_ONLY)file path=usr/platform/i86pc/include/sys/psm_types.h
1723 $(i386_ONLY)file path=usr/platform/i86pc/include/sys/rm_platform.h
1724 $(i386_ONLY)file path=usr/platform/i86pc/include/sys/sbd_ioctl.h
1725 $(i386_ONLY)file path=usr/platform/i86pc/include/sys/smp_impldefs.h
1726 $(i386_ONLY)file path=usr/platform/i86pc/include/sys/vm_machparam.h
1727 $(i386_ONLY)file path=usr/platform/i86pc/include/sys/x_call.h
1728 $(i386_ONLY)file path=usr/platform/i86pc/include/sys/xc_levels.h
1729 $(i386_ONLY)file path=usr/platform/i86pc/include/sys/xsvc.h
1730 $(i386_ONLY)file path=usr/platform/i86pc/include/vm/hat_i86.h
1731 $(i386_ONLY)file path=usr/platform/i86pc/include/vm/hat_pte.h
1732 $(i386_ONLY)file path=usr/platform/i86pc/include/vm/hment.h
1733 $(i386_ONLY)file path=usr/platform/i86pc/include/vm/htable.h
1734 $(i386_ONLY)file path=usr/platform/i86pc/include/vm/kboot_mmu.h
1735 $(i386_ONLY)file path=usr/platform/i86xpv/include/sys/balloon.h
1736 $(i386_ONLY)file path=usr/platform/i86xpv/include/sys/machprivregs.h
1737 $(i386_ONLY)file path=usr/platform/i86xpv/include/sys/xen_mmu.h
1738 $(i386_ONLY)file path=usr/platform/i86xpv/include/sys/xpv_impl.h
1739 $(i386_ONLY)file path=usr/platform/i86xpv/include/vm/seg_mf.h
1740 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/ac.h
1741 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/async.h
1742 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/cheatahregs.h
1743 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/cherrystone.h
1744 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/clock.h
1745 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/cmp.h
1746 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/cpc_ultra.h
1747 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/cpr_impl.h
1748 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/cpu_impl.h
1749 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/cpu_sgnblk_defs.h
1750 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/cvc.h
1751 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/daktari.h
1752 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/ddi_subrdefs.h
1753 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/dvma.h
1754 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/ecc_kstat.h
1755 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/EEPROM.h
1756 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/envctrl.h
1757 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/envctrl_gen.h
1758 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/envctrl_ue250.h
1759 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/envctrl_ue450.h
1760 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/environ.h
1761 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/errclassify.h
1762 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/fhc.h
1763 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/gpio_87317.h
1764 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/hpc3130_events.h
1765 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/i2c_clients/hpc3130.h
1766 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/i2c_clients/i2c_client.h
1767 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/i2c_clients/lm75.h
1768 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/i2c_clients/max1617.h
1769 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/i2c_clients/pcf8591.h
1770 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/i2c_clients/ssc050.h
1771 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/i2c/misc/i2c_svc.h
1772 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/idprom.h
1773 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/intr.h
1774 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/intreg.h

```

```

1775 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/iocache.h
1776 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/iommu.h
1777 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/ivintr.h
1778 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/lom_io.h
1779 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/machasi.h
1780 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/machclock.h
1781 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/machcpuvar.h
1782 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/machparam.h
1783 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/machsystem.h
1784 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/machthread.h
1785 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/mem_cache.h
1786 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/memlist_plat.h
1787 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/memnode.h
1788 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/mmu.h
1789 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/nexusdebug.h
1790 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/opl_hwdesc.h
1791 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/opl_module.h
1792 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/prom_debug.h
1793 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/prom_plat.h
1794 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/pte.h
1795 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/sbd_ioctl.h
1796 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/scb.h
1797 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/scsb_led.h
1798 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/simmstat.h
1799 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/splitregs.h
1800 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/sram.h
1801 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/starfire.h
1802 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/sun4asi.h
1803 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/sysctrl.h
1804 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/sysioerr.h
1805 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/syiosbus.h
1806 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/tod.h
1807 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/todmostek.h
1808 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/trapstat.h
1809 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/traptrace.h
1810 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/vis.h
1811 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/vm_machparam.h
1812 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/x_call.h
1813 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/xc_impl.h
1814 $(sparc_ONLY)file path=usr/platform/sun4u/include/sys/xsmach.h
1815 $(sparc_ONLY)file path=usr/platform/sun4u/include/vm/hat_sfmmu.h
1816 $(sparc_ONLY)file path=usr/platform/sun4u/include/vm/mach_sfmmu.h
1817 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/clock.h
1818 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/cmp.h
1819 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/cpc_ultra.h
1820 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/cpu_sgnblk_defs.h
1821 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/ddi_subrdefs.h
1822 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/ds_pri.h
1823 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/ds_snmp.h
1824 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/dvma.h
1825 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/EEPROM.h
1826 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/fcode.h
1827 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/hsvc.h
1828 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/hypervisor_api.h
1829 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/idprom.h
1830 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/intr.h
1831 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/intreg.h
1832 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/ivintr.h
1833 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/machasi.h
1834 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/machclock.h
1835 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/machcpuvar.h
1836 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/machintreg.h
1837 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/machparam.h
1838 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/machsystem.h
1839 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/machthread.h
1840 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/memlist_plat.h

```

```

1841 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/memnode.h
1842 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/mmu.h
1843 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/nexusdebug.h
1844 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/niagaraasi.h
1845 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/niagararegs.h
1846 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/ntwdt.h
1847 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/pri.h
1848 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/prom_debug.h
1849 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/prom_plat.h
1850 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/pte.h
1851 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/trapstat.h
1852 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/scb.h
1853 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/soft_state.h
1854 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/sun4asi.h
1855 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/tod.h
1856 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/trapstat.h
1857 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/traptrace.h
1858 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/vis.h
1859 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/vm_machparam.h
1860 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/x_call.h
1861 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/xc_impl.h
1862 $(sparc_ONLY)file path=usr/platform/sun4v/include/sys/zsmach.h
1863 $(sparc_ONLY)file path=usr/platform/sun4v/include/vm/hat_sfmmu.h
1864 $(sparc_ONLY)file path=usr/platform/sun4v/include/vm/mach_sfmmu.h
1865 file path=usr/share/man/man3head/acct.h.3head
1866 file path=usr/share/man/man3head/aio.h.3head
1867 file path=usr/share/man/man3head/ar.h.3head
1868 file path=usr/share/man/man3head/archives.h.3head
1869 file path=usr/share/man/man3head/assert.h.3head
1870 file path=usr/share/man/man3head/complex.h.3head
1871 file path=usr/share/man/man3head/cpio.h.3head
1872 file path=usr/share/man/man3head/dirent.h.3head
1873 file path=usr/share/man/man3head/endian.h.3head
1874 file path=usr/share/man/man3head/errno.h.3head
1875 file path=usr/share/man/man3head/fcntl.h.3head
1876 file path=usr/share/man/man3head/fenv.h.3head
1877 file path=usr/share/man/man3head/float.h.3head
1878 file path=usr/share/man/man3head/floatingpoint.h.3head
1879 file path=usr/share/man/man3head/fmtmsg.h.3head
1880 file path=usr/share/man/man3head/fnmatch.h.3head
1881 file path=usr/share/man/man3head/ftw.h.3head
1882 file path=usr/share/man/man3head/glob.h.3head
1883 file path=usr/share/man/man3head/grp.h.3head
1884 file path=usr/share/man/man3head/iconv.h.3head
1885 file path=usr/share/man/man3head/if.h.3head
1886 file path=usr/share/man/man3head/in.h.3head
1887 file path=usr/share/man/man3head/inet.h.3head
1888 file path=usr/share/man/man3head/inttypes.h.3head
1889 file path=usr/share/man/man3head/ipc.h.3head
1890 file path=usr/share/man/man3head/iso646.h.3head
1891 file path=usr/share/man/man3head/langinfo.h.3head
1892 file path=usr/share/man/man3head/libgen.h.3head
1893 file path=usr/share/man/man3head/libintl.h.3head
1894 file path=usr/share/man/man3head/limits.h.3head
1895 file path=usr/share/man/man3head/locale.h.3head
1896 file path=usr/share/man/man3head/math.h.3head
1897 file path=usr/share/man/man3head/mman.h.3head
1898 file path=usr/share/man/man3head/monetary.h.3head
1899 file path=usr/share/man/man3head/mqueue.h.3head
1900 file path=usr/share/man/man3head/msg.h.3head
1901 file path=usr/share/man/man3head/ndbm.h.3head
1902 file path=usr/share/man/man3head/netdb.h.3head
1903 file path=usr/share/man/man3head/nl_types.h.3head
1904 file path=usr/share/man/man3head/poll.h.3head
1905 file path=usr/share/man/man3head/pthread.h.3head
1906 file path=usr/share/man/man3head/pwd.h.3head

```

```

1907 file path=usr/share/man/man3head/regex.h.3head
1908 file path=usr/share/man/man3head/resource.h.3head
1909 file path=usr/share/man/man3head/sched.h.3head
1910 file path=usr/share/man/man3head/search.h.3head
1911 file path=usr/share/man/man3head/select.h.3head
1912 file path=usr/share/man/man3head/sem.h.3head
1913 file path=usr/share/man/man3head/semaphore.h.3head
1914 file path=usr/share/man/man3head/setjmp.h.3head
1915 file path=usr/share/man/man3head/shm.h.3head
1916 file path=usr/share/man/man3head/signinfo.h.3head
1917 file path=usr/share/man/man3head/signal.h.3head
1918 file path=usr/share/man/man3head/socket.h.3head
1919 file path=usr/share/man/man3head/spawn.h.3head
1920 file path=usr/share/man/man3head/stat.h.3head
1921 file path=usr/share/man/man3head/statvfs.h.3head
1922 file path=usr/share/man/man3head/stdbool.h.3head
1923 file path=usr/share/man/man3head/stddef.h.3head
1924 file path=usr/share/man/man3head/stdint.h.3head
1925 file path=usr/share/man/man3head/stdio.h.3head
1926 file path=usr/share/man/man3head/stdlib.h.3head
1927 file path=usr/share/man/man3head/string.h.3head
1928 file path=usr/share/man/man3head/strings.h.3head
1929 file path=usr/share/man/man3head/stropts.h.3head
1930 file path=usr/share/man/man3head/syslog.h.3head
1931 file path=usr/share/man/man3head/tar.h.3head
1932 file path=usr/share/man/man3head/tcp.h.3head
1933 file path=usr/share/man/man3head/termios.h.3head
1934 file path=usr/share/man/man3head/tgmath.h.3head
1935 file path=usr/share/man/man3head/time.h.3head
1936 file path=usr/share/man/man3head/timeb.h.3head
1937 file path=usr/share/man/man3head/times.h.3head
1938 file path=usr/share/man/man3head/types.h.3head
1939 file path=usr/share/man/man3head/types32.h.3head
1940 file path=usr/share/man/man3head/ucontext.h.3head
1941 file path=usr/share/man/man3head/uio.h.3head
1942 file path=usr/share/man/man3head/ulimit.h.3head
1943 file path=usr/share/man/man3head/un.h.3head
1944 file path=usr/share/man/man3head/unistd.h.3head
1945 file path=usr/share/man/man3head/utime.h.3head
1946 file path=usr/share/man/man3head/utmpx.h.3head
1947 file path=usr/share/man/man3head/utsname.h.3head
1948 file path=usr/share/man/man3head/values.h.3head
1949 file path=usr/share/man/man3head/wait.h.3head
1950 file path=usr/share/man/man3head/wchar.h.3head
1951 file path=usr/share/man/man3head/wctype.h.3head
1952 file path=usr/share/man/man3head/wordexp.h.3head
1953 file path=usr/share/man/man3head/xlocale.h.3head
1954 file path=usr/share/man/man4/note.4
1955 file path=usr/share/man/man5/prof.5
1956 file path=usr/share/man/man7i/cdio.7i
1957 file path=usr/share/man/man7i/dkio.7i
1958 file path=usr/share/man/man7i/fbio.7i
1959 file path=usr/share/man/man7i/fdio.7i
1960 file path=usr/share/man/man7i/hdio.7i
1961 file path=usr/share/man/man7i/ieec61883.7i
1962 file path=usr/share/man/man7i/mhd.7i
1963 file path=usr/share/man/man7i/mtio.7i
1964 file path=usr/share/man/man7i/prnio.7i
1965 file path=usr/share/man/man7i/quotactl.7i
1966 file path=usr/share/man/man7i/sesio.7i
1967 file path=usr/share/man/man7i/sockio.7i
1968 file path=usr/share/man/man7i/streamio.7i
1969 file path=usr/share/man/man7i/termio.7i
1970 file path=usr/share/man/man7i/termiox.7i
1971 file path=usr/share/man/man7i/uscsi.7i
1972 file path=usr/share/man/man7i/visual_io.7i

```

```

1973 file path=usr/share/man/man7i/vt.7i
1974 file path=usr/xpg4/include/curses.h
1975 file path=usr/xpg4/include/term.h
1976 file path=usr/xpg4/include/unistd.h
1977 legacy pkg=SUNWhea \
1978   desc="SunOS C/C++ header files for general development of software" \
1979   name="SunOS Header Files"
1980 license cr_Sun license=cr_Sun
1981 license lic_CDDL license=lic_CDDL
1982 license license_in_headers license=license_in_headers
1983 license usr/src/lib/pkcs11/include/THIRDPARTYLICENSE \
1984   license=usr/src/lib/pkcs11/include/THIRDPARTYLICENSE
1985 link path=usr/include/iso/assert_iso.h target=./assert.h
1986 link path=usr/include/iso/errno_iso.h target=./errno.h
1987 link path=usr/include/iso/float_iso.h target=./float.h
1988 link path=usr/include/iso/iso646_iso.h target=./iso646.h
1989 $(sparc_ONLY)link path=usr/platform/SUNW,A70/include target=./sun4u/include
1990 $(sparc_ONLY)link path=usr/platform/SUNW,Netra-T12/include \
1991   target=./sun4u/include
1992 $(sparc_ONLY)link path=usr/platform/SUNW,Netra-T4/include \
1993   target=./sun4u/include
1994 $(sparc_ONLY)link path=usr/platform/SUNW,SPARC-Enterprise/include \
1995   target=./sun4u/include
1996 $(sparc_ONLY)link path=usr/platform/SUNW,Serverblad1/include \
1997   target=./sun4u/include
1998 $(sparc_ONLY)link path=usr/platform/SUNW,Sun-Blade-100/include \
1999   target=./sun4u/include
2000 $(sparc_ONLY)link path=usr/platform/SUNW,Sun-Blade-1000/include \
2001   target=./sun4u/include
2002 $(sparc_ONLY)link path=usr/platform/SUNW,Sun-Blade-1500/include \
2003   target=./sun4u/include
2004 $(sparc_ONLY)link path=usr/platform/SUNW,Sun-Blade-2500/include \
2005   target=./sun4u/include
2006 $(sparc_ONLY)link path=usr/platform/SUNW,Sun-Fire-15000/include \
2007   target=./sun4u/include
2008 $(sparc_ONLY)link path=usr/platform/SUNW,Sun-Fire-280R/include \
2009   target=./sun4u/include
2010 $(sparc_ONLY)link path=usr/platform/SUNW,Sun-Fire-480R/include \
2011   target=./sun4u/include
2012 $(sparc_ONLY)link path=usr/platform/SUNW,Sun-Fire-880/include \
2013   target=./sun4u/include
2014 $(sparc_ONLY)link path=usr/platform/SUNW,Sun-Fire-V215/include \
2015   target=./sun4u/include
2016 $(sparc_ONLY)link path=usr/platform/SUNW,Sun-Fire-V240/include \
2017   target=./sun4u/include
2018 $(sparc_ONLY)link path=usr/platform/SUNW,Sun-Fire-V250/include \
2019   target=./sun4u/include
2020 $(sparc_ONLY)link path=usr/platform/SUNW,Sun-Fire-V440/include \
2021   target=./sun4u/include
2022 $(sparc_ONLY)link path=usr/platform/SUNW,Sun-Fire-V445/include \
2023   target=./sun4u/include
2024 $(sparc_ONLY)link path=usr/platform/SUNW,Sun-Fire-V490/include \
2025   target=./sun4u/include
2026 $(sparc_ONLY)link path=usr/platform/SUNW,Sun-Fire-V890/include \
2027   target=./sun4u/include
2028 $(sparc_ONLY)link path=usr/platform/SUNW,Sun-Fire/include \
2029   target=./sun4u/include
2030 $(sparc_ONLY)link path=usr/platform/SUNW,Ultra-2/include \
2031   target=./sun4u/include
2032 $(sparc_ONLY)link path=usr/platform/SUNW,Ultra-250/include \
2033   target=./sun4u/include
2034 $(sparc_ONLY)link path=usr/platform/SUNW,Ultra-4/include \
2035   target=./sun4u/include
2036 $(sparc_ONLY)link path=usr/platform/SUNW,Ultra-Enterprise-10000/include \
2037   target=./sun4u/include
2038 $(sparc_ONLY)link path=usr/platform/SUNW,Ultra-Enterprise/include \

```

```

2039   target=./sun4u/include
2040 $(sparc_ONLY)link path=usr/platform/SUNW,UltraSPARC-IIe-NetraCT-40/include \
2041   target=./sun4u/include
2042 $(sparc_ONLY)link path=usr/platform/SUNW,UltraSPARC-IIe-NetraCT-60/include \
2043   target=./sun4u/include
2044 $(sparc_ONLY)link path=usr/platform/SUNW,UltraSPARC-III-Netract/include \
2045   target=./sun4u/include
2046 link path=usr/share/man/man3head/acct.3head target=acct.h.3head
2047 link path=usr/share/man/man3head/aio.3head target=aio.h.3head
2048 link path=usr/share/man/man3head/ar.3head target=ar.h.3head
2049 link path=usr/share/man/man3head/archives.3head target=archives.h.3head
2050 link path=usr/share/man/man3head/assert.3head target=assert.h.3head
2051 link path=usr/share/man/man3head/complex.3head target=complex.h.3head
2052 link path=usr/share/man/man3head/cpio.3head target=cpio.h.3head
2053 link path=usr/share/man/man3head/direct.3head target=direct.h.3head
2054 link path=usr/share/man/man3head/errno.3head target=errno.h.3head
2055 link path=usr/share/man/man3head/fcntl.3head target=fcntl.h.3head
2056 link path=usr/share/man/man3head/fenv.3head target=fenv.h.3head
2057 link path=usr/share/man/man3head/float.3head target=float.h.3head
2058 link path=usr/share/man/man3head/floatingpoint.3head \
2059   target=floatingpoint.h.3head
2060 link path=usr/share/man/man3head/fmtmsg.3head target=fmtmsg.h.3head
2061 link path=usr/share/man/man3head/fnmatch.3head target=fnmatch.h.3head
2062 link path=usr/share/man/man3head/ftw.3head target=ftw.h.3head
2063 link path=usr/share/man/man3head/glob.3head target=glob.h.3head
2064 link path=usr/share/man/man3head/grp.3head target=grp.h.3head
2065 link path=usr/share/man/man3head/iconv.3head target=iconv.h.3head
2066 link path=usr/share/man/man3head/if.3head target=if.h.3head
2067 link path=usr/share/man/man3head/in.3head target=in.h.3head
2068 link path=usr/share/man/man3head/inet.3head target=inet.h.3head
2069 link path=usr/share/man/man3head/inttypes.3head target=inttypes.h.3head
2070 link path=usr/share/man/man3head/ipc.3head target=ipc.h.3head
2071 link path=usr/share/man/man3head/iso646.3head target=iso646.h.3head
2072 link path=usr/share/man/man3head/langinfo.3head target=langinfo.h.3head
2073 link path=usr/share/man/man3head/libgen.3head target=libgen.h.3head
2074 link path=usr/share/man/man3head/libintl.3head target=libintl.h.3head
2075 link path=usr/share/man/man3head/limits.3head target=limits.h.3head
2076 link path=usr/share/man/man3head/locale.3head target=locale.h.3head
2077 link path=usr/share/man/man3head/math.3head target=math.h.3head
2078 link path=usr/share/man/man3head/mman.3head target=mman.h.3head
2079 link path=usr/share/man/man3head/monetary.3head target=monetary.h.3head
2080 link path=usr/share/man/man3head/mqueue.3head target=mqueue.h.3head
2081 link path=usr/share/man/man3head/msg.3head target=msg.h.3head
2082 link path=usr/share/man/man3head/ndbm.3head target=ndbm.h.3head
2083 link path=usr/share/man/man3head/netdb.3head target=netdb.h.3head
2084 link path=usr/share/man/man3head/nl_types.3head target=nl_types.h.3head
2085 link path=usr/share/man/man3head/poll.3head target=poll.h.3head
2086 link path=usr/share/man/man3head/pthread.3head target=pthread.h.3head
2087 link path=usr/share/man/man3head/pwd.3head target=pwd.h.3head
2088 link path=usr/share/man/man3head/regex.3head target=regex.h.3head
2089 link path=usr/share/man/man3head/resource.3head target=resource.h.3head
2090 link path=usr/share/man/man3head/sched.3head target=sched.h.3head
2091 link path=usr/share/man/man3head/search.3head target=search.h.3head
2092 link path=usr/share/man/man3head/select.3head target=select.h.3head
2093 link path=usr/share/man/man3head/sem.3head target=sem.h.3head
2094 link path=usr/share/man/man3head/semaphore.3head target=semaphore.h.3head
2095 link path=usr/share/man/man3head/setjmp.3head target=setjmp.h.3head
2096 link path=usr/share/man/man3head/shm.3head target=shm.h.3head
2097 link path=usr/share/man/man3head/signinfo.3head target=signinfo.h.3head
2098 link path=usr/share/man/man3head/signal.3head target=signal.h.3head
2099 link path=usr/share/man/man3head/socket.3head target=socket.h.3head
2100 link path=usr/share/man/man3head/spawn.3head target=spawn.h.3head
2101 link path=usr/share/man/man3head/stat.3head target=stat.h.3head
2102 link path=usr/share/man/man3head/statvfs.3head target=statvfs.h.3head
2103 link path=usr/share/man/man3head/stdbool.3head target=stdbool.h.3head
2104 link path=usr/share/man/man3head/stddef.3head target=stddef.h.3head

```

```
2105 link path=usr/share/man/man3head/stdint.3head target=stdint.h.3head
2106 link path=usr/share/man/man3head/stdio.3head target=stdio.h.3head
2107 link path=usr/share/man/man3head/stdlib.3head target=stdlib.h.3head
2108 link path=usr/share/man/man3head/string.3head target=string.h.3head
2109 link path=usr/share/man/man3head/strings.3head target=strings.h.3head
2110 link path=usr/share/man/man3head/stropts.3head target=stropts.h.3head
2111 link path=usr/share/man/man3head/syslog.3head target=syslog.h.3head
2112 link path=usr/share/man/man3head/tar.3head target=tar.h.3head
2113 link path=usr/share/man/man3head/tcp.3head target=tcp.h.3head
2114 link path=usr/share/man/man3head/termios.3head target=termios.h.3head
2115 link path=usr/share/man/man3head/tgmath.3head target=tgmath.h.3head
2116 link path=usr/share/man/man3head/time.3head target=time.h.3head
2117 link path=usr/share/man/man3head/timeb.3head target=timeb.h.3head
2118 link path=usr/share/man/man3head/times.3head target=times.h.3head
2119 link path=usr/share/man/man3head/types.3head target=types.h.3head
2120 link path=usr/share/man/man3head/types32.3head target=types32.h.3head
2121 link path=usr/share/man/man3head/ucontext.3head target=ucontext.h.3head
2122 link path=usr/share/man/man3head/uio.3head target=uio.h.3head
2123 link path=usr/share/man/man3head/ulimit.3head target=ulimit.h.3head
2124 link path=usr/share/man/man3head/un.3head target=un.h.3head
2125 link path=usr/share/man/man3head/unistd.3head target=unistd.h.3head
2126 link path=usr/share/man/man3head/utime.3head target=utime.h.3head
2127 link path=usr/share/man/man3head/utmpx.3head target=utmpx.h.3head
2128 link path=usr/share/man/man3head/utsname.3head target=utsname.h.3head
2129 link path=usr/share/man/man3head/values.3head target=values.h.3head
2130 link path=usr/share/man/man3head/wait.3head target=wait.h.3head
2131 link path=usr/share/man/man3head/wchar.3head target=wchar.h.3head
2132 link path=usr/share/man/man3head/wctype.3head target=wctype.h.3head
2133 link path=usr/share/man/man3head/wordexp.3head target=wordexp.h.3head
2134 link path=usr/share/man/man3head/xlocale.3head target=xlocale.h.3head
2135 $(i386_ONLY)link path=usr/share/src/uts/i86pc/sys \
2136     target=../../../../platform/i86pc/include/sys \
2137 $(i386_ONLY)link path=usr/share/src/uts/i86pc/vm \
2138     target=../../../../platform/i86pc/include/vm \
2139 $(i386_ONLY)link path=usr/share/src/uts/i86pv/sys \
2140     target=../../../../platform/i86pv/include/sys \
2141 $(i386_ONLY)link path=usr/share/src/uts/i86pv/vm \
2142     target=../../../../platform/i86pv/include/vm \
2143 $(sparc_ONLY)link path=usr/share/src/uts/sun4u/sys \
2144     target=../../../../platform/sun4u/include/sys \
2145 $(sparc_ONLY)link path=usr/share/src/uts/sun4u/vm \
2146     target=../../../../platform/sun4u/include/vm \
2147 $(sparc_ONLY)link path=usr/share/src/uts/sun4v/sys \
2148     target=../../../../platform/sun4v/include/sys \
2149 $(sparc_ONLY)link path=usr/share/src/uts/sun4v/vm \
2150     target=../../../../platform/sun4v/include/vm
```

new/usr/src/pkg/manifests/system-library.man3proc.inc

1

```
*****
13704 Thu Jun 30 22:02:59 2016
new/usr/src/pkg/manifests/system-library.man3proc.inc
Code review comments from pmoooney (sundry), and igork (screwups in zonecfg refac)
*****
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
9 # at http://www.illumos.org/license/CDDL.
10 #
11 #
12 #
13 # Copyright 2015 Joyent, Inc.
14 #
15 #
17 file path=usr/share/man/man3proc/Lctlfd.3proc
18 file path=usr/share/man/man3proc/Lfree.3proc
19 file path=usr/share/man/man3proc/Lgrab_error.3proc
20 file path=usr/share/man/man3proc/Lgrab.3proc
21 file path=usr/share/man/man3proc/Lprochandle.3proc
22 file path=usr/share/man/man3proc/Lpsinfo.3proc
23 file path=usr/share/man/man3proc/Lstate.3proc
24 file path=usr/share/man/man3proc/Lstatus.3proc
25 file path=usr/share/man/man3proc/Paddr_to_ctf.3proc
26 file path=usr/share/man/man3proc/Paddr_to_loadobj.3proc
27 file path=usr/share/man/man3proc/Paddr_to_map.3proc
28 file path=usr/share/man/man3proc/Pasfd.3proc
29 file path=usr/share/man/man3proc/Pclearfault.3proc
30 file path=usr/share/man/man3proc/Pclearsig.3proc
31 file path=usr/share/man/man3proc/Pcontent.3proc
32 file path=usr/share/man/man3proc/Pcreate_agent.3proc
33 file path=usr/share/man/man3proc/Pcreate_error.3proc
34 file path=usr/share/man/man3proc/Pcreate.3proc
35 file path=usr/share/man/man3proc/Pcred.3proc
36 file path=usr/share/man/man3proc/Pctlfd.3proc
37 file path=usr/share/man/man3proc/Pdelbkpt.3proc
38 file path=usr/share/man/man3proc/Pdelwapt.3proc
39 file path=usr/share/man/man3proc/Pdestroy_agent.3proc
40 file path=usr/share/man/man3proc/Penv_iter.3proc
41 file path=usr/share/man/man3proc/Perror_printf.3proc
42 file path=usr/share/man/man3proc/Pexecname.3proc
43 file path=usr/share/man/man3proc/Pfault.3proc
44 file path=usr/share/man/man3proc/Pfdinfo_iter.3proc
45 file path=usr/share/man/man3proc/Pgcore.3proc
46 file path=usr/share/man/man3proc/Pgetareg.3proc
47 file path=usr/share/man/man3proc/Pgetauxval.3proc
48 file path=usr/share/man/man3proc/Pgetauxvec.3proc
49 file path=usr/share/man/man3proc/Pgetenv.3proc
50 file path=usr/share/man/man3proc/Pgrab_core.3proc
51 file path=usr/share/man/man3proc/Pgrab_error.3proc
52 file path=usr/share/man/man3proc/Pgrab_file.3proc
53 file path=usr/share/man/man3proc/Pgrab.3proc
54 file path=usr/share/man/man3proc/Pisprocd.3proc
55 file path=usr/share/man/man3proc/Pissyscall.3proc
56 file path=usr/share/man/man3proc/Pltd.3proc
57 file path=usr/share/man/man3proc/Plmid.3proc
58 file path=usr/share/man/man3proc/Plookup_by_addr.3proc
59 file path=usr/share/man/man3proc/Plwp_getasrs.3proc
60 file path=usr/share/man/man3proc/Plwp_getgwindows.3proc
61 file path=usr/share/man/man3proc/Plwp_getpsinfo.3proc
```

new/usr/src/pkg/manifests/system-library.man3proc.inc

2

```
62 file path=usr/share/man/man3proc/Plwp_getregs.3proc
63 file path=usr/share/man/man3proc/Plwp_getspymaster.3proc
64 file path=usr/share/man/man3proc/Plwp_getxregs.3proc
65 file path=usr/share/man/man3proc/Plwp_iter.3proc
66 file path=usr/share/man/man3proc/Plwp_stack.3proc
67 file path=usr/share/man/man3proc/Pmapping_iter.3proc
68 file path=usr/share/man/man3proc/Pobjname.3proc
69 file path=usr/share/man/man3proc/Pplatform.3proc
70 file path=usr/share/man/man3proc/Ppltdest.3proc
71 file path=usr/share/man/man3proc/Ppriv.3proc
72 file path=usr/share/man/man3proc/Ppsinfo.3proc
73 file path=usr/share/man/man3proc/pr_access.3proc
74 file path=usr/share/man/man3proc/pr_close.3proc
75 file path=usr/share/man/man3proc/pr_creat.3proc
76 file path=usr/share/man/man3proc/pr_door_info.3proc
77 file path=usr/share/man/man3proc/pr_exit.3proc
78 file path=usr/share/man/man3proc/pr_fcntl.3proc
79 file path=usr/share/man/man3proc/pr_fstatvfs.3proc
80 file path=usr/share/man/man3proc/pr_getitimer.3proc
81 file path=usr/share/man/man3proc/pr_getpeername.3proc
82 file path=usr/share/man/man3proc/pr_getpeercred.3proc
83 file path=usr/share/man/man3proc/pr_getprojid.3proc
84 file path=usr/share/man/man3proc/pr_getrctl.3proc
85 file path=usr/share/man/man3proc/pr_getrlimit.3proc
86 file path=usr/share/man/man3proc/pr_getsockname.3proc
87 file path=usr/share/man/man3proc/pr_getsockopt.3proc
88 file path=usr/share/man/man3proc/pr_gettaskid.3proc
89 file path=usr/share/man/man3proc/pr_getzoneid.3proc
90 file path=usr/share/man/man3proc/pr_ioctl.3proc
91 file path=usr/share/man/man3proc/pr_link.3proc
92 file path=usr/share/man/man3proc/pr_llseek.3proc
93 file path=usr/share/man/man3proc/pr_lseek.3proc
94 file path=usr/share/man/man3proc/pr_memcntl.3proc
95 file path=usr/share/man/man3proc/pr_meminfo.3proc
96 file path=usr/share/man/man3proc/pr_mmap.3proc
97 file path=usr/share/man/man3proc/pr_munmap.3proc
98 file path=usr/share/man/man3proc/pr_open.3proc
99 file path=usr/share/man/man3proc/pr_processor_bind.3proc
100 file path=usr/share/man/man3proc/pr_rename.3proc
101 file path=usr/share/man/man3proc/pr_setitimer.3proc
102 file path=usr/share/man/man3proc/pr_setrctl.3proc
103 file path=usr/share/man/man3proc/pr_setrlimit.3proc
104 file path=usr/share/man/man3proc/pr_settaskid.3proc
105 file path=usr/share/man/man3proc/pr_sigaction.3proc
106 file path=usr/share/man/man3proc/pr_stat.3proc
107 file path=usr/share/man/man3proc/pr_statvfs.3proc
108 file path=usr/share/man/man3proc/pr_unlink.3proc
109 file path=usr/share/man/man3proc/pr_waitid.3proc
110 file path=usr/share/man/man3proc/Prd_agent.3proc
111 file path=usr/share/man/man3proc/Pread.3proc
112 file path=usr/share/man/man3proc/Prelease.3proc
113 file path=usr/share/man/man3proc/Preopen.3proc
114 file path=usr/share/man/man3proc/Preset_maps.3proc
115 file path=usr/share/man/man3proc/proc_arg_grab.3proc
116 file path=usr/share/man/man3proc/proc_arg_psinfo.3proc
117 file path=usr/share/man/man3proc/proc_content2str.3proc
118 file path=usr/share/man/man3proc/proc_fltname.3proc
119 file path=usr/share/man/man3proc/procfltset2str.3proc
120 file path=usr/share/man/man3proc/proc_get_auxv.3proc
121 file path=usr/share/man/man3proc/proc_get_cred.3proc
122 file path=usr/share/man/man3proc/proc_get_priv.3proc
123 file path=usr/share/man/man3proc/proc_get_psinfo.3proc
124 file path=usr/share/man/man3proc/proc_get_status.3proc
125 file path=usr/share/man/man3proc/proc_initstdio.3proc
126 file path=usr/share/man/man3proc/proc_lwp_in_set.3proc
127 file path=usr/share/man/man3proc/proc_str2flt.3proc
```

```

128 file path=usr/share/man/man3proc/proc_str2fltset.3proc
129 file path=usr/share/man/man3proc/proc_uncctrl_psinfnfo.3proc
130 file path=usr/share/man/man3proc/proc_walk.3proc
131 file path=usr/share/man/man3proc/Psetcflags.3proc
132 #endif /* ! codereview */
133 file path=usr/share/man/man3proc/Psetbkpt.3proc
134 file path=usr/share/man/man3proc/Psetcred.3proc
135 file path=usr/share/man/man3proc/Psetfault.3proc
136 file path=usr/share/man/man3proc/Psetflags.3proc
137 file path=usr/share/man/man3proc/Psetpriv.3proc
138 file path=usr/share/man/man3proc/Psetrun.3proc
139 file path=usr/share/man/man3proc/Psetsignal.3proc
140 file path=usr/share/man/man3proc/Psetsysentry.3proc
141 file path=usr/share/man/man3proc/Psetwapt.3proc
142 file path=usr/share/man/man3proc/Psetzoneid.3proc
143 file path=usr/share/man/man3proc/Psignal.3proc
144 file path=usr/share/man/man3proc/Pstack_iter.3proc
145 file path=usr/share/man/man3proc/Pstate.3proc
146 file path=usr/share/man/man3proc/Pstatus.3proc
147 file path=usr/share/man/man3proc/Pstopstatus.3proc
148 file path=usr/share/man/man3proc/Psymbol_iter.3proc
149 file path=usr/share/man/man3proc/Psync.3proc
150 file path=usr/share/man/man3proc/Psysentry.3proc
151 file path=usr/share/man/man3proc/Puname.3proc
152 file path=usr/share/man/man3proc/Pupdate_maps.3proc
153 file path=usr/share/man/man3proc/Pupdate_syms.3proc
154 file path=usr/share/man/man3proc/Pwrite.3proc
155 file path=usr/share/man/man3proc/Pxecbkpt.3proc
156 file path=usr/share/man/man3proc/Pzonename.3proc
157 link path=usr/share/man/man3proc/Lalt_stack.3proc target=Plwp_stack.3proc
158 link path=usr/share/man/man3proc/Lclearfault.3proc target=Pclearfault.3proc
159 link path=usr/share/man/man3proc/Lclearsig.3proc target=Pclearsig.3proc
160 link path=usr/share/man/man3proc/Ldstop.3proc target=Pstopstatus.3proc
161 link path=usr/share/man/man3proc/Lgetareg.3proc target=Pgetareg.3proc
162 link path=usr/share/man/man3proc/Lmain_stack.3proc target=Plwp_stack.3proc
163 link path=usr/share/man/man3proc/Lputareg.3proc target=Pgetareg.3proc
164 link path=usr/share/man/man3proc/Lsetrun.3proc target=Psetrun.3proc
165 link path=usr/share/man/man3proc/Lstack.3proc target=Plwp_stack.3proc
166 link path=usr/share/man/man3proc/Lstop.3proc target=Pstopstatus.3proc
167 link path=usr/share/man/man3proc/Lsync.3proc target=Psync.3proc
168 link path=usr/share/man/man3proc/Lwait.3proc target=Pstopstatus.3proc
169 link path=usr/share/man/man3proc/Lxecbkpt.3proc target=Pxecbkpt.3proc
170 link path=usr/share/man/man3proc/Lxecwapt.3proc target=Pxecbkpt.3proc
171 link path=usr/share/man/man3proc/Paddr_to_text_map.3proc target=Paddr_to_map.3proc
172 link path=usr/share/man/man3proc/Pcreate_callback.3proc target=Pcreate.3proc
173 link path=usr/share/man/man3proc/Pdstop.3proc target=Pstopstatus.3proc
174 link path=usr/share/man/man3proc/Pfgcore.3proc target=Pgcore.3proc
175 link path=usr/share/man/man3proc/Pfgrab_core.3proc target=Pgrab_core.3proc
176 link path=usr/share/man/man3proc/Pfree.3proc target=Prelease.3proc
177 link path=usr/share/man/man3proc/Pissyscall_prev.3proc target=Pissyscall.3proc
178 link path=usr/share/man/man3proc/Plmid_to_ctf.3proc target=Paddr_to_ctf.3proc
179 link path=usr/share/man/man3proc/Plmid_to_loadobj.3proc target=Paddr_to_loadobj.3proc
180 link path=usr/share/man/man3proc/Plmid_to_map.3proc target=Paddr_to_map.3proc
181 link path=usr/share/man/man3proc/Plookup_by_name.3proc target=Plookup_by_addr.3proc
182 link path=usr/share/man/man3proc/Plwp_alt_stack.3proc target=Plwp_stack.3proc
183 link path=usr/share/man/man3proc/Plwp_getfpregs.3proc target=Plwp_getregs.3proc
184 link path=usr/share/man/man3proc/Plwp_iter_all.3proc target=Plwp_iter.3proc
185 link path=usr/share/man/man3proc/Plwp_main_stack.3proc target=Plwp_stack.3proc
186 link path=usr/share/man/man3proc/Plwp_setasrs.3proc target=Plwp_getasrs.3proc
187 link path=usr/share/man/man3proc/Plwp_setfpregs.3proc target=Plwp_getregs.3proc
188 link path=usr/share/man/man3proc/Plwp_setregs.3proc target=Plwp_getregs.3proc
189 link path=usr/share/man/man3proc/Plwp_setxregs.3proc target=Plwp_getxregs.3proc
190 link path=usr/share/man/man3proc/Pmapping_iter_resolved.3proc target=Pmapping_it
191 link path=usr/share/man/man3proc/Pname_to_ctf.3proc target=Paddr_to_ctf.3proc
192 link path=usr/share/man/man3proc/Pname_to_loadobj.3proc target=Paddr_to_loadobj.3proc
193 link path=usr/share/man/man3proc/Pname_to_map.3proc target=Paddr_to_map.3proc

```

```

194 link path=usr/share/man/man3proc/Pobject_iter_resolved.3proc target=Pmapping_it
195 link path=usr/share/man/man3proc/Pobject_iter.3proc target=Pmapping_iter.3proc
196 link path=usr/share/man/man3proc/Pobjname_resolved.3proc target=Pobjname.3proc
197 link path=usr/share/man/man3proc/Ppriv_free.3proc target=Ppriv.3proc
198 link path=usr/share/man/man3proc/Pputareg.3proc target=Pgetareg.3proc
199 link path=usr/share/man/man3proc/pr_fstat.3proc target=pr_stat.3proc
200 link path=usr/share/man/man3proc/pr_fstat64.3proc target=pr_stat.3proc
201 link path=usr/share/man/man3proc/pr_getrlimit64.3proc target=pr_getrlimit.3proc
202 link path=usr/share/man/man3proc/pr_lstat.3proc target=pr_stat.3proc
203 link path=usr/share/man/man3proc/pr_lstat64.3proc target=pr_stat.3proc
204 link path=usr/share/man/man3proc/pr_setrlimit64.3proc target=pr_setrlimit.3proc
205 link path=usr/share/man/man3proc/pr_stat64.3proc target=pr_stat.3proc
206 link path=usr/share/man/man3proc/Pread_string.3proc target=Pread.3proc
207 link path=usr/share/man/man3proc/proc_arg_xgrab.3proc target=proc_arg_grab.3proc
208 link path=usr/share/man/man3proc/proc_arg_xpsinfo.3proc target=proc_arg_psinfnfo.3
209 link path=usr/share/man/man3proc/proc_finistdio.3proc target=proc_initstdio.3proc
210 link path=usr/share/man/man3proc/proc_flushstdio.3proc target=proc_initstdio.3pr
211 link path=usr/share/man/man3proc/proc_free_priv.3proc target=proc_get_priv.3proc
212 link path=usr/share/man/man3proc/proc_get_ldt.3proc target=Pldt.3proc
213 link path=usr/share/man/man3proc/proc_lwp_range_valid.3proc target=proc_lwp_in_s
214 link path=usr/share/man/man3proc/proc_signame.3proc target=proc_filename.3proc
215 link path=usr/share/man/man3proc/proc_sigset2str.3proc target=proc_ftset2str.3p
216 link path=usr/share/man/man3proc/proc_str2content.3proc target=proc_content2str.
217 link path=usr/share/man/man3proc/proc_str2sig.3proc target=proc_str2flt.3proc
218 link path=usr/share/man/man3proc/proc_str2sigset.3proc target=proc_str2fltset.3p
219 link path=usr/share/man/man3proc/proc_str2sys.3proc target=proc_str2flt.3proc
220 link path=usr/share/man/man3proc/proc_str2sysset.3proc target=proc_str2fltset.3p
221 link path=usr/share/man/man3proc/proc_sysname.3proc target=proc_filename.3proc
222 link path=usr/share/man/man3proc/proc_sysset2str.3proc target=proc_ftset2str.3p
223 link path=usr/share/man/man3proc/Psetsysexit.3proc target=Psetsysentry.3proc
224 link path=usr/share/man/man3proc/Pstop.3proc target=Pstopstatus.3proc
225 link path=usr/share/man/man3proc/Psymbol_iter_by_addr.3proc target=Psymbol_iter.
226 link path=usr/share/man/man3proc/Psymbol_iter_by_lmid.3proc target=Psymbol_iter.
227 link path=usr/share/man/man3proc/Psymbol_iter_by_name.3proc target=Psymbol_iter.
228 link path=usr/share/man/man3proc/Psysexit.3proc target=Psysentry.3proc
229 link path=usr/share/man/man3proc/Punsetflags.3proc target=Psetflags.3proc
230 link path=usr/share/man/man3proc/Pwait.3proc target=Pstopstatus.3proc
231 link path=usr/share/man/man3proc/Pxcreate.3proc target=Pcreate.3proc
232 link path=usr/share/man/man3proc/Pxecwapt.3proc target=Pxecbkpt.3proc
233 link path=usr/share/man/man3proc/Pxlookup_by_addr_resolved.3proc target=Plookup_
234 link path=usr/share/man/man3proc/Pxlookup_by_addr.3proc target=Plookup_by_addr.3
235 link path=usr/share/man/man3proc/Pxlookup_by_name.3proc target=Plookup_by_addr.3
236 link path=usr/share/man/man3proc/Pxsymbol_iter.3proc target=Psymbol_iter.3proc
237 link path=usr/share/man/man3proc/Pzonepath.3proc target=Pzonename.3proc
238 link path=usr/share/man/man3proc/Pzoneroot.3proc target=Pzonename.3proc

```

new/usr/src/pkg/manifests/system-test-ostest.mf

1

```
*****
2481 Thu Jun 30 22:02:59 2016
new/usr/src/pkg/manifests/system-test-ostest.mf
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
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 (c) 2012 by Delphix. All rights reserved.
14 # Copyright 2014, OmniTI Computer Consulting, Inc. All rights reserved.
15 #
17 set name=pkg.fmri value=pkg:/system/test/ostest@$(PKGVERS)
18 set name=pkg.description value="Miscellaneous OS Unit Tests"
19 set name=pkg.summary value="OS Unit Test Suite"
20 set name=info.classification \
21 value=org.opensolaris.category.2008:Development/System
22 set name=variant.arch value=$(ARCH)
23 dir path=opt/os-tests
24 dir path=opt/os-tests/bin
25 dir path=opt/os-tests/runfiles
26 dir path=opt/os-tests/tests
27 dir path=opt/os-tests/tests/secflags
28 #endif /* !codereview */
29 dir path=opt/os-tests/tests/sigqueue
30 file path=opt/os-tests/README mode=0444
31 file path=opt/os-tests/bin/ostest mode=0555
32 file path=opt/os-tests/runfiles/delphix.run mode=0444
33 file path=opt/os-tests/runfiles/omnios.run mode=0444
34 file path=opt/os-tests/runfiles/openindiana.run mode=0444
35 file path=opt/os-tests/tests/poll_test mode=0555
36 file path=opt/os-tests/tests/secflags/addr32 mode=0555
37 file path=opt/os-tests/tests/secflags/addr32 mode=0555
38 file path=opt/os-tests/tests/secflags/secflags_aslr mode=0555
39 file path=opt/os-tests/tests/secflags/secflags_core mode=0555
40 file path=opt/os-tests/tests/secflags/secflags_dts mode=0555
41 file path=opt/os-tests/tests/secflags/secflags_elfdump mode=0555
42 file path=opt/os-tests/tests/secflags/secflags_forbidnullmap mode=0555
43 file path=opt/os-tests/tests/secflags/secflags_limits mode=0555
44 file path=opt/os-tests/tests/secflags/secflags_noexecstack mode=0555
45 file path=opt/os-tests/tests/secflags/secflags_proc mode=0555
46 file path=opt/os-tests/tests/secflags/secflags_psecflags mode=0555
47 file path=opt/os-tests/tests/secflags/secflags_syscall mode=0555
48 file path=opt/os-tests/tests/secflags/secflags_truss mode=0555
49 file path=opt/os-tests/tests/secflags/secflags_zonecfg mode=0555
50 file path=opt/os-tests/tests/secflags/stacky mode=0555
51 #endif /* !codereview */
52 file path=opt/os-tests/tests/sigqueue/sigqueue_queue_size mode=0555
53 file path=opt/os-tests/tests/spoof-ras mode=0555
54 license cr_Sun license=cr_Sun
55 license lic_CDDL license=lic_CDDL
56 depend fmri=system/test/testrunner type=require
```

new/usr/src/test/os-tests/runfiles/delphix.run

1

1058 Thu Jun 30 22:03:00 2016

new/usr/src/test/os-tests/runfiles/delphix.run

7029 want per-process exploit mitigation features (secflags)

7030 want basic address space layout randomization (aslr)

7031 noexec_user_stack should be a secflag

7032 want a means to forbid mappings around NULL.

```
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 (c) 2012 by Delphix. All rights reserved.
14 #
```

```
16 [DEFAULT]
17 pre =
18 verbose = False
19 quiet = False
20 timeout = 60
21 post =
22 outputdir = /var/tmp/test_results
```

```
24 [/opt/os-tests/tests/poll_test]
25 user = root
```

```
27 [/opt/os-tests/tests/secflags]
28 user = root
29 tests = ['secflags_aslr',
30          'secflags_core',
31          'secflags_dts',
32          'secflags_elfdump',
33          'secflags_forbidnullmap',
34          'secflags_limits',
35          'secflags_noexecstack',
36          'secflags_proc',
37          'secflags_psecflags',
38          'secflags_syscall',
39          'secflags_truss',
40          'secflags_zonecfg']
```

```
42 #endif /* ! codereview */
43 [/opt/os-tests/tests/sigqueue]
44 tests = ['sigqueue_queue_size']
```


new/usr/src/test/os-tests/runfiles/omnios.run

1

1058 Thu Jun 30 22:03:01 2016

new/usr/src/test/os-tests/runfiles/omnios.run

7029 want per-process exploit mitigation features (secflags)

7030 want basic address space layout randomization (aslr)

7031 noexec_user_stack should be a secflag

7032 want a means to forbid mappings around NULL.

```
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 (c) 2012 by Delphix. All rights reserved.
14 #
```

```
16 [DEFAULT]
17 pre =
18 verbose = False
19 quiet = False
20 timeout = 60
21 post =
22 outputdir = /var/tmp/test_results
```

```
24 [/opt/os-tests/tests/poll_test]
25 user = root
```

```
27 [/opt/os-tests/tests/secflags]
28 user = root
29 tests = ['secflags_aslr',
30          'secflags_core',
31          'secflags_dts',
32          'secflags_elfdump',
33          'secflags_forbidnullmap',
34          'secflags_limits',
35          'secflags_noexecstack',
36          'secflags_proc',
37          'secflags_psecflags',
38          'secflags_syscall',
39          'secflags_truss',
40          'secflags_zonecfg']
```

```
42 #endif /* ! codereview */
43 [/opt/os-tests/tests/sigqueue]
44 tests = ['sigqueue_queue_size']
```

new/usr/src/test/os-tests/runfiles/openindiana.run

1

1059 Thu Jun 30 22:03:02 2016

new/usr/src/test/os-tests/runfiles/openindiana.run

7029 want per-process exploit mitigation features (secflags)

7030 want basic address space layout randomization (aslr)

7031 noexec_user_stack should be a secflag

7032 want a means to forbid mappings around NULL.

```
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 (c) 2012 by Delphix. All rights reserved.
14 #
```

```
16 [DEFAULT]
17 pre =
18 verbose = False
19 quiet = False
20 timeout = 60
21 post =
22 outputdir = /var/tmp/test_results
```

```
24 [/opt/os-tests/tests/poll_test]
25 user = root
```

```
27 [/opt/os-tests/tests/secflags]
28 user = root
29 tests = ['secflags_aslr',
30          'secflags_core',
31          'secflags_dts',
32          'secflags_elfdump',
33          'secflags_forbidnullmap',
34          'secflags_limits',
35          'secflags_noexecstack',
36          'secflags_proc',
37          'secflags_psecflags',
38          'secflags_syscall',
39          'secflags_truss',
40          'secflags_zonecfg']
```

```
42 #endif /* ! codereview */
43 [/opt/os-tests/tests/sigqueue]
44 tests = ['sigqueue_queue_size']
```

new/usr/src/test/os-tests/tests/Makefile

1

539 Thu Jun 30 22:03:02 2016

new/usr/src/test/os-tests/tests/Makefile

7029 want per-process exploit mitigation features (secflags)

7030 want basic address space layout randomization (aslr)

7031 noexec_user_stack should be a secflag

7032 want a means to forbid mappings around NULL.

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 (c) 2012 by Delphix. All rights reserved.

14 #

16 SUBDIRS = poll secflags sigqueue spoof-ras

16 SUBDIRS = poll sigqueue spoof-ras

18 include \$(SRC)/test/Makefile.com

```

*****
1454 Thu Jun 30 22:03:03 2016
new/usr/src/test/os-tests/tests/secflags/Makefile
Code review comments from jeffpc
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
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 # Copyright 2015, Richard Lowe.

15 include $(SRC)/cmd/Makefile.cmd
16 include $(SRC)/test/Makefile.com

18 PROG = secflags_aslr      \
19         secflags_core     \
20         secflags_dts      \
21         secflags_elfdump  \
22         secflags_forbidnullmap \
23         secflags_limits   \
24         secflags_noexecstack \
25         secflags_proc     \
26         secflags_psecflags \
27         secflags_syscall  \
28         secflags_truss    \
29         secflags_zonecfg

31 PROG += addr32 addr64 stacky

33 ROOTOPTPKG = $(ROOT)/opt/os-tests
34 TESTDIR = $(ROOTOPTPKG)/tests/secflags

36 CMDS = $(PROG:%=$(TESTDIR)/%)
37 $(CMDS) := FILEMODE = 0555

39 addr32: addr.c
40         $(LINK.c) addr.c -o $@ $(LDLIBS)
41         $(POST_PROCESS)

43 addr64: addr.c
44         $(LINK64.c) addr.c -o $@ $(LDLIBS)
45         $(POST_PROCESS)

47 stacky := MAPFILE.NES=          # Will foil the test, clearly
48 stacky: stacky.o
49         $(LINK.c) stacky.o -o $@ $(LDLIBS)
50         $(POST_PROCESS)

52 secflags_syscall: secflags_syscall.c
53         $(LINK.c) secflags_syscall.c -o $@ $(LDLIBS)
54         $(POST_PROCESS)

56 all: $(PROG)

```

```

58 install: all $(CMDS)

60 lint:

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

65 clean:

67 $(CMDS): $(TESTDIR) $(PROG)

69 $(TESTDIR):
70         $(INS.dir)

72 $(TESTDIR)/%: %
73         $(INS.file)
74 #endif /* !codereview */

```

new/usr/src/test/os-tests/tests/secflags/addr.c

1

```
*****  
537 Thu Jun 30 22:03:04 2016  
new/usr/src/test/os-tests/tests/secflags/addr.c  
7029 want per-process exploit mitigation features (secflags)  
7030 want basic address space layout randomization (aslr)  
7031 noexec_user_stack should be a secflag  
7032 want a means to forbid mappings around NULL.  
*****
```

```
1 #include <sys/mman.h>  
  
3 #include <stdlib.h>  
4 #include <unistd.h>  
5 #include <err.h>  
  
7 int  
8 main(int argc, char **argv)  
9 {  
10     int stack = 0;  
11     void *heap = NULL;  
12     void *mapping = NULL;  
  
14     if ((heap = malloc(10)) == NULL)  
15         err(1, "couldn't allocate");  
  
17     if ((mapping = mmap((caddr_t)0, 10, (PROT_READ | PROT_WRITE),  
18         MAP_ANON|MAP_PRIVATE, -1, 0)) == (void*)-1)  
19         err(1, "couldn't map");  
  
21     printf("  stack: 0x%p\n", &stack);  
22     printf("  heap: 0x%p\n", heap);  
23     printf("mapping: 0x%p\n", mapping);  
24     printf("  text: 0x%p\n", &main);  
25     return (0);  
26 }  
27 #endif /* ! codereview */
```

```
new/usr/src/test/os-tests/tests/secflags/secflags_aslr.sh
```

1

```
*****
```

```
1381 Thu Jun 30 22:03:04 2016
```

```
new/usr/src/test/os-tests/tests/secflags/secflags_aslr.sh
```

```
Code review comments from jeffpc
```

```
7029 want per-process exploit mitigation features (secflags)
```

```
7030 want basic address space layout randomization (aslr)
```

```
7031 noexec_user_stack should be a secflag
```

```
7032 want a means to forbid mappings around NULL.
```

```
*****
```

```
1 #! /usr/bin/ksh
```

```
2 #
```

```
3 #
```

```
4 # This file and its contents are supplied under the terms of the
```

```
5 # Common Development and Distribution License ("CDDL"), version 1.0.
```

```
6 # You may only use this file in accordance with the terms of version
```

```
7 # 1.0 of the CDDL.
```

```
8 #
```

```
9 # A full copy of the text of the CDDL should have accompanied this
```

```
10 # source. A copy of the CDDL is also available via the Internet at
```

```
11 # http://www.illumos.org/license/CDDL.
```

```
12 #
```

```
14 # Copyright 2015, Richard Lowe.
```

```
16 # Verify that aslr messes things up, by comparing the mappings of 2 identical
```

```
17 # processes
```

```
19 LC_ALL=C # Collation is important
```

```
21 /usr/bin/psecflags -s aslr $$
```

```
23 tmpdir=/tmp/test.$$
```

```
25 mkdir $tmpdir
```

```
26 cd $tmpdir
```

```
28 cleanup() {
```

```
29     cd /
```

```
30     rm -fr $tmpdir
```

```
31 }
```

```
33 trap 'cleanup' EXIT
```

```
35 check() {
```

```
36     typeset name=$1
```

```
37     typeset command=$2
```

```
38
```

```
39     for (( i=0; i < 1000; i++ )); do
```

```
40         $command > out.$i
```

```
41     done
```

```
43     cat out.* | sort | uniq -c | sort -nk 1 | nawk '
```

```
44     BEGIN {
```

```
45         tot = 0
```

```
46         colls = 0
```

```
47     }
```

```
49     $2 != "text:" {
```

```
50         tot += $1
```

```
51         if ($1 > 1) {
```

```
52             colls += $1
```

```
53         }
```

```
54     }
```

```
56     END {
```

```
57         prc = (colls / tot) * 100
```

```
new/usr/src/test/os-tests/tests/secflags/secflags_aslr.sh
```

2

```
58     printf "$name Collisions: %d/%d (%g%)\n", colls, tot, prc
```

```
59     exit prc
```

```
60 }
```

```
61 '
```

```
62     return $?
```

```
63 }
```

```
65 # Somewhat arbitrary
```

```
66 ACCEPTABLE=70
```

```
68 ret=0
```

```
69 check 32bit /opt/os-tests/tests/secflags/addrs-32
```

```
70 (( $? > $ACCEPTABLE )) && ret=1
```

```
71 check 64bit /opt/os-tests/tests/secflags/addrs-64
```

```
72 (( $? > $ACCEPTABLE )) && ret=1
```

```
74 exit $ret
```

```
75 #endif /* !codereview */
```

new/usr/src/test/os-tests/tests/secflags/secflags_core.sh

1

```
1275 Thu Jun 30 22:03:04 2016
new/usr/src/test/os-tests/tests/secflags/secflags_core.sh
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
```

```
1 #!/usr/bin/ksh
2 #
3 #
4 # This file and its contents are supplied under the terms of the
5 # Common Development and Distribution License ("CDDL"), version 1.0.
6 # You may only use this file in accordance with the terms of version
7 # 1.0 of the CDDL.
8 #
9 # A full copy of the text of the CDDL should have accompanied this
10 # source. A copy of the CDDL is also available via the Internet at
11 # http://www.illumos.org/license/CDDL.
12 #
13 #
14 #
15 # Copyright 2015, Richard Lowe.
16 #
17 #
18 mkdir /tmp/secflags-test.$$
19 cd /tmp/secflags-test.$$
20 #
21 /usr/bin/psecflags -s aslr -e sleep 100000 &
22 pid=$!
23 coreadm -p core $pid # We need to be able to reliably find the core
24 #
25 cleanup() {
26     kill $pid >/dev/null 2>&1
27     cd /
28     rm -fr /tmp/secflags-test.$$
29 }
30 #
31 trap cleanup EXIT
32 #
33 ## gcore-produced core
34 gcore $pid >/dev/null
35 #
36 cat > gcore-expected.$$ <<EOF
37 core 'core.$pid' of $pid:      sleep 100000
38     E:      aslr
39     I:      aslr
40 EOF
41 #
42 /usr/bin/psecflags core.${pid} | grep -v '[LU]:' > gcore-output.$$
43 #
44 if ! diff -u gcore-expected.$$ gcore-output.$$; then
45     exit 1;
46 fi
47 #
48 ## kernel-produced core
49 kill -SEGV $pid
50 #
51 cat > core-expected.$$ <<EOF
52 core 'core' of $pid:      sleep 100000
53     E:      aslr
54     I:      aslr
55 EOF
56 #
57 /usr/bin/psecflags core | grep -v '[LU]:' > core-output.$$
```

new/usr/src/test/os-tests/tests/secflags/secflags_core.sh

2

```
59 if ! diff -u core-expected.$$ core-output.$$; then
60     exit 1;
61 fi
62 #
63 exit 0
64 #endif /* ! codereview */
```

```
new/usr/src/test/os-tests/tests/secflags/secflags_dts.sh
```

1

```
*****
1618 Thu Jun 30 22:03:05 2016
new/usr/src/test/os-tests/tests/secflags/secflags_dts.sh
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
1 #!/usr/bin/ksh
2 #
3 #
4 # This file and its contents are supplied under the terms of the
5 # Common Development and Distribution License ("CDDL"), version 1.0.
6 # You may only use this file in accordance with the terms of version
7 # 1.0 of the CDDL.
8 #
9 # A full copy of the text of the CDDL should have accompanied this
10 # source. A copy of the CDDL is also available via the Internet at
11 # http://www.illumos.org/license/CDDL.
12 #
13 #
14 # Copyright 2015, Richard Lowe.
15 #
16 tmpdir=/tmp/test.$$
17 mkdir $tmpdir
18 cd $tmpdir
19 #
20 cleanup() {
21     cd /
22     rm -fr $tmpdir
23 }
24 #
25 trap 'cleanup' EXIT
26 #
27 cat > tester.c <<EOF
28 #include <stdio.h>
29 #include <unistd.h>
30 #
31 int
32 main(int argc, char **argv)
33 {
34     sleep(10000);
35     return (0);
36 }
37 EOF
38 #
39 gcc -o tester-aslr tester.c -Wl,-z,aslr=enabled
40 gcc -o tester-noaslr tester.c -Wl,-z,aslr=disabled
41 #
42 # This is the easiest way I've found to get many many DTs, but it's gross
43 gcc -o many-dts-aslr tester.c -Wl,-z,aslr=enabled $(for elt in /usr/lib/lib*.so;
44 gcc -o many-dts-noaslr tester.c -Wl,-z,aslr=disabled $(for elt in /usr/lib/lib*.
45 #
46 check() {
47     bin=$1
48     state=$2
49     ret=0
50 #
51     $bin &
52     pid=$!
53     psecflags $pid | grep -q 'E:.*aslr'
54     (( $? != $state )) && ret=1
55     kill -9 $pid
56     return $ret
57 }
```

```
new/usr/src/test/os-tests/tests/secflags/secflags_dts.sh
```

2

```
59 fail() {
60     echo $@
61     exit 1
62 }
63 #
64 psecflags -s none $$
65 check ./tester-aslr 0 || fail "DT_SUNW_ASLR 1 failed"
66 check ./many-dts-aslr 0 || fail "DT_SUNW_ASLR 1 with many DTs failed"
67 #
68 psecflags -s aslr $$
69 check ./tester-noaslr 1 || fail "DT_SUNW_ASLR 0 failed"
70 check ./many-dts-noaslr 1 || fail "DT_SUNW_ASLR 0 with many DTs failed"
71 #
72 #endif /* ! codereview */
```



```
new/usr/src/test/os-tests/tests/secflags/secflags_elfdump.sh
```

1

```
*****
```

```
1754 Thu Jun 30 22:03:05 2016
new/usr/src/test/os-tests/tests/secflags/secflags_elfdump.sh
Code review comments from jeffpc
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
```

```
*****
```

```
1 #! /usr/bin/ksh
2 #
3 #
4 # This file and its contents are supplied under the terms of the
5 # Common Development and Distribution License ("CDDL"), version 1.0.
6 # You may only use this file in accordance with the terms of version
7 # 1.0 of the CDDL.
8 #
9 # A full copy of the text of the CDDL should have accompanied this
10 # source. A copy of the CDDL is also available via the Internet at
11 # http://www.illumos.org/license/CDDL.
12 #
13 #
14 #
15 # Copyright 2015, Richard Lowe.
16 #
17 #
18 mkdir /tmp/secflags-test.$$
19 cd /tmp/secflags-test.$$
20 #
21 /usr/bin/prsecflags -s aslr -e sleep 100000 &
22 pid=$!
23 coreadm -p core $pid # We need to be able to reliably find the core
24 #
25 cleanup() {
26     kill $pid >/dev/null 2>&1
27     cd /
28     rm -fr /tmp/secflags-test.$$
29 }
30 #
31 trap cleanup EXIT
32 #
33 ## gcore-produced core
34 gcore $pid >/dev/null
35 #
36 cat > gcore-expected.$$ <<EOF
37 namesz: 0x5
38 descsz: 0x28
39 type: [ NT_SECFLAGS ]
40 name:
41     CORE\0
42 desc: (prsecflags_t)
43     pr_version: 1
44     pr_effective: [ ASLR ]
45     pr_inherit: [ ASLR ]
46     pr_lower: 0
47     pr_upper: [ ASLR FORBIDNULLMAP NOEXECSTACK ]
48 EOF
49 #
50 /usr/bin/elfdump -n core.${pid} | grep -B5 -A5 prsecflags_t > gcore-output.$$
51 #
52 if ! diff -u gcore-expected.$$ gcore-output.$$; then
53     exit 1;
54 fi
55 #
56 ## kernel-produced core
57 kill -SEGV $pid
```

```
new/usr/src/test/os-tests/tests/secflags/secflags_elfdump.sh
```

2

```
59 cat > core-expected.$$ <<EOF
60 namesz: 0x5
61 descsz: 0x28
62 type: [ NT_SECFLAGS ]
63 name:
64     CORE\0
65 desc: (prsecflags_t)
66     pr_version: 1
67     pr_effective: [ ASLR ]
68     pr_inherit: [ ASLR ]
69     pr_lower: 0
70     pr_upper: [ ASLR FORBIDNULLMAP NOEXECSTACK ]
71 EOF
72 #
73 /usr/bin/elfdump -n core | grep -B5 -A5 prsecflags_t > core-output.$$
74 #
75 if ! diff -u core-expected.$$ core-output.$$; then
76     exit 1;
77 fi
78 #
79 exit 0
80 #endif /* ! codereview */
```

```
new/usr/src/test/os-tests/tests/secflags/secflags_forbidnullmap.sh
```

1

```
*****
```

```
620 Thu Jun 30 22:03:06 2016
```

```
new/usr/src/test/os-tests/tests/secflags/secflags_forbidnullmap.sh
```

```
7029 want per-process exploit mitigation features (secflags)
```

```
7030 want basic address space layout randomization (aslr)
```

```
7031 noexec_user_stack should be a secflag
```

```
7032 want a means to forbid mappings around NULL.
```

```
*****
```

```
1 #! /usr/bin/ksh
```

```
2 #
```

```
3 #
```

```
4 # This file and its contents are supplied under the terms of the
```

```
5 # Common Development and Distribution License ("CDDL"), version 1.0.
```

```
6 # You may only use this file in accordance with the terms of version
```

```
7 # 1.0 of the CDDL.
```

```
8 #
```

```
9 # A full copy of the text of the CDDL should have accompanied this
```

```
10 # source. A copy of the CDDL is also available via the Internet at
```

```
11 # http://www.illumos.org/license/CDDL.
```

```
12 #
```

```
14 # Copyright 2015, Richard Lowe.
```

```
16 /usr/bin/psecflags -s forbidnullmap $$
```

```
18 LD_PRELOAD=@0.so.1 /usr/bin/sleep 100000 &
```

```
19 pid=$!
```

```
21 ret=0
```

```
22 (pmap $pid | grep -q '^00000000 ') && ret=1
```

```
23 kill -9 $pid
```

```
25 exit $ret
```

```
26 #endif /* ! codereview */
```

new/usr/src/test/os-tests/tests/secflags/secflags_limits.sh

1

1490 Thu Jun 30 22:03:06 2016

new/usr/src/test/os-tests/tests/secflags/secflags_limits.sh

7029 want per-process exploit mitigation features (secflags)

7030 want basic address space layout randomization (aslr)

7031 noexec_user_stack should be a secflag

7032 want a means to forbid mappings around NULL.

1 #!/usr/bin/ksh

2 #

3 #

4 # This file and its contents are supplied under the terms of the

5 # Common Development and Distribution License ("CDDL"), version 1.0.

6 # You may only use this file in accordance with the terms of version

7 # 1.0 of the CDDL.

8 #

9 # A full copy of the text of the CDDL should have accompanied this

10 # source. A copy of the CDDL is also available via the Internet at

11 # <http://www.illumos.org/license/CDDL>.

12 #

14 #

15 # Copyright 2015, Richard Lowe.

16 #

18 mkdir /tmp/secflags-test.\$\$

19 cd /tmp/secflags-test.\$\$

21 cleanup() {

22 kill \$pid >/dev/null 2>&1

23 cd /

24 rm -fr /tmp/secflags-test.\$\$

25 }

27 trap cleanup EXIT

29 # Check that lower implies setting of inheritable

30 echo "Setting lower also adds to inheritable"

31 /usr/bin/psecflags -s L=aslr \$\$

33 cat > expected <<EOF

34 I: aslr

35 EOF

36 /usr/bin/psecflags \$\$ | grep 'I:' > output

38 diff -u expected output || exit 1

40 echo "Setting in lower cannot be removed from inheritable"

41 /usr/bin/psecflags -s I=current,-aslr \$\$ 2>/dev/null && exit 1

43 echo "Setting in lower cannot be removed"

44 /usr/bin/psecflags -s L=current,-aslr \$\$ 2>/dev/null && exit 1

47 echo "Setting in lower cannot be removed from upper"

48 /usr/bin/psecflags -s U=current,-aslr \$\$ 2>/dev/null && exit 1

50 /usr/bin/psecflags -s U=current,-noexecstack \$\$

52 echo "Setting in default cannot exceed upper"

53 /usr/bin/psecflags -s I=noexecstack \$\$ 2>/dev/null && exit 1

55 echo "Setting cannot ever be added to upper"

56 /usr/bin/psecflags -s U=current,+noexecstack \$\$ 2>/dev/null && exit 1

58 exit 0

new/usr/src/test/os-tests/tests/secflags/secflags_limits.sh

2

61 #endif /* ! codereview */

new/usr/src/test/os-tests/tests/secflags/secflags_noexecstack.sh

1

618 Thu Jun 30 22:03:06 2016

new/usr/src/test/os-tests/tests/secflags/secflags_noexecstack.sh

7029 want per-process exploit mitigation features (secflags)

7030 want basic address space layout randomization (aslr)

7031 noexec_user_stack should be a secflag

7032 want a means to forbid mappings around NULL.

```
1 #!/usr/bin/ksh
2 #
3 #
4 # This file and its contents are supplied under the terms of the
5 # Common Development and Distribution License ("CDDL"), version 1.0.
6 # You may only use this file in accordance with the terms of version
7 # 1.0 of the CDDL.
8 #
9 # A full copy of the text of the CDDL should have accompanied this
10 # source. A copy of the CDDL is also available via the Internet at
11 # http://www.illumos.org/license/CDDL.
12 #
13 #
14 # Copyright 2015, Richard Lowe.
15 #
16 /usr/bin/psecflags -s noexecstack $$
17 #
18 /opt/os-tests/tests/secflags/stacky &
19 pid=$!
20 #
21 ret=0
22 (pmap $pid | grep -q 'rwx.*\[ stack \]$') && re=1
23 #
24 kill -9 $pid
25 exit $ret
26 #endif /* ! codereview */
```

new/usr/src/test/os-tests/tests/secflags/secflags_proc.sh

1

810 Thu Jun 30 22:03:07 2016

new/usr/src/test/os-tests/tests/secflags/secflags_proc.sh

7029 want per-process exploit mitigation features (secflags)

7030 want basic address space layout randomization (aslr)

7031 noexec_user_stack should be a secflag

7032 want a means to forbid mappings around NULL.

```
1 #!/usr/bin/ksh
2 #
3 #
4 # This file and its contents are supplied under the terms of the
5 # Common Development and Distribution License ("CDDL"), version 1.0.
6 # You may only use this file in accordance with the terms of version
7 # 1.0 of the CDDL.
8 #
9 # A full copy of the text of the CDDL should have accompanied this
10 # source. A copy of the CDDL is also available via the Internet at
11 # http://www.illumos.org/license/CDDL.
12 #
13 #
14 #
15 # Copyright 2015, Richard Lowe.
16 #
17 #
18 /usr/bin/psecflags -s aslr -e sleep 100000 &
19 pid=$!
20 #
21 cleanup() {
22     kill $pid
23     rm /tmp/output.$$
24     rm /tmp/expected.$$
25 }
26 #
27 trap cleanup EXIT
28 #
29 cat > /tmp/expected.$$ <<EOF
30 $pid: sleep 100000
31 E: aslr
32 I: aslr
33 EOF
34 #
35 /usr/bin/psecflags $pid | grep -v '[LU]:' > /tmp/output.$$
36 #
37 if ! diff -u /tmp/expected.$$ /tmp/output.$$; then
38     exit 1;
39 fi
40 #
41 exit 0
42 #endif /* ! codereview */
```

new/usr/src/test/os-tests/tests/secflags/secflags_psecflags.sh

1

```
*****
3711 Thu Jun 30 22:03:07 2016
new/usr/src/test/os-tests/tests/secflags/secflags_psecflags.sh
Code review comments from jeffpc
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
1 #! /usr/bin/ksh
2 #
3 #
4 # This file and its contents are supplied under the terms of the
5 # Common Development and Distribution License ("CDDL"), version 1.0.
6 # You may only use this file in accordance with the terms of version
7 # 1.0 of the CDDL.
8 #
9 # A full copy of the text of the CDDL should have accompanied this
10 # source. A copy of the CDDL is also available via the Internet at
11 # http://www.illumos.org/license/CDDL.
12 #
13 #
14 #
15 # Copyright 2015, Richard Lowe.
16 #
17 #
18 mkdir /tmp/$$-secflags-test
19 cd /tmp/$$-secflags-test
20 #
21 /usr/bin/psecflags -s none $$ # Clear ourselves out
22 cat > expected <<EOF
23 I: none
24 EOF
25 #
26 /usr/bin/psecflags $$ | grep I: > output
27 diff -u expected output || exit 1 # Make sure the setting of 'none' worked
28 #
29 cleanup() {
30 cd /
31 rm -fr /tmp/$$-secflags-test
32 }
33 trap cleanup EXIT
34 #
35 ## Tests of manipulating a running process (ourselves)
36 #
37 self_set() {
38 echo "Set (self)"
39 /usr/bin/psecflags -s aslr $$
40 #
41 cat > expected <<EOF
42 I: aslr
43 EOF
44 #
45 /usr/bin/psecflags $$ | grep I: > output
46 diff -u expected output || exit 1
47 }
48 #
49 self_add() {
50 echo "Add (self)"
51 /usr/bin/psecflags -s current,noexecstack $$
52 cat > expected <<EOF
53 I: aslr,noexecstack
54 EOF
55 #
56 /usr/bin/psecflags $$ | grep I: > output
57 diff -u expected output || exit 1
```

new/usr/src/test/os-tests/tests/secflags/secflags_psecflags.sh

2

```
58 }
59 #
60 self_remove() {
61 echo "Remove (self)"
62 /usr/bin/psecflags -s current,-aslr $$
63 cat > expected <<EOF
64 I: noexecstack
65 EOF
66 #
67 /usr/bin/psecflags $$ | grep I: > output
68 diff -u expected output || exit 1
69 }
70 #
71 self_all() {
72 echo "All (self)"
73 /usr/bin/psecflags -s all $$
74 /usr/bin/psecflags $$ | grep -q 'I:.*,.*, ' || exit 1 # This is lame, but fun
75 }
76 #
77 self_none() {
78 echo "None (self)"
79 /usr/bin/psecflags -s all $$
80 /usr/bin/psecflags -s none $$
81 cat > expected <<EOF
82 I: none
83 EOF
84 /usr/bin/psecflags $$ | grep I: > output
85 diff -u expected output || exit 1
86 }
87 #
88 child_set() {
89 echo "Set (child)"
90 #
91 typeset pid;
92 #
93 /usr/bin/psecflags -s aslr -e sleep 10000 &
94 pid=$!
95 cat > expected <<EOF
96 E: aslr
97 I: aslr
98 EOF
99 /usr/bin/psecflags $pid | grep '[IE:]' > output
100 kill $pid
101 diff -u expected output || exit 1
102 }
103 #
104 child_add() {
105 echo "Add (child)"
106 #
107 typeset pid;
108 #
109 /usr/bin/psecflags -s aslr $$
110 /usr/bin/psecflags -s current,noexecstack -e sleep 10000 &
111 pid=$!
112 cat > expected <<EOF
113 E: aslr,noexecstack
114 I: aslr,noexecstack
115 EOF
116 /usr/bin/psecflags $pid | grep '[IE:]' > output
117 kill $pid
118 /usr/bin/psecflags -s none $$
119 diff -u expected output || exit 1
120 }
121 #
122 child_remove() {
123 echo "Remove (child)"
```

```

125 typeset pid;

127 /usr/bin/psecflags -s aslr $$
128 /usr/bin/psecflags -s current,-aslr -e sleep 10000 &
129 pid=$!
130 cat > expected <<EOF
131     E:     none
132     I:     none
133 EOF
134 /usr/bin/psecflags $pid | grep '[IE:]' > output
135 kill $pid
136 /usr/bin/psecflags -s none $$
137 diff -u expected output || exit 1
138 }

140 child_all() {
141     echo "All (child)"

143     typeset pid ret

145     /usr/bin/psecflags -s all -e sleep 10000 &
146     pid=$!
147     /usr/bin/psecflags $pid | grep -q 'E:.*,*' # This is lame, but functional
148     ret=$?
149     kill $pid
150     (( $ret != 0 )) && exit $ret
151 }

153 child_none() {
154     echo "None (child)"

156     typeset pid

157     /usr/bin/psecflags -s all $$

160     /usr/bin/psecflags -s none -e sleep 10000 &
161     pid=$!
162     cat > expected <<EOF
163         E:     none
164         I:     none
165 EOF
166 /usr/bin/psecflags $pid | grep '[IE:]' > output
167 kill $pid
168 diff -u expected output || exit 1
169 }

171 list() {
172     echo "List"
173     cat > expected<<EOF
174 aslr
175 forbidnullmap
176 noexecstack
177 EOF

179     /usr/bin/psecflags -l > output
180     diff -u expected output || exit 1
181 }

183 self_set
184 self_add
185 self_remove
186 self_all
187 self_none
188 child_set
189 child_add

```

```

190 child_remove
191 child_all
192 child_none
193 list

195 exit 0
196 #endif /* ! codereview */

```

```

*****
1878 Thu Jun 30 22:03:08 2016
new/usr/src/test/os-tests/tests/secflags/secflags_syscall.c
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
1 #include <stdio.h>
2 #include <err.h>
3 #include <errno.h>

5 #include <sys/secflags.h>
6 #include <sys/syscall.h>

8 int
9 main(int argc, char **argv)
10 {
11     int err = 0;
12     secflagdelta_t act = {0};

14     if ((err = syscall(SYS_psecflags, NULL, PSF_INHERIT, NULL)) != 0) {
15         if (errno != EFAULT)
16             warnx("attempt to set secflags with a NULL procset "
17                 "set errno other than EFAULT (%d)", errno);
18     } else {
19         warnx("attempt to set secflags with a NULL procset succeeded");
20     }

22     if ((err = syscall(SYS_psecflags, (void*)0xdeadbeef,
23         PSF_INHERIT, NULL)) != 0) {
24         if (errno != EFAULT)
25             warnx("attempt to set secflags with a bad procset "
26                 "set errno other than EFAULT (%d)", errno);
27     } else {
28         warnx("attempt to set secflags with a bad procset succeeded");
29     }

32     if ((err = psecflags(P_PID, P_MYID, PSF_INHERIT, NULL)) != 0) {
33         if (errno != EFAULT)
34             warnx("attempt to set secflags with a NULL "
35                 "delta set errno to other than EFAULT (%d)",
36                 errno);
37     } else {
38         warnx("attempt to set secflags with a NULL delta succeeded");
39     }

41     if ((err = psecflags(P_PID, P_MYID, PSF_INHERIT,
42         (void*)0xdeadbeef)) != 0) {
43         if (errno != EFAULT)
44             warnx("attempt to set secflags with a bad "
45                 "delta set errno to other than EFAULT (%d)",
46                 errno);
47     } else {
48         warnx("attempt to set secflags with a bad delta succeeded");
49     }

51     if ((err = psecflags(P_LWPID, P_MYID, PSF_INHERIT, &act)) != 0) {
52         if (errno != EINVAL)
53             warnx("attempt to set secflags of an lwpid set errno "
54                 "to other than EINVAL (%d)", errno);
55     } else {
56         warnx("attempt to set secflags of an lwpid succeeded");
57     }

```

```

59     if ((err = psecflags(P_LWPID, P_MYID, PSF_EFFECTIVE, &act)) != 0) {
60         if (errno != EINVAL)
61             warnx("attempt to set effective secflags set errno "
62                 "to other than EINVAL (%d)", errno);
63     } else {
64         warnx("attempt to set effective secflags succeeded");
65     }

67     return (0);
68 }
69 #endif /* ! codereview */

```


new/usr/src/test/os-tests/tests/secflags/secflags_truss.sh

1

```
*****
967 Thu Jun 30 22:03:08 2016
new/usr/src/test/os-tests/tests/secflags/secflags_truss.sh
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
1  #! /usr/bin/ksh
2  #
3  #
4  # This file and its contents are supplied under the terms of the
5  # Common Development and Distribution License ("CDDL"), version 1.0.
6  # You may only use this file in accordance with the terms of version
7  # 1.0 of the CDDL.
8  #
9  # A full copy of the text of the CDDL should have accompanied this
10 # source. A copy of the CDDL is also available via the Internet at
11 # http://www.illumos.org/license/CDDL.
12 #
13 #
14 #
15 # Copyright 2015, Richard Lowe.
16 #
17 #
18 cd /tmp;
19 #
20 cleanup() {
21     rm /tmp/output.$$
22     rm /tmp/expected.$$
23 }
24 #
25 trap cleanup EXIT
26 #
27 cat > /tmp/expected.$$ <<EOF
28 ^psecflags\([0-9A-F]+\, PSF_INHERIT, \{ PROC_SEC_ASLR, 0x0, 0x0, B_FALSE \}\) =
29 EOF
30 #
31 truss -t psecflags /usr/bin/psecflags -s current,aslr -e ls \
32     >/dev/null 2>output.$$
33 #
34 if ! grep -qEf /tmp/expected.$$ /tmp/output.$$; then
35     echo "truss: failed"
36     echo "output:"
37     sed -e 's/^/ /' output.$$
38     echo "should match:"
39     sed -e 's/^/ /' expected.$$
40     exit 1;
41 fi
42 #
43 exit 0
44 #endif /* ! codereview */
```

new/usr/src/test/os-tests/tests/secflags/secflags_zonecfg.sh

1

```
*****
3782 Thu Jun 30 22:03:08 2016
new/usr/src/test/os-tests/tests/secflags/secflags_zonecfg.sh
Code review comments from pmooney (sundry), and igork (screwups in zonecfg refac
Code review comments from jeffpc
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
1 #!/usr/bin/ksh
2 #
3 #
4 # This file and its contents are supplied under the terms of the
5 # Common Development and Distribution License ("CDDL"), version 1.0.
6 # You may only use this file in accordance with the terms of version
7 # 1.0 of the CDDL.
8 #
9 # A full copy of the text of the CDDL should have accompanied this
10 # source. A copy of the CDDL is also available via the Internet at
11 # http://www.illumos.org/license/CDDL.
12 #
13 #
14 # Copyright 2015, Richard Lowe.
15 #
16 # Verify that zones can be configured with security-flags
17 LC_ALL=C # Collation is important
18 #
19 expect_success() {
20     name=$1
21 #
22     (echo "create -b";
23      echo "set zonepath=/${name}.$$";
24      cat /dev/stdin;
25      echo "verify";
26      echo "commit";
27      echo "exit") | zonecfg -z ${name}.$$ > out.$$ 2>&1
28 #
29     r=$?
30 #
31     zonecfg -z ${name}.$$ delete -F
32 #
33     if (( $r != 0 )); then
34         printf "%s: FAIL\n" $name
35         cat out.$$
36         rm out.$$
37         return 1
38     else
39         rm out.$$
40         printf "%s: PASS\n" $name
41         return 0
42     fi
43 }
44 #
45 expect_fail() {
46     name=$1
47     expect=$2
48 #
49     (echo "create -b";
50      echo "set zonepath=/${name}.$$";
51      cat /dev/stdin;
52      echo "verify";
53      echo "commit";
54      echo "exit") | zonecfg -z ${name}.$$ > out.$$ 2>&1
55 #
56     r=$?
```

new/usr/src/test/os-tests/tests/secflags/secflags_zonecfg.sh

2

```
58 # Ideally will fail, since we don't want the create to have succeeded.
59 zonecfg -z ${name}.$$ delete -F >/dev/null 2>&1
60 #
61 #
62 if (( $r == 0 )); then
63     printf "%s: FAIL (succeeded)\n" $name
64     rm out.$$
65     return 1
66 else
67     grep -q "$expect" out.$$
68     if (( $ $? != 0 )); then
69         printf "%s: FAIL (error didn't match)\n" $name
70         echo "Wanted:"
71         echo "$expect"
72         echo "Got:"
73         sed -e 's/^/ /' out.$$
74         rm out.$$
75         return 1;
76     else
77         rm out.$$
78         printf "%s: PASS\n" $name
79         return 0
80     fi
81 fi
82 }
83 #
84 ret=0
85 #
86 expect_success valid-no-config <<EOF
87 EOF
88 (( $ $? != 0 )) && ret=1
89 #
90 expect_success valid-full-config <<EOF
91 add security-flags
92 set lower=none
93 set default=aslr
94 set upper=all
95 end
96 EOF
97 (( $ $? != 0 )) && ret=1
98 #
99 expect_success valid-partial-config <<EOF
100 add security-flags
101 set default=aslr
102 end
103 EOF
104 (( $ $? != 0 )) && ret=1
105 #
106 expect_fail invalid-full-lower-gt-def "default secflags must be above the lower
107 add security-flags
108 set lower=aslr
109 set default=none
110 set upper=all
111 end
112 EOF
113 (( $ $? != 0 )) && ret=1
114 #
115 expect_fail invalid-partial-lower-gt-def "default secflags must be above the low
116 add security-flags
117 set lower=aslr
118 set default=none
119 end
120 EOF
121 (( $ $? != 0 )) && ret=1
```

```
123 expect_fail invalid-full-def-gt-upper "default secflags must be within the upper
124 add security-flags
125 set lower=none
126 set default=all
127 set upper=none
128 end
129 EOF
130 (( $? != 0 )) && ret=1

132 expect_fail invalid-partial-def-gt-upper "default secflags must be within the up
133 add security-flags
134 set default=all
135 set upper=none
136 end
137 EOF
138 (( $? != 0 )) && ret=1

140 expect_fail invalid-full-def-gt-upper "default secflags must be within the upper
141 add security-flags
142 set lower=none
143 set default=all
144 set upper=none
145 end
146 EOF
147 (( $? != 0 )) && ret=1

149 expect_fail invalid-partial-lower-gt-upper "lower secflags must be within the up
150 add security-flags
151 set lower=all
152 set upper=none
153 end
154 EOF
155 (( $? != 0 )) && ret=1

157 expect_fail invalid-parse-fail-def "default security flags 'fail' are invalid" <
158 add security-flags
159 set default=fail
160 end
161 EOF
162 (( $? != 0 )) && ret=1

164 expect_fail invalid-parse-fail-lower "lower security flags 'fail' are invalid" <
165 add security-flags
166 set lower=fail
167 end
168 EOF
169 (( $? != 0 )) && ret=1

171 expect_fail invalid-parse-fail-def "upper security flags 'fail' are invalid" <<E
172 add security-flags
173 set upper=fail
174 end
175 EOF
176 (( $? != 0 )) && ret=1

178 exit $ret
179 #endif /* ! codereview */
```

new/usr/src/test/os-tests/tests/secflags/stacky.c

1

```
*****  
      86 Thu Jun 30 22:03:09 2016  
new/usr/src/test/os-tests/tests/secflags/stacky.c  
7029 want per-process exploit mitigation features (secflags)  
7030 want basic address space layout randomization (aslr)  
7031 noexec_user_stack should be a secflag  
7032 want a means to forbid mappings around NULL.  
*****
```

```
1 #include <unistd.h>  
  
3 int  
4 main(int argc, char **argv)  
5 {  
6     sleep(100000);  
7     return (0);  
8 }  
9 #endif /* ! codereview */
```

new/usr/src/uts/common/Makefile.files

1

```
*****
46683 Thu Jun 30 22:03:09 2016
new/usr/src/uts/common/Makefile.files
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
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 #
23 # Copyright (c) 1991, 2010, Oracle and/or its affiliates. All rights reserved.
24 # Copyright (c) 2012 Joyent, Inc. All rights reserved.
25 # Copyright (c) 2011, 2014 by Delphix. All rights reserved.
26 # Copyright (c) 2013 by Saso Kiselkov. All rights reserved.
27 # Copyright 2015 Nexenta Systems, Inc. All rights reserved.
28 # Copyright 2016 Garrett D'Amore <garrett@damore.org>
29 #
30 #
31 #
32 # This Makefile defines all file modules for the directory uts/common
33 # and its children. These are the source files which may be considered
34 # common to all SunOS systems.
35 #
36 i386_CORE_OBJS += \
37     atomic.o      \
38     avintr.o      \
39     pic.o
40 #
41 sparc_CORE_OBJS +=
42 #
43 COMMON_CORE_OBJS += \
44     beep.o        \
45     bitset.o      \
46     bp_map.o      \
47     brand.o       \
48     cpucaps.o     \
49     cmt.o         \
50     cmt_policy.o  \
51     cpu.o         \
52     cpu_event.o   \
53     cpu_intr.o    \
54     cpu_pm.o      \
55     cpupart.o     \
56     cap_util.o    \
57     disp.o        \
58     group.o
```

new/usr/src/uts/common/Makefile.files

2

```
59     kstat_fr.o    \
60     iscsiboot_prop.o \
61     lgrp.o        \
62     lgrp_topo.o   \
63     mmapobj.o     \
64     mutex.o       \
65     page_lock.o   \
66     page_retire.o \
67     panic.o       \
68     param.o       \
69     pg.o          \
70     pghw.o        \
71     putnext.o     \
72     rctl_proc.o   \
73     rwalk.o       \
74     seg_kmem.o    \
75     softint.o     \
76     string.o      \
77     strtol.o      \
78     strtoul.o     \
79     strtoll.o     \
80     strtoull.o    \
81     thread_intr.o \
82     vm_page.o     \
83     vm_pagelist.o \
84     zlib_obj.o    \
85     clock_tick.o
86 #
87 CORE_OBJS += $(COMMON_CORE_OBJS) $(MACH)_CORE_OBJS
88 #
89 ZLIB_OBJS = zutil.o zmod.o zmod_subr.o \
90     Adler32.o crc32.o deflate.o inffast.o \
91     inflate.o inftrees.o trees.o
92 #
93 GENUINIX_OBJS += \
94     access.o      \
95     acl.o         \
96     acl_common.o  \
97     adjtime.o     \
98     alarm.o       \
99     aio_subr.o    \
100    auditsys.o     \
101    audit_core.o   \
102    audit_zone.o   \
103    audit_memory.o \
104    autoconf.o     \
105    avl.o          \
106    bdev_dsort.o   \
107    bio.o          \
108    bitmap.o       \
109    blabel.o       \
110    brandsys.o     \
111    bz2blocksort.o \
112    bz2compress.o  \
113    bz2decompress.o \
114    bz2randtable.o \
115    bz2zlib.o      \
116    bz2crctable.o \
117    bz2huffman.o   \
118    callb.o        \
119    callout.o      \
120    chdir.o        \
121    chmod.o        \
122    chown.o        \
123    cladm.o        \
124    class.o
```

new/usr/src/uts/common/Makefile.files

```

125      clock.o \
126      clock_highres.o \
127      clock_realtime.o \
128      close.o \
129      compress.o \
130      condvar.o \
131      conf.o \
132      console.o \
133      contract.o \
134      copyops.o \
135      core.o \
136      corectl.o \
137      cred.o \
138      cs_stubs.o \
139      dacf.o \
140      dacf_clnt.o \
141      damap.o \
142      cyclic.o \
143      ddi.o \
144      ddi_fm.o \
145      ddi_hp_impl.o \
146      ddi_hp_ndi.o \
147      ddi_intr.o \
148      ddi_intr_impl.o \
149      ddi_intr_irm.o \
150      ddi_nodeid.o \
151      ddi_periodic.o \
152      devcfg.o \
153      devcache.o \
154      device.o \
155      devid.o \
156      devid_cache.o \
157      devid_scsi.o \
158      devid_smp.o \
159      devpolicy.o \
160      disp_lock.o \
161      dnlc.o \
162      driver.o \
163      dumpsubr.o \
164      driver_lyr.o \
165      dtrace_subr.o \
166      errorq.o \
167      etheraddr.o \
168      evchannels.o \
169      exacct.o \
170      exacct_core.o \
171      exec.o \
172      exit.o \
173      fbio.o \
174      fcntl.o \
175      fdbuffer.o \
176      fdsync.o \
177      fem.o \
178      ffs.o \
179      fio.o \
180      flock.o \
181      fm.o \
182      fork.o \
183      vpm.o \
184      fs_reparse.o \
185      fs_subr.o \
186      fsflush.o \
187      ftrace.o \
188      getcwd.o \
189      getdents.o \
190      getloadavg.o \

```

3

new/usr/src/uts/common/Makefile.files

```

191      getpagesizes.o \
192      getpid.o \
193      gfs.o \
194      rusage_sys.o \
195      gid.o \
196      groups.o \
197      grow.o \
198      hat_refmod.o \
199      id32.o \
200      id_space.o \
201      inet_ntop.o \
202      instance.o \
203      ioctl.o \
204      ip_cksum.o \
205      issetugid.o \
206      ippconf.o \
207      kpcp.o \
208      kdi.o \
209      kiconv.o \
210      klpd.o \
211      kmem.o \
212      ksyms_snapshot.o \
213      l_strplumb.o \
214      labelsys.o \
215      link.o \
216      list.o \
217      lockstat_subr.o \
218      log_sysevent.o \
219      logsubr.o \
220      lookup.o \
221      lseek.o \
222      ltos.o \
223      lwp.o \
224      lwp_create.o \
225      lwp_info.o \
226      lwp_self.o \
227      lwp_sobj.o \
228      lwp_timer.o \
229      lwpsys.o \
230      main.o \
231      mmapobjsys.o \
232      memcntl.o \
233      memstr.o \
234      lgrpsys.o \
235      mkdir.o \
236      mknod.o \
237      mount.o \
238      move.o \
239      msacct.o \
240      multidata.o \
241      nbmllock.o \
242      ndifm.o \
243      nice.o \
244      netstack.o \
245      ntptime.o \
246      nvpair.o \
247      nvpair_alloc_system.o \
248      nvpair_alloc_fixed.o \
249      fnvpair.o \
250      octet.o \
251      open.o \
252      p_online.o \
253      pathconf.o \
254      pathname.o \
255      pause.o \
256      serializer.o \

```

4

```

257         pci_intr_lib.o \
258         pci_cap.o \
259         pcifm.o \
260         pgrp.o \
261         pgrpsys.o \
262         pid.o \
263         pkp_hash.o \
264         policy.o \
265         poll.o \
266         pool.o \
267         pool_pset.o \
268         port_subr.o \
269         ppriv.o \
270         printf.o \
271         pricntl.o \
272         priv.o \
273         priv_const.o \
274         proc.o \
275         psecflags.o \
276 #endif /* ! codereview */
277         procset.o \
278         processor_bind.o \
279         processor_info.o \
280         profil.o \
281         project.o \
282         qsort.o \
283         getrandom.o \
284         rctl.o \
285         rctlsys.o \
286         readlink.o \
287         refstr.o \
288         rename.o \
289         resolvepath.o \
290         retire_store.o \
291         process.o \
292         rlimit.o \
293         rmap.o \
294         rw.o \
295         rwstlock.o \
296         sad_conf.o \
297         sid.o \
298         sidsys.o \
299         sched.o \
300         schedctl.o \
301         sctp_crc32.o \
302         secflags.o \
303 #endif /* ! codereview */
304         seg_dev.o \
305         seg_kp.o \
306         seg_kpm.o \
307         seg_map.o \
308         seg_vn.o \
309         seg_spt.o \
310         semaphore.o \
311         sendfile.o \
312         session.o \
313         share.o \
314         shuttle.o \
315         sig.o \
316         sigaction.o \
317         sigaltstack.o \
318         signotify.o \
319         sigpending.o \
320         sigprocmask.o \
321         sigqueue.o \
322         sigsendset.o \

```

```

323         sigsuspend.o \
324         sigtimedwait.o \
325         sleepq.o \
326         sock_conf.o \
327         space.o \
328         sscanf.o \
329         stat.o \
330         statfs.o \
331         statvfs.o \
332         stol.o \
333         str_conf.o \
334         strcalls.o \
335         stream.o \
336         streamio.o \
337         strext.o \
338         strsubr.o \
339         strsun.o \
340         subr.o \
341         sunddi.o \
342         sunmdi.o \
343         sunndi.o \
344         sunpci.o \
345         sunpm.o \
346         sundlpi.o \
347         suntpi.o \
348         swap_subr.o \
349         swap_vnops.o \
350         symlink.o \
351         sync.o \
352         sysclass.o \
353         sysconfig.o \
354         sysent.o \
355         sysfs.o \
356         systeminfo.o \
357         task.o \
358         taskq.o \
359         tasksys.o \
360         time.o \
361         timer.o \
362         times.o \
363         timers.o \
364         thread.o \
365         tlabel.o \
366         tnf_res.o \
367         turnstile.o \
368         tty_common.o \
369         u8_textprep.o \
370         uadmin.o \
371         uconv.o \
372         ucredsys.o \
373         uid.o \
374         umask.o \
375         umount.o \
376         uname.o \
377         unix_bb.o \
378         unlink.o \
379         urw.o \
380         utime.o \
381         utssys.o \
382         uucopy.o \
383         vfs.o \
384         vfs_conf.o \
385         vmem.o \
386         vm_anon.o \
387         vm_as.o \
388         vm_meter.o \

```

```

389         vm_pageout.o \
390         vm_pvn.o \
391         vm_rm.o \
392         vm_seg.o \
393         vm_subr.o \
394         vm_swap.o \
395         vm_usage.o \
396         vnode.o \
397         vuid_queue.o \
398         vuid_store.o \
399         waitq.o \
400         watchpoint.o \
401         yield.o \
402         scsi_confdata.o \
403         xattr.o \
404         xattr_common.o \
405         xdr_mblk.o \
406         xdr_mem.o \
407         xdr.o \
408         xdr_array.o \
409         xdr_refer.o \
410         zone.o

412 #
413 #     Stubs for the stand-alone linker/loader
414 #
415 sparc_GENSTUBS_OBJS = \
416     kobj_stubs.o

418 i386_GENSTUBS_OBJS =

420 COMMON_GENSTUBS_OBJS =

422 GENSTUBS_OBJS += $(COMMON_GENSTUBS_OBJS) $($ (MACH)_GENSTUBS_OBJS)

424 #
425 #     DTrace and DTrace Providers
426 #
427 DTRACE_OBJS += dtrace.o dtrace_isa.o dtrace_asm.o

429 SDT_OBJS += sdt_subr.o

431 PROFILE_OBJS += profile.o

433 SYSTRACE_OBJS += systrace.o

435 LOCKSTAT_OBJS += lockstat.o

437 FASTTRAP_OBJS += fasttrap.o fasttrap_isa.o

439 DCPC_OBJS += dcpc.o

441 #
442 #     Driver (pseudo-driver) Modules
443 #
444 IPP_OBJS += ippctl.o

446 AUDIO_OBJS += audio_client.o audio_ddi.o audio_engine.o \
447     audio_fltdata.o audio_format.o audio_ctrl.o \
448     audio_grc3.o audio_output.o audio_input.o \
449     audio_oss.o audio_sun.o

451 AUDIOEMU10K_OBJS += audioemu10k.o

453 AUDIOENS_OBJS += audioens.o

```

```

455 AUDIOVIA823X_OBJS += audiovia823x.o

457 AUDIOVIA97_OBJS += audiovia97.o

459 AUDIO1575_OBJS += audio1575.o

461 AUDIO810_OBJS += audio810.o

463 AUDIOCMI_OBJS += audiocmi.o

465 AUDIOCMIHD_OBJS += audiocmihd.o

467 AUDIOHD_OBJS += audiohd.o

469 AUDIOIXP_OBJS += audioixp.o

471 AUDIOLS_OBJS += audiolis.o

473 AUDIOP16X_OBJS += audiop16x.o

475 AUDIOPCI_OBJS += audiopci.o

477 AUDIOSOLO_OBJS += audiosolo.o

479 AUDIOTS_OBJS += audiotis.o

481 AC97_OBJS += ac97.o ac97_ad.o ac97_alc.o ac97_cmi.o

483 BLKDEV_OBJS += blkdev.o

485 CARDBUS_OBJS += cardbus.o cardbus_hp.o cardbus_cfg.o

487 CONSKBD_OBJS += conskbd.o

489 CONSMS_OBJS += consms.o

491 OLDPTY_OBJS += tty_ptyconf.o

493 PTC_OBJS += tty_pty.o

495 PTSL_OBJS += tty_pts.o

497 PTM_OBJS += ptm.o

499 MII_OBJS += mii.o mii_cicada.o mii_natsemi.o mii_intel.o mii_qualsemi.o \
500     mii_marvell.o mii_realtek.o mii_other.o

502 PTS_OBJS += pts.o

504 PTY_OBJS += ptms_conf.o

506 SAD_OBJS += sad.o

508 MD4_OBJS += md4.o md4_mod.o

510 MD5_OBJS += md5.o md5_mod.o

512 SHA1_OBJS += sha1.o sha1_mod.o

514 SHA2_OBJS += sha2.o sha2_mod.o

516 SKEIN_OBJS += skein.o skein_block.o skein_iv.o skein_mod.o

518 EDONR_OBJS += edonr.o edonr_mod.o

520 IPGPC_OBJS += classifierddi.o classifier.o filters.o trie.o table.o \

```



```

521          ba_table.o
523 DSCPMK_OBJS += dscpmk.o dscpmkddi.o
525 DLCOSMK_OBJS += dlcosmk.o dlcosmkddi.o
527 FLOWACCT_OBJS +=          flowacctddi.o flowacct.o
529 TOKENMT_OBJS += tokenmt.o tokenmtddi.o
531 TSWTCL_OBJS += tswtcl.o tswtclddi.o
533 ARP_OBJS += arpddi.o
535 ICMP_OBJS += icmpddi.o
537 ICMP6_OBJS += icmp6ddi.o
539 RTS_OBJS += rtsddi.o

541 IP_ICMP_OBJS = icmp.o icmp_opt_data.o
542 IP_RTS_OBJS = rts.o rts_opt_data.o
543 IP_TCP_OBJS = tcp.o tcp_fusion.o tcp_opt_data.o tcp_sack.o tcp_stats.o \
544             tcp_misc.o tcp_timers.o tcp_time_wait.o tcp_tpi.o tcp_output.o \
545             tcp_input.o tcp_socket.o tcp_bind.o tcp_cluster.o tcp_tunables.o
546 IP_UDP_OBJS = udp.o udp_opt_data.o udp_tunables.o udp_stats.o
547 IP_SCTP_OBJS = sctp.o sctp_opt_data.o sctp_output.o \
548              sctp_init.o sctp_input.o sctp_cookie.o \
549              sctp_conn.o sctp_error.o sctp_snmp.o \
550              sctp_tunables.o sctp_shutdown.o sctp_common.o \
551              sctp_timer.o sctp_heartbeat.o sctp_hash.o \
552              sctp_bind.o sctp_notify.o sctp_asconf.o \
553              sctp_addr.o tn_ipopt.o tnet.o ip_netinfo.o \
554              sctp_misc.o
555 IP_ILB_OBJS = ilb.o ilb_nat.o ilb_conn.o ilb_alg_hash.o ilb_alg_rr.o

557 IP_OBJS += igmp.o ipmp.o ip.o ip6.o ip6_asp.o ip6_if.o ip6_ire.o \
558            ip6_rts.o ip_if.o ip_ire.o ip_listutils.o ip_mrout.o \
559            ip_multi.o ip2mac.o ip_ndp.o ip_rts.o ip_srcid.o \
560            ipddi.o ipdrop.o mi.o nd.o tunables.o optcom.o snmpcom.o \
561            ipsec_loader.o spd.o ipclassifier.o inet_common.o ip_queue.o \
562            squeue.o ip_sadb.o ip_ftable.o proto_set.o radix.o ip_dummy.o \
563            ip_helper_stream.o ip_tunables.o \
564            ip_output.o ip_input.o ip6_input.o ip6_output.o ip_arp.o \
565            conn_opt.o ip_attr.o ip_dce.o \
566            $(IP_ICMP_OBJS) \
567            $(IP_RTS_OBJS) \
568            $(IP_TCP_OBJS) \
569            $(IP_UDP_OBJS) \
570            $(IP_SCTP_OBJS) \
571            $(IP_ILB_OBJS)

573 IP6_OBJS += ip6ddi.o

575 HOOK_OBJS += hook.o

577 NETI_OBJS += neti_impl.o neti_mod.o neti_stack.o

579 KEYSOCK_OBJS += keysockddi.o keysock.o keysock_opt_data.o

581 IPNET_OBJS += ipnet.o ipnet_bpf.o

583 SPDSOCK_OBJS += spdsockddi.o spdsock.o spdsock_opt_data.o

585 IPSECESP_OBJS += ipsecespddi.o ipsecesp.o

```

```

587 IPSECAH_OBJS += ipsecahddi.o ipsecah.o sadb.o
589 SPPP_OBJS += sPPP.o sPPP_dlpi.o sPPP_mod.o s_common.o
591 SPPPTUN_OBJS += sppptun.o sppptun_mod.o
593 SPPPASYN_OBJS += spppasyn.o spppasyn_mod.o

595 SPPPCOMP_OBJS += sppppcomp.o sppppcomp_mod.o deflate.o bsd-comp.o vjcompress.o \
596                zlib.o

598 TCP_OBJS += tcpddi.o

600 TCP6_OBJS += tcp6ddi.o

602 NCA_OBJS += ncaddi.o

604 SDP SOCK_MOD_OBJS += sockmod_sdp.o socksdp.o socksdpsubr.o

606 SCTP SOCK_MOD_OBJS += sockmod_sctp.o socksctp.o socksctpsubr.o

608 PFP SOCK_MOD_OBJS += sockmod_pfp.o

610 RDS SOCK_MOD_OBJS += sockmod_rds.o

612 RDS_OBJS += rdsddi.o rdssubr.o rds_opt.o rds_ioctl.o

614 RDSIB_OBJS += rdsib.o rdsib_ib.o rdsib_cm.o rdsib_ep.o rdsib_buf.o \
615              rdsib_debug.o rdsib_sc.o

617 RDSV3_OBJS += af_rds.o rdsv3_ddi.o bind.o loop.o threads.o connection.o \
618              transport.o cong.o sysctl.o message.o rds_recv.o send.o \
619              stats.o info.o page.o rdma_transport.o ib_ring.o ib_rdma.o \
620              ib_recv.o ib.o ib_send.o ib_sysctl.o ib_stats.o ib_cm.o \
621              rdsv3_sc.o rdsv3_debug.o rdsv3_impl.o rdma.o rdsv3_af_thr.o

623 ISER_OBJS += iser.o iser_cm.o iser_cq.o iser_ib.o iser_idm.o \
624              iser_resource.o iser_xfer.o

626 UDP_OBJS += udpddi.o

628 UDP6_OBJS += udp6ddi.o

630 SY_OBJS += gentyty.o

632 TCO_OBJS += ticots.o

634 TCOO_OBJS += ticotsord.o

636 TCL_OBJS += ticlts.o

638 TL_OBJS += tl.o

640 DUMP_OBJS += dump.o

642 BPF_OBJS += bpf.o bpf_filter.o bpf_mod.o bpf_dlt.o bpf_mac.o

644 CLONE_OBJS += clone.o

646 CN_OBJS += cons.o

648 DLD_OBJS += dld_drv.o dld_proto.o dld_str.o dld_flow.o

650 DLS_OBJS += dls.o dls_link.o dls_mod.o dls_stat.o dls_mgmt.o

652 GLD_OBJS += gld.o gldutil.o

```

```

654 MAC_OBJS +=      mac.o mac_bcast.o mac_client.o mac_datapath_setup.o mac_flow.o
655                mac_hio.o mac_mod.o mac_ndd.o mac_provider.o mac_sched.o \
656                mac_protect.o mac_soft_ring.o mac_stat.o mac_util.o

658 MAC_6TO4_OBJS +=      mac_6to4.o

660 MAC_ETHER_OBJS +=      mac_ether.o

662 MAC_IPV4_OBJS +=      mac_ipv4.o

664 MAC_IPV6_OBJS +=      mac_ipv6.o

666 MAC_WIFI_OBJS +=      mac_wifi.o

668 MAC_IB_OBJS +=        mac_ib.o

670 IPTUN_OBJS +=        iptun_dev.o iptun_ctl.o iptun.o

672 AGGR_OBJS +=          aggr_dev.o aggr_ctl.o aggr_grp.o aggr_port.o \
673                      aggr_send.o aggr_recv.o aggr_lacp.o

675 SOFTMAC_OBJS +=       softmac_main.o softmac_ctl.o softmac_capab.o \
676                      softmac_dev.o softmac_stat.o softmac_pkt.o softmac_fp.o

678 NET80211_OBJS +=      net80211.o net80211_proto.o net80211_input.o \
679                      net80211_output.o net80211_node.o net80211_crypto.o \
680                      net80211_crypto_none.o net80211_crypto_wep.o net80211_ioctl.o \
681                      net80211_crypto_tkip.o net80211_crypto_ccmp.o \
682                      net80211_ht.o

684 VNIC_OBJS +=          vnic_ctl.o vnic_dev.o

686 SIMNET_OBJS +=        simnet.o

688 IB_OBJS +=            ibnex.o ibnex_ioctl.o ibnex_hca.o

690 IBCM_OBJS +=           ibcm_impl.o ibcm_sm.o ibcm_ti.o ibcm_utils.o ibcm_path.o \
691                      ibcm_arp.o ibcm_arp_link.o

693 IBDM_OBJS +=           ibdm.o

695 IBDMA_OBJS +=          ibdma.o

697 IBMF_OBJS +=           ibmf.o ibmf_impl.o ibmf_dr.o ibmf_wqe.o ibmf_ud_dest.o ibmf_mod.o
698                      ibmf_send.o ibmf_recv.o ibmf_handlers.o ibmf_trans.o \
699                      ibmf_timers.o ibmf_msg.o ibmf_utils.o ibmf_rmpp.o \
700                      ibmf_saa.o ibmf_saa_impl.o ibmf_saa_utils.o ibmf_saa_events.o

702 IBTL_OBJS +=           ibtl_impl.o ibtl_util.o ibtl_mem.o ibtl_handlers.o ibtl_qp.o \
703                      ibtl_cg.o ibtl_wr.o ibtl_hca.o ibtl_chan.o ibtl_cm.o \
704                      ibtl_mcg.o ibtl_ibnex.o ibtl_srqp.o ibtl_part.o

706 TAVOR_OBJS +=          tavor.o tavor_agents.o tavor_cfg.o tavor_ci.o tavor_cmd.o \
707                      tavor_cq.o tavor_event.o tavor_ioctl.o tavor_misc.o \
708                      tavor_mr.o tavor_qp.o tavor_qpmod.o tavor_rsrc.o \
709                      tavor_srqp.o tavor_stats.o tavor_umap.o tavor_wr.o

711 HERMON_OBJS +=        hermon.o hermon_agents.o hermon_cfg.o hermon_ci.o hermon_cmd.o \
712                      hermon_cq.o hermon_event.o hermon_ioctl.o hermon_misc.o \
713                      hermon_mr.o hermon_qp.o hermon_qpmod.o hermon_rsrc.o \
714                      hermon_srqp.o hermon_stats.o hermon_umap.o hermon_wr.o \
715                      hermon_fcoib.o hermon_fm.o

717 DAPLT_OBJS +=          daplt.o

```

```

719 SOL_OFS_OBJS +=       sol_cma.o sol_ib_cma.o sol_uobj.o \
720                      sol_ofs_debug_util.o sol_ofs_gen_util.o \
721                      sol_kverbs.o

723 SOL_UCMA_OBJS +=       sol_ucma.o

725 SOL_UVERBS_OBJS +=     sol_uverbs.o sol_uverbs_comp.o sol_uverbs_event.o \
726                      sol_uverbs_hca.o sol_uverbs_qp.o

728 SOL_UMAD_OBJS +=       sol_umad.o

730 KSTAT_OBJS +=          kstat.o

732 KSYMS_OBJS +=          ksyms.o

734 INSTANCE_OBJS +=       inst_sync.o

736 IWSCN_OBJS +=          iwscons.o

738 LOFI_OBJS +=           lofi.o LzmaDec.o

740 FSSNAP_OBJS +=          fssnap.o

742 FSSNAPIF_OBJS +=        fssnap_if.o

744 MM_OBJS +=              mem.o

746 PHYSMEM_OBJS +=        physmem.o

748 OPTIONS_OBJS +=        options.o

750 WINLOCK_OBJS +=        winlockio.o

752 PM_OBJS +=              pm.o
753 SRN_OBJS +=              srn.o

755 PSEUDO_OBJS +=          pseudonex.o

757 RAMDISK_OBJS +=         ramdisk.o

759 LLC1_OBJS +=            llc1.o

761 USBKBM_OBJS +=          usbkbm.o

763 USBWCM_OBJS +=          usbwcm.o

765 BOFI_OBJS +=            bofi.o

767 HID_OBJS +=             hid.o

769 USBSKEL_OBJS +=         usbskel.o

771 USBVC_OBJS +=           usbvc.o usbvc_v412.o

773 HIDPARSER_OBJS +=       hidparser.o

775 USB_AC_OBJS +=          usb_ac.o

777 USB_AS_OBJS +=          usb_as.o

779 USB_AH_OBJS +=          usb_ah.o

781 USBMS_OBJS +=           usbms.o

783 USBPRN_OBJS +=          usbprn.o

```

```

785 UGEN_OBJS += ugen.o
787 USBSER_OBJS += usbser.o usbser_rseq.o
789 USBSACM_OBJS += usbsacm.o
791 USBSER_KEYSPAN_OBJS += usbser_keyspan.o keyspan_dsd.o keyspan_pipe.o
793 USBS49_FW_OBJS += keyspan_49fw.o
795 USBSPRL_OBJS += usbser_pl2303.o pl2303_dsd.o
797 USBFTDI_OBJS += usbser_uftdi.o uftdi_dsd.o
799 USBECM_OBJS += usbecm.o
801 WC_OBJS += wscons.o vcons.o
803 VCONS_CONF_OBJS += vcons_conf.o
805 SCSI_OBJS +=      scsi_capabilities.o scsi_confsubr.o scsi_control.o \
806                  scsi_data.o scsi_fm.o scsi_hba.o scsi_reset_notify.o \
807                  scsi_resource.o scsi_subr.o scsi_transport.o scsi_watch.o \
808                  smp_transport.o
810 SCSI_VHCI_OBJS +=      scsi_vhci.o mpapi_impl.o scsi_vhci_tpgs.o
812 SCSI_VHCI_F_SYM_OBJS +=      sym.o
814 SCSI_VHCI_F_TPGS_OBJS +=      tpgs.o
816 SCSI_VHCI_F_ASYM_SUN_OBJS +=  asym_sun.o
818 SCSI_VHCI_F_SYM_HDS_OBJS +=  sym_hds.o
820 SCSI_VHCI_F_TAPE_OBJS +=      tape.o
822 SCSI_VHCI_F_TPGS_TAPE_OBJS += tpgs_tape.o
824 SGEN_OBJS +=      sgen.o
826 SMP_OBJS +=      smp.o
828 SATA_OBJS +=      sata.o
830 USBA_OBJS +=      hcidi.o usba.o usbai.o hubdi.o parser.o genconsole.o \
831                  usbai_pipe_mgmt.o usbai_req.o usbai_util.o usbai_register.o \
832                  usba_devdb.o usbai0_calls.o usba_ugen.o
834 USBA10_OBJS +=      usba10.o
836 RSM_OBJS +=      rsm.o rsmka_pathmanager.o rsmka_util.o
838 RSMOPS_OBJS +=      rsmops.o
840 S1394_OBJS +=      t1394.o t1394_errmsg.o s1394.o s1394_addr.o s1394_async.o \
841                  s1394_bus_reset.o s1394_cmp.o s1394_csr.o s1394_dev_disc.o \
842                  s1394_fa.o s1394_fcp.o \
843                  s1394_hotplug.o s1394_isoch.o s1394_misc.o h1394.o nx1394.o
845 HCI1394_OBJS +=      hcil1394.o hcil1394_async.o hcil1394_attach.o hcil1394_buf.o \
846                  hcil1394_csr.o hcil1394_detach.o hcil1394_extern.o \
847                  hcil1394_ioctl.o hcil1394_isoch.o hcil1394_isr.o \
848                  hcil1394_ixl_comp.o hcil1394_ixl_isr.o hcil1394_ixl_misc.o \
849                  hcil1394_ixl_update.o hcil1394_misc.o hcil1394_ohci.o \
850                  hcil1394_q.o hcil1394_s1394if.o hcil1394_tlabel.o \

```

```

851                  hcil1394_tlist.o hcil1394_vendor.o
853 AV1394_OBJS +=      avl1394.o avl1394_as.o avl1394_async.o avl1394_cfgrom.o \
854                  avl1394_cmp.o avl1394_fcp.o avl1394_isoch.o avl1394_isoch_chan.o \
855                  avl1394_isoch_recv.o avl1394_isoch_xmit.o avl1394_list.o \
856                  avl1394_queue.o
858 DCAM1394_OBJS +=      dcam.o dcam_frame.o dcam_param.o dcam_reg.o \
859                  dcam_ring_buff.o
861 SCSA1394_OBJS +=      hba.o sbp2_driver.o sbp2_bus.o
863 SBP2_OBJS +=      cfgrom.o sbp2.o
865 PMODEM_OBJS +=      pmodem.o pmodem_cis.o cis.o cis_callout.o cis_handlers.o cis_para
867 DSW_OBJS +=      dsw.o dsw_dev.o ii_tree.o
869 NCALL_OBJS +=      ncall.o \
870                  ncall_stub.o
872 RDC_OBJS +=      rdc.o \
873                  rdc_dev.o \
874                  rdc_io.o \
875                  rdc_clnt.o \
876                  rdc_prot_xdr.o \
877                  rdc_svc.o \
878                  rdc_bitmap.o \
879                  rdc_health.o \
880                  rdc_subr.o \
881                  rdc_diskq.o
883 RDCSRV_OBJS +=      rdcsrv.o
885 RDCSTUB_OBJS +=      rdc_stub.o
887 SDBC_OBJS +=      sd_bcache.o \
888                  sd_bio.o \
889                  sd_conf.o \
890                  sd_ft.o \
891                  sd_hash.o \
892                  sd_io.o \
893                  sd_misc.o \
894                  sd_pcu.o \
895                  sd_tdaemon.o \
896                  sd_trace.o \
897                  sd_iob_impl0.o \
898                  sd_iob_impl1.o \
899                  sd_iob_impl2.o \
900                  sd_iob_impl3.o \
901                  sd_iob_impl4.o \
902                  sd_iob_impl5.o \
903                  sd_iob_impl6.o \
904                  sd_iob_impl7.o \
905                  safestore.o \
906                  safestore_ram.o
908 NSCTL_OBJS +=      nsctl.o \
909                  nsc_cache.o \
910                  nsc_disk.o \
911                  nsc_dev.o \
912                  nsc_freeze.o \
913                  nsc_gen.o \
914                  nsc_mem.o \
915                  nsc_ncallio.o \
916                  nsc_power.o \

```

```

917         nsc_resv.o \
918         nsc_rmspin.o \
919         nsc_solaris.o \
920         nsc_trap.o \
921         nsc_list.o
922 UNISTAT_OBJS += spuni.o \
923         spcs_s_k.o

925 NSKERN_OBJS += nsc_ddi.o \
926         nsc_proc.o \
927         nsc_raw.o \
928         nsc_thread.o \
929         nskernd.o

931 SV_OBJS += sv.o

933 PMCS_OBJS += pmcs_attach.o pmcs_ds.o pmcs_intr.o pmcs_nvram.o pmcs_sata.o \
934         pmcs_scsa.o pmcs_smhba.o pmcs_subr.o pmcs_fwlog.o

936 PMCS8001FW_C_OBJS += pmcs_fw_hdr.o
937 PMCS8001FW_OBJS += $(PMCS8001FW_C_OBJS) SPCBoot.o ila.o firmware.o

939 #
940 #   Build up defines and paths.

942 ST_OBJS += st.o st_conf.o

944 EMLXS_OBJS += emlxs_clock.o emlxs_dfc.o emlxs_dhchap.o emlxs_diag.o \
945         emlxs_download.o emlxs_dump.o emlxs_els.o emlxs_event.o \
946         emlxs_fcf.o emlxs_fcp.o emlxs_fct.o emlxs_hba.o emlxs_ip.o \
947         emlxs_mbox.o emlxs_mem.o emlxs_msg.o emlxs_node.o \
948         emlxs_pkt.o emlxs_sli3.o emlxs_sli4.o emlxs_solaris.o \
949         emlxs_thread.o

951 EMLXS_FW_OBJS += emlxs_fw.o

953 OCE_OBJS += oce_buf.o oce_fm.o oce_gld.o oce_hw.o oce_intr.o oce_main.o \
954         oce_mbx.o oce_mq.o oce_queue.o oce_rx.o oce_stat.o oce_tx.o \
955         oce_utils.o

957 FCT_OBJS += discovery.o fct.o

959 QLT_OBJS += 2400.o 2500.o 8100.o qlt.o qlt_dma.o

961 SRPT_OBJS += srpt_mod.o srpt_ch.o srpt_cm.o srpt_ioc.o srpt_stp.o

963 FCOE_OBJS += fcoe.o fcoe_eth.o fcoe_fc.o

965 FCOET_OBJS += fcoet.o fcoet_eth.o fcoet_fc.o

967 FCOEI_OBJS += fcoei.o fcoei_eth.o fcoei_lv.o

969 ISCSIT_SHARED_OBJS += \
970         iscsit_common.o

972 ISCSIT_OBJS += $(ISCSIT_SHARED_OBJS) \
973         iscsit.o iscsit_tgt.o iscsit_sess.o iscsit_login.o \
974         iscsit_text.o iscsit_isns.o iscsit_radiusauth.o \
975         iscsit_radiuspacket.o iscsit_auth.o iscsit_authclient.o

977 PPPT_OBJS += alua_ic_if.o pppt.o pppt_msg.o pppt_tgt.o

979 STMF_OBJS += lun_map.o stmf.o

981 STMF_SBD_OBJS += sbd.o sbd_scsi.o sbd_pgr.o sbd_zvol.o

```

```

983 SYSMMSG_OBJS += sysmsg.o

985 SES_OBJS += ses.o ses_sen.o ses_saft.o ses_ses.o

987 TNF_OBJS += tnf_buf.o tnf_trace.o tnf_writer.o trace_init.o \
988         trace_funcs.o tnf_probe.o tnf.o

990 LOGINDMUX_OBJS += logindmux.o

992 DEVINFO_OBJS += devinfo.o

994 DEVPOLL_OBJS += devpoll.o

996 DEVPOOL_OBJS += devpool.o

998 EVENTFD_OBJS += eventfd.o

1000 SIGNALFD_OBJS += signalfd.o

1002 I8042_OBJS += i8042.o

1004 KB8042_OBJS += \
1005         at_keyprocess.o \
1006         kb8042.o \
1007         kb8042_keytables.o

1009 MOUSE8042_OBJS += mouse8042.o

1011 FDC_OBJS += fdc.o

1013 ASY_OBJS += asy.o

1015 ECPP_OBJS += ecpp.o

1017 VUIDM3P_OBJS += vuidmice.o vuidm3p.o

1019 VUIDM4P_OBJS += vuidmice.o vuidm4p.o

1021 VUIDM5P_OBJS += vuidmice.o vuidm5p.o

1023 VUIDPS2_OBJS += vuidmice.o vuidps2.o

1025 HPCSVCSVC_OBJS += hpcsvc.o

1027 PCIE_MISC_OBJS += pcie.o pcie_fault.o pcie_hp.o pciehpc.o pcishpc.o pcie_pwr.o p

1029 PCIHPNEXUS_OBJS += pcihp.o

1031 OPENEEPROM_OBJS += openprom.o

1033 RANDOM_OBJS += random.o

1035 PSHOT_OBJS += pshot.o

1037 GEN_DRV_OBJS += gen_drv.o

1039 TCLIENT_OBJS += tclient.o

1041 TIMERFD_OBJS += timerfd.o

1043 TPHCI_OBJS += tphci.o

1045 TVHCI_OBJS += tvhci.o

1047 EMUL64_OBJS += emul64.o emul64_bsd.o

```

```

1049 FCP_OBJS += fcp.o
1051 FCIP_OBJS += fcip.o
1053 FCSM_OBJS += fcsm.o
1055 FCTL_OBJS += fctl.o
1057 FP_OBJS += fp.o
1059 QLC_OBJS += ql_api.o ql_debug.o ql_hba_fru.o ql_init.o ql_iocb.o ql_ioctl.o \
1060 ql_isr.o ql_mbx.o ql_nx.o ql_xioctl.o ql_fw_table.o
1062 QLC_FW_2200_OBJS += ql_fw_2200.o
1064 QLC_FW_2300_OBJS += ql_fw_2300.o
1066 QLC_FW_2400_OBJS += ql_fw_2400.o
1068 QLC_FW_2500_OBJS += ql_fw_2500.o
1070 QLC_FW_6322_OBJS += ql_fw_6322.o
1072 QLC_FW_8100_OBJS += ql_fw_8100.o
1074 QLGE_OBJS += qlge.o qlge_dbg.o qlge_flash.o qlge_fm.o qlge_gld.o qlge_mpi.o
1076 ZCONS_OBJS += zcons.o
1078 NV_SATA_OBJS += nv_sata.o
1080 SI3124_OBJS += si3124.o
1082 AHCI_OBJS += ahci.o
1084 PCIIDE_OBJS += pci-ide.o
1086 PCEPP_OBJS += pcepp.o
1088 CPC_OBJS += cpc.o
1090 CPUID_OBJS += cpuid_drv.o
1092 SYSEVENT_OBJS += sysevent.o
1094 BL_OBJS += bl.o
1096 DRM_OBJS += drm_sunmod.o drm_kstat.o drm_agpsupport.o \
1097 drm_auth.o drm_bufs.o drm_context.o drm_dma.o \
1098 drm_drawable.o drm_drv.o drm_fops.o drm_ioctl.o drm_irq.o \
1099 drm_lock.o drm_memory.o drm_msg.o drm_pci.o drm_scatter.o \
1100 drm_cache.o drm_gem.o drm_mm.o ati_pciart.o
1102 FM_OBJS += devfm.o devfm_machdep.o
1104 RTLS_OBJS += rtls.o
1106 #
1107 #           exec modules
1108 #
1109 AOUTEXEC_OBJS += aout.o
1111 ELFEXEC_OBJS += elf.o elf_notes.o old_notes.o
1113 INTPEXEC_OBJS += intp.o

```

```

1115 SHBINEXEC_OBJS += shbin.o
1117 JAVAEXEC_OBJS += java.o
1119 #
1120 #           file system modules
1121 #
1122 AUTOFS_OBJS += auto_vfsops.o auto_vnops.o auto_subr.o auto_xdr.o auto_sys.o
1124 DCFS_OBJS += dc_vnops.o
1126 DEVFS_OBJS += devfs_subr.o devfs_vfsops.o devfs_vnops.o
1128 DEV_OBJS += sdev_subr.o sdev_vfsops.o sdev_vnops.o \
1129 sdev_ptsops.o sdev_zvolops.o sdev_comm.o \
1130 sdev_profile.o sdev_ncache.o sdev_netops.o \
1131 sdev_ipnetops.o \
1132 sdev_vttops.o
1134 CTFS_OBJS += ctfs_all.o ctfs_cdir.o ctfs_ctl.o ctfs_event.o \
1135 ctfs_latest.o ctfs_root.o ctfs_sym.o ctfs_tdir.o ctfs_tmpl.o
1137 OBJFS_OBJS += objfs_vfs.o objfs_root.o objfs_common.o \
1138 objfs_odir.o objfs_data.o
1140 FDFS_OBJS += fdops.o
1142 FIFO_OBJS += fifosubr.o fifovnops.o
1144 PIPE_OBJS += pipe.o
1146 HSFS_OBJS += hsfs_node.o hsfs_subr.o hsfs_vfsops.o hsfs_vnops.o \
1147 hsfs_susp.o hsfs_rrip.o hsfs_susp_subr.o
1149 LOFS_OBJS += lofs_subr.o lofs_vfsops.o lofs_vnops.o
1151 NAMEFS_OBJS += namevfs.o namevno.o
1153 NFS_OBJS += nfs_client.o nfs_common.o nfs_dump.o \
1154 nfs_subr.o nfs_vfsops.o nfs_vnops.o \
1155 nfs_xdr.o nfs_sys.o nfs_strerror.o \
1156 nfs3_vfsops.o nfs3_vnops.o nfs3_xdr.o \
1157 nfs_acl_vnops.o nfs_acl_xdr.o nfs4_vfsops.o \
1158 nfs4_vnops.o nfs4_xdr.o nfs4_idmap.o \
1159 nfs4_shadow.o nfs4_subr.o \
1160 nfs4_attr.o nfs4_rnode.o nfs4_client.o \
1161 nfs4_acache.o nfs4_common.o nfs4_client_state.o \
1162 nfs4_callback.o nfs4_recovery.o nfs4_client_secinfo.o \
1163 nfs4_client_debug.o nfs_stats.o \
1164 nfs4_acl.o nfs4_stub_vnops.o nfs_cmd.o
1166 NFSSRV_OBJS += nfs_server.o nfs_srv.o nfs3_srv.o \
1167 nfs_acl_srv.o nfs_auth.o nfs_auth_xdr.o \
1168 nfs_export.o nfs_log.o nfs_log_xdr.o \
1169 nfs4_srv.o nfs4_state.o nfs4_srv_attr.o \
1170 nfs4_srv_ns.o nfs4_db.o nfs4_srv_deleg.o \
1171 nfs4_deleg_ops.o nfs4_srv_readdir.o nfs4_dispatch.o
1173 SMBSRV_SHARED_OBJS += \
1174 smb_door_legacy.o \
1175 smb_inet.o \
1176 smb_match.o \
1177 smb_msgbuf.o \
1178 smb_native.o \
1179 smb_netbios_util.o \
1180 smb_oem.o \

```

```

1181         smb_sid.o \
1182         smb_status2winerr.o \
1183         smb_string.o \
1184         smb_token.o \
1185         smb_token_xdr.o \
1186         smb_utf8.o \
1187         smb_xdr.o

1189 # See also: $SRC/lib/smb/rv/libfksmb/rv/Makefile.com
1190 SMBSRV_OBJS += $(SMBSRV_SHARED_OBJS) \
1191         smb_acl.o \
1192         smb_alloc.o \
1193         smb_authenticate.o \
1194         smb_close.o \
1195         smb_cmn_rename.o \
1196         smb_cmn_setfile.o \
1197         smb_common_open.o \
1198         smb_common_transact.o \
1199         smb_create.o \
1200         smb_cred.o \
1201         smb_delete.o \
1202         smb_dfs.o \
1203         smb_directory.o \
1204         smb_dispatch.o \
1205         smb_echo.o \
1206         smb_errno.o \
1207         smb_fem.o \
1208         smb_find.o \
1209         smb_flush.o \
1210         smb_fsinfo.o \
1211         smb_fsops.o \
1212         smb_idmap.o \
1213         smb_init.o \
1214         smb_kdoor.o \
1215         smb_kshare.o \
1216         smb_kutil.o \
1217         smb_lock.o \
1218         smb_lock_byte_range.o \
1219         smb_locking_andx.o \
1220         smb_logoff_andx.o \
1221         smb_mangle_name.o \
1222         smb_mbuf_marshall.o \
1223         smb_mbuf_util.o \
1224         smb_negotiate.o \
1225         smb_net.o \
1226         smb_node.o \
1227         smb_notify.o \
1228         smb_nt_cancel.o \
1229         smb_nt_create_andx.o \
1230         smb_nt_transact_create.o \
1231         smb_nt_transact_ioctl.o \
1232         smb_nt_transact_notify_change.o \
1233         smb_nt_transact_quota.o \
1234         smb_nt_transact_security.o \
1235         smb_odir.o \
1236         smb_ofile.o \
1237         smb_open_andx.o \
1238         smb_opipe.o \
1239         smb_oplock.o \
1240         smb_pathname.o \
1241         smb_print.o \
1242         smb_process_exit.o \
1243         smb_query_fileinfo.o \
1244         smb_quota.o \
1245         smb_read.o \
1246         smb_rename.o \

```

```

1247         smb_sd.o \
1248         smb_seek.o \
1249         smb_server.o \
1250         smb_session.o \
1251         smb_session_setup_andx.o \
1252         smb_set_fileinfo.o \
1253         smb_sign_kcf.o \
1254         smb_signing.o \
1255         smb_thread.o \
1256         smb_tree.o \
1257         smb_trans2_create_directory.o \
1258         smb_trans2_dfs.o \
1259         smb_trans2_find.o \
1260         smb_tree_connect.o \
1261         smb_unlock_byte_range.o \
1262         smb_user.o \
1263         smb_vfs.o \
1264         smb_vops.o \
1265         smb_vss.o \
1266         smb_write.o \
1267         \
1268         smb2_dispatch.o \
1269         smb2_cancel.o \
1270         smb2_change_notify.o \
1271         smb2_close.o \
1272         smb2_create.o \
1273         smb2_echo.o \
1274         smb2_flush.o \
1275         smb2_ioctl.o \
1276         smb2_lock.o \
1277         smb2_logoff.o \
1278         smb2_negotiate.o \
1279         smb2_ofile.o \
1280         smb2_oplock.o \
1281         smb2_qinfo_file.o \
1282         smb2_qinfo_fs.o \
1283         smb2_qinfo_sec.o \
1284         smb2_qinfo_quota.o \
1285         smb2_query_dir.o \
1286         smb2_query_info.o \
1287         smb2_read.o \
1288         smb2_session_setup.o \
1289         smb2_set_info.o \
1290         smb2_setinfo_file.o \
1291         smb2_setinfo_fs.o \
1292         smb2_setinfo_quota.o \
1293         smb2_setinfo_sec.o \
1294         smb2_signing.o \
1295         smb2_tree_connect.o \
1296         smb2_tree_disconn.o \
1297         smb2_write.o

1299 PCFS_OBJS += pc_alloc.o      pc_dir.o      pc_node.o      pc_subr.o \
1300             pc_vfsops.o      pc_vnops.o

1302 PROC_OBJS += prcontrol.o     prioctl.o     prsubr.o       prusr.o \
1303             prvfsops.o       prvnops.o

1305 MNTFS_OBJS += mntvfsops.o     mntvnops.o

1307 SHAREFS_OBJS += sharetab.o     sharefs_vfsops.o     sharefs_vnops.o

1309 SPEC_OBJS += specsubr.o       specvfsops.o     specvnops.o

1311 SOCK_OBJS += socksubr.o       sockvfsops.o     sockparams.o   \
1312             socksyscalls.o   socktpi.o       sockstr.o \

```

```

1313          sockcommon_vnops.o      sockcommon_subr.o \
1314          sockcommon_sops.o        sockcommon.o      \
1315          sock_notsupp.o            socknotify.o \
1316          nl7c.o                    nl7chttp.o        nl7clogd.o \
1317          nl7cnca.o                 sodirect.o        sockfilter.o

1319 TMPFS_OBJS += tmp_dir.o          tmp_subr.o          tmp_tnode.o        tmp_vfsops.o \
1320          tmp_vnops.o

1322 UDFS_OBJS += udf_alloc.o          udf_bmap.o          udf_dir.o          \
1323          udf_inode.o              udf_subr.o          udf_vfsops.o      \
1324          udf_vnops.o

1326 UFS_OBJS += ufs_alloc.o           ufs_bmap.o          ufs_dir.o          ufs_xattr.o \
1327          ufs_inode.o              ufs_subr.o          ufs_tables.o      ufs_vfsops.o \
1328          ufs_vnops.o              quota.o             quotacalls.o       quota_ufs.o \
1329          ufs_filio.o              ufs_lockfs.o        ufs_thread.o       ufs_trans.o \
1330          ufs_acl.o                 ufs_panic.o         ufs_directio.o    ufs_log.o \
1331          ufs_extvnops.o            ufs_snap.o          lufs.o             lufs_thread.o \
1332          lufs_log.o               lufs_map.o          lufs_top.o         lufs_debug.o
1333 VSCAN_OBJS += vscan_drv.o           vscan_svc.o         vscan_door.o

1335 NSMB_OBJS += smb_conn.o           smb_dev.o           smb_iod.o           smb_pass.o \
1336          smb_rq.o                 smb_sign.o          smb_smb.o           smb_subrs.o \
1337          smb_time.o              smb_tran.o          smb_trantcp.o       smb_usr.o \
1338          subr_mchain.o

1340 SMBFS_COMMON_OBJS += smbfs_ntacl.o
1341 SMBFS_OBJS += smbfs_vfsops.o      smbfs_vnops.o      smbfs_node.o      \
1342          smbfs_acl.o              smbfs_client.o     smbfs_smb.o        \
1343          smbfs_subr.o              smbfs_subr2.o      \
1344          smbfs_rwlock.o            smbfs_xattr.o      \
1345          $(SMBFS_COMMON_OBJS)

1347 BOOTFS_OBJS += bootfs_construct.o bootfs_vfsops.o bootfs_vnops.o

1349 #
1350 #           LVM modules
1351 #
1352 MD_OBJS += md.o md_error.o md_ioctl.o md_mddb.o md_names.o \
1353          md_med.o md_rename.o md_subr.o

1355 MD_COMMON_OBJS = md_convert.o md_crc.o md_revchk.o

1357 MD_DERIVED_OBJS = metamed_xdr.o meta_basic_xdr.o

1359 SOFTPART_OBJS += sp.o sp_ioctl.o

1361 STRIPE_OBJS += stripe.o stripe_ioctl.o

1363 HOTSPARES_OBJS += hotspares.o

1365 RAID_OBJS += raid.o raid_ioctl.o raid_replay.o raid_resync.o raid_hotspare.o

1367 MIRROR_OBJS += mirror.o mirror_ioctl.o mirror_resync.o

1369 NOTIFY_OBJS += md_notify.o

1371 TRANS_OBJS += mdtrans.o trans_ioctl.o trans_log.o

1373 ZFS_COMMON_OBJS += \
1374          arc.o \
1375          blkptr.o \
1376          bplist.o \
1377          bpobj.o \
1378          bptree.o

```

```

1379          bqueue.o \
1380          dbuf.o \
1381          ddt.o \
1382          ddt_zap.o \
1383          dmuf.o \
1384          dmuf_diff.o \
1385          dmuf_send.o \
1386          dmuf_object.o \
1387          dmuf_objset.o \
1388          dmuf_traverse.o \
1389          dmuf_tx.o \
1390          dnode.o \
1391          dnode_sync.o \
1392          dsl_bookmark.o \
1393          dsl_dir.o \
1394          dsl_dataset.o \
1395          dsl_deadlist.o \
1396          dsl_destroy.o \
1397          dsl_pool.o \
1398          dsl_synctask.o \
1399          dsl_userhold.o \
1400          dmuf_zfetch.o \
1401          dsl_deleg.o \
1402          dsl_prop.o \
1403          dsl_scan.o \
1404          zfeature.o \
1405          gzip.o \
1406          lz4.o \
1407          lzjb.o \
1408          metaslab.o \
1409          multilist.o \
1410          range_tree.o \
1411          refcount.o \
1412          rrwlock.o \
1413          sa.o \
1414          sha256.o \
1415          edonr_zfs.o \
1416          skein_zfs.o \
1417          spa.o \
1418          spa_config.o \
1419          spa_errlog.o \
1420          spa_history.o \
1421          spa_misc.o \
1422          space_map.o \
1423          space_reftree.o \
1424          txg.o \
1425          uberblock.o \
1426          unique.o \
1427          vdev.o \
1428          vdev_cache.o \
1429          vdev_file.o \
1430          vdev_label.o \
1431          vdev_mirror.o \
1432          vdev_missing.o \
1433          vdev_queue.o \
1434          vdev_raidz.o \
1435          vdev_root.o \
1436          zap.o \
1437          zap_leaf.o \
1438          zap_micro.o \
1439          zfs_byteswap.o \
1440          zfs_debug.o \
1441          zfs_fm.o \
1442          zfs_fuid.o \
1443          zfs_sa.o \
1444          zfs_znode.o

```

```

1445 zil.o \
1446 zio.o \
1447 zio_checksum.o \
1448 zio_compress.o \
1449 zio_inject.o \
1450 zle.o \
1451 zrlock.o

1453 ZFS_SHARED_OBJS += \
1454 zfeature_common.o \
1455 zfs_comutil.o \
1456 zfs_deleg.o \
1457 zfs_fletcher.o \
1458 zfs_namecheck.o \
1459 zfs_prop.o \
1460 zpool_prop.o \
1461 zprop_common.o

1463 ZFS_OBJS += \
1464 $(ZFS_COMMON_OBJS) \
1465 $(ZFS_SHARED_OBJS) \
1466 vdev_disk.o \
1467 zfs_acl.o \
1468 zfs_ctldir.o \
1469 zfs_dir.o \
1470 zfs_ioctl.o \
1471 zfs_log.o \
1472 zfs_onexit.o \
1473 zfs_replay.o \
1474 zfs_rlock.o \
1475 zfs_vfsops.o \
1476 zfs_vnops.o \
1477 zvol.o

1479 ZUT_OBJS += \
1480 zut.o

1482 #
1483 # streams modules
1484 #
1485 BUFMOD_OBJS += bufmod.o

1487 CONNLD_OBJS += connld.o

1489 DEDUMP_OBJS += dedump.o

1491 DRCOMPAT_OBJS += drcompat.o

1493 LDLINUX_OBJS += ldlinux.o

1495 LDTERM_OBJS += ldterm.o uwidth.o

1497 PCKT_OBJS += pckt.o

1499 PFMOD_OBJS += pfmod.o

1501 PTEM_OBJS += ptem.o

1503 REDIRMOD_OBJS += strredirm.o

1505 TIMOD_OBJS += timod.o

1507 TIRDWR_OBJS += tirdwr.o

1509 TTCOMPAT_OBJS +=ttcompat.o

```

```

1511 LOG_OBJS += log.o

1513 PIPEMOD_OBJS += pipemod.o

1515 RPCMOD_OBJS += rpcmod.o clnt_cots.o clnt_clts.o \
1516 clnt_gen.o clnt_perr.o mt_rpcinit.o rpc_calmsg.o \
1517 rpc_prot.o rpc_sztypes.o rpc_subr.o rpcb_prot.o \
1518 svc.o svc_clts.o svc_gen.o svc_cots.o \
1519 rpcsys.o xdr_sizeof.o clnt_rdma.o svc_rdma.o \
1520 xdr_rdma.o rdma_subr.o xdrdma_sizeof.o

1522 KLMMOD_OBJS += klmmod.o \
1523 nlm_impl.o \
1524 nlm_rpc_handle.o \
1525 nlm_dispatch.o \
1526 nlm_rpc_svc.o \
1527 nlm_client.o \
1528 nlm_service.o \
1529 nlm_prot_clnt.o \
1530 nlm_prot_xdr.o \
1531 nlm_rpc_clnt.o \
1532 nsm_addr_clnt.o \
1533 nsm_addr_xdr.o \
1534 sm_inter_clnt.o \
1535 sm_inter_xdr.o

1537 KLMOPS_OBJS += klmops.o

1539 TLMOD_OBJS += tlmod.o t_kalloc.o t_kbind.o t_kclose.o \
1540 t_kconnect.o t_kfree.o t_kgtstate.o t_kopen.o \
1541 t_krcvudat.o t_ksndudat.o t_kspoll.o t_kunbind.o \
1542 t_kutil.o

1544 RLMOD_OBJS += rlmod.o

1546 TELMOD_OBJS += telmod.o

1548 CRYPTMOD_OBJS += cryptmod.o

1550 KB_OBJS += kbd.o keytables.o

1552 #
1553 # ID mapping module
1554 #
1555 IDMAP_OBJS += idmap_mod.o idmap_kapi.o idmap_xdr.o idmap_cache.o

1557 #
1558 # scheduling class modules
1559 #
1560 SDC_OBJS += sysdc.o

1562 RT_OBJS += rt.o
1563 RT_DPTBL_OBJS += rt_dptbl.o

1565 TS_OBJS += ts.o
1566 TS_DPTBL_OBJS += ts_dptbl.o

1568 IA_OBJS += ia.o

1570 FSS_OBJS += fss.o

1572 FX_OBJS += fx.o
1573 FX_DPTBL_OBJS += fx_dptbl.o

1575 #
1576 # Inter-Process Communication (IPC) modules

```



```

1577 #
1578 IPC_OBJS += ipc.o

1580 IPCMSG_OBJS += msg.o

1582 IPCSEM_OBJS += sem.o

1584 IPCSHM_OBJS += shm.o

1586 #
1587 #         bignum module
1588 #
1589 COMMON_BIGNUM_OBJS += bignum_mod.o bignumimpl.o

1591 BIGNUM_OBJS += $(COMMON_BIGNUM_OBJS) $(BIGNUM_PSR_OBJS)

1593 #
1594 #         kernel cryptographic framework
1595 #
1596 KCF_OBJS += kcf.o kcf_callprov.o kcf_cbufcall.o kcf_cipher.o kcf_crypto.o \
1597 kcf_cryptoadm.o kcf_ctxops.o kcf_digest.o kcf_dual.o \
1598 kcf_keys.o kcf_mac.o kcf_mech_tabs.o kcf_miscapi.o \
1599 kcf_object.o kcf_policy.o kcf_prov_lib.o kcf_prov_tabs.o \
1600 kcf_sched.o kcf_session.o kcf_sign.o kcf_spi.o kcf_verify.o \
1601 kcf_random.o modes.o ecb.o cbc.o ctr.o ccm.o gcm.o \
1602 fips_random.o

1604 CRYPTOADM_OBJS += cryptoadm.o

1606 CRYPTO_OBJS += crypto.o

1608 DPROV_OBJS += dprov.o

1610 DCA_OBJS += dca.o dca_3des.o dca_debug.o dca_dsa.o dca_kstat.o dca_rng.o \
1611 dca_rsa.o

1613 AESPROV_OBJS += aes.o aes_impl.o aes_modes.o

1615 ARCFOURPROV_OBJS += arcfour.o arcfour_crypt.o

1617 BLOWFISHPROV_OBJS += blowfish.o blowfish_impl.o

1619 ECCPROV_OBJS += ecc.o ec.o ec2_163.o ec2_mont.o ecdecode.o ecl_mult.o \
1620 ecp_384.o ecp_jac.o ec2_193.o ecl.o ecp_192.o ecp_521.o \
1621 ecp_jm.o ec2_233.o ecl_curve.o ecp_224.o ecp_aff.o \
1622 ecp_mont.o ec2_aff.o ec_naf.o ecl_gf.o ecp_256.o mp_gf2m.o \
1623 mpi.o mplogic.o mpmontg.o mprime.o oid.o \
1624 secitem.o ec2_test.o ecp_test.o

1626 RSAPROV_OBJS += rsa.o rsa_impl.o pkcs1.o

1628 SWRANDPROV_OBJS += swrand.o

1630 #
1631 #         kernel SSL
1632 #
1633 KSSL_OBJS += kssl.o ksslioctl.o

1635 KSSL_SOCKETFIL_MOD_OBJS += ksslfilter.o ksslapi.o ksslrec.o

1637 #
1638 #         misc. modules
1639 #

1641 C2AUDIT_OBJS += adr.o audit.o audit_event.o audit_io.o \
1642 audit_path.o audit_start.o audit_syscalls.o audit_token.o \

```

```

1643         audit_mem.o

1645 PCIC_OBJS += pcic.o

1647 RPCSEC_OBJS += secmod.o         sec_clnt.o         sec_svc.o         sec_gen.o \
1648 auth_des.o         auth_kern.o         auth_none.o         auth_loopb.o \
1649 authdesprt.o         authdesubr.o         authu_prot.o \
1650 key_call.o         key_prot.o         svc_authu.o         svcauthdes.o

1652 RPCSEC_GSS_OBJS += rpcsec_gssmod.o rpcsec_gss.o rpcsec_gss_misc.o \
1653 rpcsec_gss_utils.o svc_rpcsec_gss.o

1655 CONSCONFIG_OBJS += consconfig.o

1657 CONSCONFIG_DACF_OBJS += consconfig_dacf.o consplat.o

1659 TEM_OBJS += tem.o tem_safe.o 6x10.o 7x14.o 12x22.o

1661 KBTRANS_OBJS += \
1662 kbtrans.o \
1663 kbtrans_keytables.o \
1664 kbtrans_polled.o \
1665 kbtrans_streams.o \
1666 usb_keytables.o

1668 KGSSD_OBJS += gssd_clnt_stubs.o gssd_handle.o gssd_prot.o \
1669 gss_display_name.o gss_release_name.o gss_import_name.o \
1670 gss_release_buffer.o gss_release_oid_set.o gen_oids.o gssdmod.o

1672 KGSSD_DERIVED_OBJS = gssd_xdr.o

1674 KGSS_DUMMY_OBJS += dmecch.o

1676 KSOCKET_OBJS += ksocket.o ksocket_mod.o

1678 CRYPTO= cksumtypes.o decrypt.o encrypt.o encrypt_length.o etypes.o \
1679 nfold.o verify_checksum.o prng.o block_size.o make_checksum.o \
1680 checksum_length.o hmac.o default_state.o mandatory_sumtype.o

1682 # crypto/des
1683 CRYPTO_DES= fcbc.o fcksum.o fparity.o weak_key.o d3cbc.o ef_crypto.o

1685 CRYPTO_DK= checksum.o derive.o dk_decrypt.o dk_encrypt.o

1687 CRYPTO_ARCFOUR= k5_arcfour.o

1689 # crypto/enc_provider
1690 CRYPTO_ENC= des.o des3.o arcfour_provider.o aes_provider.o

1692 # crypto/hash_provider
1693 CRYPTO_HASH= hash_kef_generic.o hash_kmd5.o hash_crc32.o hash_kshal.o

1695 # crypto/keyhash_provider
1696 CRYPTO_KEYHASH= descbc.o k5_kmd5des.o k_hmac_md5.o

1698 # crypto/crc32
1699 CRYPTO_CRC32= crc32.o

1701 # crypto/old
1702 CRYPTO_OLD= old_decrypt.o old_encrypt.o

1704 # crypto/raw
1705 CRYPTO_RAW= raw_decrypt.o raw_encrypt.o

1707 K5_KRB= kfree.o copy_key.o \
1708 parse.o init_ctx.o \

```

```

1709     ser_adata.o ser_addr.o \
1710     ser_auth.o ser_cksum.o \
1711     ser_key.o ser_princ.o \
1712     serialize.o unparse.o \
1713     ser_actx.o

1715 K5_OS=  timeofday.o toffset.o \
1716         init_os_ctx.o c_ustime.o

1718 SEAL=   seal.o unseal.o

1720 MECH=   delete_sec_context.o \
1721         import_sec_context.o \
1722         gssapi_krb5.o \
1723         k5seal.o k5unseal.o k5sealv3.o \
1724         ser_sctx.o \
1725         sign.o \
1726         util_crypt.o \
1727         util_validate.o util_ordering.o \
1728         util_seqnum.o util_set.o util_seed.o \
1729         wrap_size_limit.o verify.o

1733 MECH_GEN= util_token.o

1736 KGSS_KRB5_OBJS += krb5mech.o \
1737     $(MECH) $(SEAL) $(MECH_GEN) \
1738     $(CRYPTO) $(CRYPTO_DES) $(CRYPTO_DK) $(CRYPTO_ARCFOUR) \
1739     $(CRYPTO_ENC) $(CRYPTO_HASH) \
1740     $(CRYPTO_KEYHASH) $(CRYPTO_CRC32) \
1741     $(CRYPTO_OLD) \
1742     $(CRYPTO_RAW) $(K5_KRB) $(K5_OS)

1744 DES_OBJS +=  des_crypt.o des_impl.o des_ks.o des_soft.o

1746 DLBOOT_OBJS +=  bootparam_xdr.o nfs_dlinet.o scan.o

1748 KRTLD_OBJS +=  kobj_bootflags.o getoptstr.o \
1749     kobj.o kobj_kdi.o kobj_lm.o kobj_subr.o

1751 MOD_OBJS +=  modctl.o modsubr.o modsysfile.o modconf.o modhash.o

1753 STRPLUMB_OBJS +=  strplumb.o

1755 CPR_OBJS +=  cpr_driver.o cpr_dump.o \
1756     cpr_main.o cpr_misc.o cpr_mod.o cpr_stat.o \
1757     cpr_uthread.o

1759 PROF_OBJS +=  prf.o

1761 SE_OBJS +=  se_driver.o

1763 SYSACCT_OBJS +=  acct.o

1765 ACCTCTL_OBJS +=  acctctl.o

1767 EXACCTSYS_OBJS +=  exacctsys.o

1769 KATIO_OBJS +=  aio.o

1771 PCMCIA_OBJS +=  pcmcia.o cs.o cis.o cis_callout.o cis_handlers.o cis_params.o

1773 BUSRA_OBJS +=  busra.o

```

```

1775 PCS_OBJS +=  pcs.o

1777 PSET_OBJS +=  pset.o

1779 OHCI_OBJS +=  ohci.o ohci_hub.o ohci_polled.o

1781 UHCI_OBJS +=  uhci.o uhciutil.o uhci_tgt.o uhcihub.o uhcipolled.o

1783 EHCI_OBJS +=  ehci.o ehci_hub.o ehci_xfer.o ehci_intr.o ehci_util.o ehci_polled.o

1785 HUBD_OBJS +=  hubd.o

1787 USB_MID_OBJS +=  usb_mid.o

1789 USB_IA_OBJS +=  usb_ia.o

1791 SCSA2USB_OBJS +=  scsa2usb.o usb_ms_bulkonly.o usb_ms_cbi.o

1793 IPF_OBJS +=  ip_fil_solaris.o fil.o solaris.o ip_state.o ip_frag.o ip_nat.o \
1794     ip_proxy.o ip_auth.o ip_pool.o ip_hstable.o ip_lookup.o \
1795     ip_log.o misc.o ip_compat.o ip_nat6.o drand48.o

1797 IPD_OBJS +=  ipd.o

1799 IBD_OBJS +=  ibd.o ibd_cm.o

1801 EIBNX_OBJS +=  enx_main.o enx_hdlrs.o enx_ibt.o enx_log.o enx_fip.o \
1802     enx_misc.o enx_q.o enx_ctl.o

1804 EOIB_OBJS +=  eib_adm.o eib_chan.o eib_cmn.o eib_ctl.o eib_data.o \
1805     eib_fip.o eib_ibt.o eib_log.o eib_mac.o eib_main.o \
1806     eib_rsrc.o eib_svc.o eib_vnic.o

1808 DLPSTUB_OBJS +=  dlpistub.o

1810 SDP_OBJS +=  sdpddi.o

1812 TRILL_OBJS +=  trill.o

1814 CTF_OBJS +=  ctf_create.o ctf_decl.o ctf_error.o ctf_hash.o ctf_labels.o \
1815     ctf_lookup.o ctf_open.o ctf_types.o ctf_util.o ctf_subr.o ctf_mod.o

1817 SMBIOS_OBJS +=  smb_error.o smb_info.o smb_open.o smb_subr.o smb_dev.o

1819 RPCIB_OBJS +=  rpcib.o

1821 KMDB_OBJS +=  kdrv.o

1823 AFE_OBJS +=  afe.o

1825 BGE_OBJS +=  bge_main2.o bge_chip2.o bge_kstats.o bge_log.o bge_ndd.o \
1826     bge_atomic.o bge_mii.o bge_send.o bge_recv2.o bge_mii_5906.o

1828 DMFE_OBJS +=  dmfe_log.o dmfe_main.o dmfe_mii.o

1830 EFE_OBJS +=  efe.o

1832 ELXL_OBJS +=  elxl.o

1834 HME_OBJS +=  hme.o

1836 IXGB_OBJS +=  ixgb.o ixgb_atomic.o ixgb_chip.o ixgb_gld.o ixgb_kstats.o \
1837     ixgb_log.o ixgb_ndd.o ixgb_rx.o ixgb_tx.o ixgb_xmii.o

1839 NGE_OBJS +=  nge_main.o nge_atomic.o nge_chip.o nge_ndd.o nge_kstats.o \
1840     nge_log.o nge_rx.o nge_tx.o nge_xmii.o

```

```

1842 PCN_OBJS += pcn.o
1844 RGE_OBJS += rge_main.o rge_chip.o rge_ndd.o rge_kstats.o rge_log.o rge_rxtx.o
1846 URTW_OBJS += urtw.o
1848 ARN_OBJS += arn_hw.o arn_eeprom.o arn_mac.o arn_calib.o arn_ani.o arn_phy.o arn_
1849 arn_main.o arn_recv.o arn_xmit.o arn_rc.o
1851 ATH_OBJS += ath_aux.o ath_main.o ath_osdep.o ath_rate.o
1853 ATU_OBJS += atu.o
1855 IPW_OBJS += ipw2100_hw.o ipw2100.o
1857 IWI_OBJS += ipw2200_hw.o ipw2200.o
1859 IWH_OBJS += iwh.o
1861 IWK_OBJS += iwk2.o
1863 IWP_OBJS += iwp.o
1865 MWL_OBJS += mwl.o
1867 MWLFW_OBJS += mwlfw_mode.o
1869 WPI_OBJS += wpi.o
1871 RAL_OBJS += rt2560.o ral_rate.o
1873 RUM_OBJS += rum.o
1875 RWD_OBJS += rt2661.o
1877 RWN_OBJS += rt2860.o
1879 UATH_OBJS += uath.o
1881 UATHFW_OBJS += uathfw_mod.o
1883 URAL_OBJS += ural.o
1885 RTW_OBJS += rtw.o smc93cx6.o rtwphy.o rtwphyio.o
1887 ZYD_OBJS += zyd.o zyd_usb.o zyd_hw.o zyd_fw.o
1889 MXFE_OBJS += mxfe.o
1891 MPTSAS_OBJS += mptsas.o mptsas_hash.o mptsas_impl.o mptsas_init.o \
1892 mptsas_raid.o mptsas_smhba.o
1894 SFE_OBJS += sfe.o sfe_util.o
1896 BFE_OBJS += bfe.o
1898 BRIDGE_OBJS += bridge.o
1900 IDM_SHARED_OBJS += base64.o
1902 IDM_OBJS += $(IDM_SHARED_OBJS) \
1903 idm.o idm_impl.o idm_text.o idm_conn_sm.o idm_so.o
1905 VR_OBJS += vr.o

```

```

1907 ATGE_OBJS += atge_main.o atge_lle.o atge_mii.o atge_ll.o atge_llc.o
1909 YGE_OBJS = yge.o
1911 SKD_OBJS = skd.o
1913 NVME_OBJS = nvme.o
1915 #
1916 #     Build up defines and paths.
1917 #
1918 LINT_DEFS      += -Dunix
1920 #
1921 #     This duality can be removed when the native and target compilers
1922 #     are the same (or at least recognize the same command line syntax!)
1923 #     It is a bug in the current compilation system that the assembler
1924 #     can't process the -Y I, flag.
1925 #
1926 NATIVE_INC_PATH += $(INC_PATH) $(CCYFLAG)$(UTSBASE)/common
1927 AS_INC_PATH     += $(INC_PATH) -I$(UTSBASE)/common
1928 INCLUDE_PATH    += $(INC_PATH) $(CCYFLAG)$(UTSBASE)/common
1930 PCIEB_OBJS += pcieb.o
1932 #     Chelsio N110 10G NIC driver module
1933 #
1934 CH_OBJS = ch.o glue.o pe.o sge.o
1936 CH_COM_OBJS = ch_mac.o ch_subr.o cspi.o espi.o ixfl1010.o mc3.o mc4.o mc5.o \
1937 mv88e1xxx.o mv88x201x.o my3126.o pm3393.o tp.o ulp.o \
1938 vsc7321.o vsc7326.o xpak.o
1940 #
1941 #     Chelsio Terminator 4 10G NIC nexus driver module
1942 #
1943 CXGBE_FW_OBJS = t4_fw.o t4_cfg.o
1944 CXGBE_COM_OBJS = t4_hw.o common.o
1945 CXGBE_NEX_OBJS = t4_nexus.o t4_sge.o t4_mac.o t4_ioctl.o shared.o \
1946 t4_l2t.o adapter.o osdep.o
1948 #
1949 #     Chelsio Terminator 4 10G NIC driver module
1950 #
1951 CXGBE_OBJS = cxgbe.o
1953 #
1954 #     PCI strings file
1955 #
1956 PCI_STRING_OBJS = pci_strings.o
1958 NET_DACF_OBJS += net_dacf.o
1960 #
1961 #     Xframe 10G NIC driver module
1962 #
1963 XGE_OBJS = xge.o xgell.o
1965 XGE_HAL_OBJS = xgehal-channel.o xgehal-fifo.o xgehal-ring.o xgehal-config.o \
1966 xgehal-driver.o xgehal-mm.o xgehal-stats.o xgehal-device.o \
1967 xge-queue.o xgehal-mgmt.o xgehal-mgmtaux.o
1969 #
1970 #     e1000/igb common objs
1971 #
1972 #     Historically e1000g and igb had separate copies of all of the common

```

```

1973 # code. At this time while they are now sharing the same copy of it, they
1974 # are building it into their own modules which is due to the differences
1975 # in the osdep and debug portions of their code.
1976 #
1977 E1000API_OBJS += e1000_80003es2lan.o e1000_82540.o e1000_82541.o e1000_82542.o \
1978 e1000_82543.o e1000_82571.o e1000_api.o e1000_ich8lan.o \
1979 e1000_mac.o e1000_manage.o e1000_nvm.o e1000_phy.o \
1980 e1000_82575.o e1000_i210.o e1000_mbx.o e1000_vf.o
1982 #
1983 # e1000g module
1984 #
1985 E1000G_OBJS += e1000g_debug.o e1000g_main.o e1000g_alloc.o \
1986 e1000g_tx.o e1000g_rx.o e1000g_stat.o \
1987 e1000g_osdep.o e1000g_workarounds.o
1990 #
1991 # Intel 82575 1G NIC driver module
1992 #
1993 IGB_OBJS = igb_buf.o igb_debug.o igb_gld.o igb_log.o igb_main.o \
1994 igb_rx.o igb_stat.o igb_tx.o igb_osdep.o
1996 #
1997 # Intel Pro/100 NIC driver module
1998 #
1999 IPRB_OBJS = iprb.o
2001 #
2002 # Intel 10GbE PCIE NIC driver module
2003 #
2004 IXGBE_OBJS = ixgbe_82598.o ixgbe_82599.o ixgbe_api.o \
2005 ixgbe_common.o ixgbe_phy.o \
2006 ixgbe_buf.o ixgbe_debug.o ixgbe_gld.o \
2007 ixgbe_log.o ixgbe_main.o \
2008 ixgbe_osdep.o ixgbe_rx.o ixgbe_stat.o \
2009 ixgbe_tx.o ixgbe_x540.o ixgbe_mbx.o
2011 #
2012 # Solarflare 1/10/40GbE NIC driver module
2013 #
2014 # NB: The illumos specific sources are listed first, with the
2015 # common (OS-independent) sources afterwards.
2016 #
2017 SFXGE_OBJS = sfxge_err.o sfxge_ev.o sfxge_hash.o sfxge_intr.o sfxge_mac.o \
2018 sfxge_gld_v3.o sfxge_mon.o sfxge_phy.o \
2019 sfxge_sram.o sfxge_bar.o sfxge_pci.o sfxge_nvram.o \
2020 sfxge_rx.o sfxge_tcp.o sfxge_tx.o sfxge_mcdi.o sfxge_vpd.o \
2021 sfxge.o sfxge_dma.o
2022 SFXGE_SF_OBJS = efx_bootcfg.o efx_crc32.o efx_ev.o efx_filter.o \
2023 efx_hash.o efx_intr.o efx_mac.o efx_mcdi.o efx_mon.o \
2024 efx_nic.o efx_nvram.o efx_phy.o efx_port.o efx_rx.o \
2025 efx_sram.o efx_tx.o efx_vpd.o efx_wol.o mcdi_mon.o \
2026 siena_mac.o siena_mcdi.o siena_nic.o siena_nvram.o \
2027 siena_phy.o siena_sram.o siena_vpd.o \
2028 ef10_ev.o ef10_filter.o ef10_intr.o ef10_mac.o ef10_mcdi.o \
2029 ef10_nic.o ef10_nvram.o ef10_phy.o ef10_rx.o ef10_tx.o \
2030 ef10_vpd.o hunt_nic.o hunt_phy.o
2032 #
2033 # NIU 10G/1G driver module
2034 #
2035 NXGE_OBJS = nxge_mac.o nxge_ipp.o nxge_rxdma.o \
2036 nxge_txdma.o nxge_txc.o nxge_main.o \
2037 nxge_hw.o nxge_fzc.o nxge_virtual.o \
2038 nxge_send.o nxge_classify.o nxge_fflp.o \

```

```

2039 nxge_fflp_hash.o nxge_ndd.o nxge_kstats.o \
2040 nxge_zcp.o nxge_fm.o nxge_espc.o nxge_hv.o \
2041 nxge_hio.o nxge_hio_guest.o nxge_intr.o
2043 NXGE_NPI_OBJS = \
2044 napi.o napi_mac.o napi_ipp.o \
2045 napi_txdma.o napi_rxdma.o napi_txc.o \
2046 napi_zcp.o napi_espc.o napi_fflp.o \
2047 napi_vir.o
2049 NXGE_HCALL_OBJS = \
2050 nxge_hcall.o
2052 #
2053 # Virtio modules
2054 #
2056 # Virtio core
2057 VIRTIO_OBJS = virtio.o
2059 # Virtio block driver
2060 VIOBLK_OBJS = vioblk.o
2062 # Virtio network driver
2063 VIOIF_OBJS = vioif.o
2065 #
2066 # kiconv modules
2067 #
2068 KICONV_EMEA_OBJS += kiconv_emea.o
2070 KICONV_JA_OBJS += kiconv_ja.o
2072 KICONV_KO_OBJS += kiconv_cck_common.o kiconv_ko.o
2074 KICONV_SC_OBJS += kiconv_cck_common.o kiconv_sc.o
2076 KICONV_TC_OBJS += kiconv_cck_common.o kiconv_tc.o
2078 #
2079 # AAC module
2080 #
2081 AAC_OBJS = aac.o aac_ioctl.o
2083 #
2084 # sdcard modules
2085 #
2086 SDA_OBJS = sda_cmd.o sda_host.o sda_init.o sda_mem.o sda_mod.o sda_slot.o
2087 SDHOST_OBJS = sdhost.o
2089 #
2090 # hxge 10G driver module
2091 #
2092 HXGE_OBJS = hxge_main.o hxge_vmac.o hxge_send.o \
2093 hxge_txdma.o hxge_rxdma.o hxge_virtual.o \
2094 hxge_fm.o hxge_fzc.o hxge_hw.o hxge_kstats.o \
2095 hxge_ndd.o hxge_pfc.o \
2096 hpi.o hpi_vmac.o hpi_rxdma.o hpi_txdma.o \
2097 hpi_vir.o hpi_pfc.o
2099 #
2100 # MEGARAID_SAS module
2101 #
2102 MEGA_SAS_OBJS = megaraid_sas.o
2104 #

```

```

2105 # MR_SAS module
2106 #
2107 MR_SAS_OBJS = ld_pd_map.o mr_sas.o mr_sas_tbolt.o mr_sas_list.o

2109 #
2110 # CPQARY3 module
2111 #
2112 CPQARY3_OBJS = cpqary3.o cpqary3_noe.o cpqary3_talk2ctrl.o \
2113 cpqary3_isr.o cpqary3_transport.o cpqary3_mem.o \
2114 cpqary3_scsi.o cpqary3_util.o cpqary3_ioctl.o \
2115 cpqary3_bd.o

2117 #
2118 # ISCSI_INITIATOR module
2119 #
2120 ISCSI_INITIATOR_OBJS = chap.o iscsi_io.o iscsi_thread.o \
2121 iscsi_ioctl.o iscsid.o iscsi.o \
2122 iscsi_login.o isns_client.o iscsiAuthClient.o \
2123 iscsi_lun.o iscsiAuthClientGlue.o \
2124 iscsi_net.o nvfile.o iscsi_cmd.o \
2125 iscsi_queue.o persistent.o iscsi_conn.o \
2126 iscsi_sess.o radius_auth.o iscsi_crc.o \
2127 iscsi_stats.o radius_packet.o iscsi_doorclt.o \
2128 iscsi_targetparam.o utils.o kifconf.o

2130 #
2131 # ntxn 10Gb/1Gb NIC driver module
2132 #
2133 NTXN_OBJS = unm_nic_init.o unm_gem.o unm_nic_hw.o unm_ndd.o \
2134 unm_nic_main.o unm_nic_isr.o unm_nic_ctx.o niu.o

2136 #
2137 # Myricom 10Gb NIC driver module
2138 #
2139 MYRI10GE_OBJS = myri10ge.o myri10ge_lro.o

2141 #
2142 # nulldriver module
2143 NULLDRIVER_OBJS = nulldriver.o

2145 TPM_OBJS = tpm.o tpm_hcall.o

2147 #
2148 # BNXE objects
2149 #
2150 BNXE_OBJS += bnxe_cfg.o \
2151 bnxe_fcoe.o \
2152 bnxe_debug.o \
2153 bnxe_gld.o \
2154 bnxe_hw.o \
2155 bnxe_intr.o \
2156 bnxe_kstat.o \
2157 bnxe_lock.o \
2158 bnxe_main.o \
2159 bnxe_mm.o \
2160 bnxe_mm_14.o \
2161 bnxe_mm_15.o \
2162 bnxe_rr.o \
2163 bnxe_rx.o \
2164 bnxe_timer.o \
2165 bnxe_tx.o \
2166 bnxe_workq.o \
2167 bnxe_clc.o \
2168 ecore_sp_verbs.o \
2169 bnxe_context.o \
2170 57710_init_values.o

```

```

2171 57711_init_values.o \
2172 57712_init_values.o \
2173 bnxe_fw_funcs.o \
2174 bnxe_hw_debug.o \
2175 lm_14fp.o \
2176 lm_14rx.o \
2177 lm_14sp.o \
2178 lm_14tx.o \
2179 lm_15.o \
2180 lm_15sp.o \
2181 lm_dcbx.o \
2182 lm_devinfo.o \
2183 lm_dmae.o \
2184 lm_er.o \
2185 lm_hw_access.o \
2186 lm_hw_attn.o \
2187 lm_hw_init_reset.o \
2188 lm_main.o \
2189 lm_mcp.o \
2190 lm_niv.o \
2191 lm_nvram.o \
2192 lm_phy.o \
2193 lm_power.o \
2194 lm_rcv.o \
2195 lm_resc.o \
2196 lm_sb.o \
2197 lm_send.o \
2198 lm_sp.o \
2199 lm_dcbx_mp.o \
2200 lm_sp_req_mgr.o \
2201 lm_stats.o \
2202 lm_util.o

```

```

*****
75293 Thu Jun 30 22:03:10 2016
new/usr/src/uts/common/Makefile.rules
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
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 #
23 # Copyright (c) 1991, 2010, Oracle and/or its affiliates. All rights reserved.
24 # Copyright 2016 Garrett D'Amore <garrett@damore.org>
25 # Copyright 2015 Nexenta Systems, Inc. All rights reserved.
26 # Copyright 2013 Saso Kiselkov. All rights reserved.
27 #
28 #
29 #
30 # uts/common/Makefile.rules
31 #
32 # This Makefile defines all the file build rules for the directory
33 # uts/common and its children. These are the source files which may
34 # be considered common to all SunOS systems.
35 #
36 # The following two-level ordering must be maintained in this file.
37 # Lines are sorted first in order of decreasing specificity based on
38 # the first directory component. That is, sun4u rules come before
39 # sparc rules come before common rules.
40 #
41 # Lines whose initial directory components are equal are sorted
42 # alphabetically by the remaining components.
43 #
44 #
45 # Section 1a: C objects build rules
46 #
47 $(OBJS_DIR)/%.o: $(COMMONBASE)/crypto/aes/%.c
48 $(COMPILE.c) -o $@ $<
49 $(CTFCONVERT_O)
50 #
51 $(OBJS_DIR)/%.o: $(COMMONBASE)/crypto/arcfour/%.c
52 $(COMPILE.c) -o $@ $<
53 $(CTFCONVERT_O)
54 #
55 $(OBJS_DIR)/%.o: $(COMMONBASE)/crypto/blowfish/%.c
56 $(COMPILE.c) -o $@ $<
57 $(CTFCONVERT_O)

```

```

59 $(OBJS_DIR)/%.o: $(COMMONBASE)/crypto/ecc/%.c
60 $(COMPILE.c) -o $@ $<
61 $(CTFCONVERT_O)
62 #
63 $(OBJS_DIR)/%.o: $(COMMONBASE)/crypto/modes/%.c
64 $(COMPILE.c) -o $@ $<
65 $(CTFCONVERT_O)
66 #
67 $(OBJS_DIR)/%.o: $(COMMONBASE)/crypto/padding/%.c
68 $(COMPILE.c) -o $@ $<
69 $(CTFCONVERT_O)
70 #
71 $(OBJS_DIR)/%.o: $(COMMONBASE)/crypto/rng/%.c
72 $(COMPILE.c) -o $@ $<
73 $(CTFCONVERT_O)
74 #
75 $(OBJS_DIR)/%.o: $(COMMONBASE)/crypto/rsa/%.c
76 $(COMPILE.c) -o $@ $<
77 $(CTFCONVERT_O)
78 #
79 $(OBJS_DIR)/%.o: $(COMMONBASE)/bignum/%.c
80 $(COMPILE.c) -o $@ $<
81 $(CTFCONVERT_O)
82 #
83 $(OBJS_DIR)/%.o: $(UTSBASE)/common/bignum/%.c
84 $(COMPILE.c) -o $@ $<
85 $(CTFCONVERT_O)
86 #
87 $(OBJS_DIR)/%.o: $(COMMONBASE)/mpi/%.c
88 $(COMPILE.c) -o $@ $<
89 $(CTFCONVERT_O)
90 #
91 $(OBJS_DIR)/%.o: $(COMMONBASE)/acl/%.c
92 $(COMPILE.c) -o $@ $<
93 $(CTFCONVERT_O)
94 #
95 $(OBJS_DIR)/%.o: $(COMMONBASE)/avl/%.c
96 $(COMPILE.c) -o $@ $<
97 $(CTFCONVERT_O)
98 #
99 $(OBJS_DIR)/%.o: $(COMMONBASE)/ucode/%.c
100 $(COMPILE.c) -o $@ $<
101 $(CTFCONVERT_O)
102 #
103 $(OBJS_DIR)/%.o: $(UTSBASE)/common/brand/snl/%.c
104 $(COMPILE.c) -o $@ $<
105 $(CTFCONVERT_O)
106 #
107 $(OBJS_DIR)/%.o: $(UTSBASE)/common/brand/solaris10/%.c
108 $(COMPILE.c) -o $@ $<
109 $(CTFCONVERT_O)
110 #
111 $(OBJS_DIR)/%.o: $(UTSBASE)/common/c2/%.c
112 $(COMPILE.c) -o $@ $<
113 $(CTFCONVERT_O)
114 #
115 $(OBJS_DIR)/%.o: $(UTSBASE)/common/conf/%.c
116 $(COMPILE.c) -o $@ $<
117 $(CTFCONVERT_O)
118 #
119 $(OBJS_DIR)/%.o: $(UTSBASE)/common/contract/%.c
120 $(COMPILE.c) -o $@ $<
121 $(CTFCONVERT_O)
122 #
123 $(OBJS_DIR)/%.o: $(UTSBASE)/common/cpr/%.c
124 $(COMPILE.c) -o $@ $<

```

```

125     $(CTFCONVERT_O)

127 $(OBJSDIR)/%.o:                $(UTSBASE)/common/ctf/%.c
128     $(COMPILE.c) -o $@ $<
129     $(CTFCONVERT_O)

131 $(OBJSDIR)/%.o:                $(COMMONBASE)/ctf/%.c
132     $(COMPILE.c) -o $@ $<
133     $(CTFCONVERT_O)

135 $(OBJSDIR)/%.o:                $(COMMONBASE)/crypto/des/%.c
136     $(COMPILE.c) -o $@ $<
137     $(CTFCONVERT_O)

139 $(OBJSDIR)/%.o:                $(COMMONBASE)/secflags/%.c
140     $(COMPILE.c) -o $@ $<
141     $(CTFCONVERT_O)

143 #endif /* ! codereview */
144 $(OBJSDIR)/%.o:                $(COMMONBASE)/smbios/%.c
145     $(COMPILE.c) -o $@ $<
146     $(CTFCONVERT_O)

148 $(OBJSDIR)/%.o:                $(UTSBASE)/common/des/%.c
149     $(COMPILE.c) -o $@ $<
150     $(CTFCONVERT_O)

152 $(OBJSDIR)/%.o:                $(UTSBASE)/common/crypto/api/%.c
153     $(COMPILE.c) -o $@ $<
154     $(CTFCONVERT_O)

156 $(OBJSDIR)/%.o:                $(UTSBASE)/common/crypto/core/%.c
157     $(COMPILE.c) -o $@ $<
158     $(CTFCONVERT_O)

160 $(OBJSDIR)/%.o:                $(UTSBASE)/common/crypto/io/%.c
161     $(COMPILE.c) -o $@ $<
162     $(CTFCONVERT_O)

164 $(OBJSDIR)/%.o:                $(UTSBASE)/common/crypto/spi/%.c
165     $(COMPILE.c) -o $@ $<
166     $(CTFCONVERT_O)

168 $(OBJSDIR)/%.o:                $(COMMONBASE)/pci/%.c
169     $(COMPILE.c) -o $@ $<
170     $(CTFCONVERT_O)

172 $(OBJSDIR)/%.o:                $(COMMONBASE)/devid/%.c
173     $(COMPILE.c) -o $@ $<
174     $(CTFCONVERT_O)

176 $(OBJSDIR)/%.o:                $(UTSBASE)/common/disp/%.c
177     $(COMPILE.c) -o $@ $<
178     $(CTFCONVERT_O)

180 $(OBJSDIR)/%.o:                $(UTSBASE)/common/dtrace/%.c
181     $(COMPILE.c) -o $@ $<
182     $(CTFCONVERT_O)

184 $(OBJSDIR)/%.o:                $(COMMONBASE)/execct/%.c
185     $(COMPILE.c) -o $@ $<
186     $(CTFCONVERT_O)

188 $(OBJSDIR)/%.o:                $(UTSBASE)/common/exec/aout/%.c
189     $(COMPILE.c) -o $@ $<
190     $(CTFCONVERT_O)

```

```

192 $(OBJSDIR)/%.o:                $(UTSBASE)/common/exec/elf/%.c
193     $(COMPILE.c) -o $@ $<
194     $(CTFCONVERT_O)

196 $(OBJSDIR)/%.o:                $(UTSBASE)/common/exec/intp/%.c
197     $(COMPILE.c) -o $@ $<
198     $(CTFCONVERT_O)

200 $(OBJSDIR)/%.o:                $(UTSBASE)/common/exec/shbin/%.c
201     $(COMPILE.c) -o $@ $<
202     $(CTFCONVERT_O)

204 $(OBJSDIR)/%.o:                $(UTSBASE)/common/exec/java/%.c
205     $(COMPILE.c) -o $@ $<
206     $(CTFCONVERT_O)

208 $(OBJSDIR)/%.o:                $(UTSBASE)/common/fs/%.c
209     $(COMPILE.c) -o $@ $<
210     $(CTFCONVERT_O)

212 $(OBJSDIR)/%.o:                $(UTSBASE)/common/fs/autofs/%.c
213     $(COMPILE.c) -o $@ $<
214     $(CTFCONVERT_O)

216 $(OBJSDIR)/%.o:                $(UTSBASE)/common/fs/bootfs/%.c
217     $(COMPILE.c) -o $@ $<
218     $(CTFCONVERT_O)

220 $(OBJSDIR)/%.o:                $(UTSBASE)/common/fs/dcfis/%.c
221     $(COMPILE.c) -o $@ $<
222     $(CTFCONVERT_O)

224 $(OBJSDIR)/%.o:                $(UTSBASE)/common/fs/devfs/%.c
225     $(COMPILE.c) -o $@ $<
226     $(CTFCONVERT_O)

228 $(OBJSDIR)/%.o:                $(UTSBASE)/common/fs/ctfs/%.c
229     $(COMPILE.c) -o $@ $<
230     $(CTFCONVERT_O)

232 $(OBJSDIR)/%.o:                $(UTSBASE)/common/fs/doorfs/%.c
233     $(COMPILE.c) -o $@ $<
234     $(CTFCONVERT_O)

236 $(OBJSDIR)/%.o:                $(UTSBASE)/common/fs/dev/%.c
237     $(COMPILE.c) -o $@ $<
238     $(CTFCONVERT_O)

240 $(OBJSDIR)/%.o:                $(UTSBASE)/common/fs/fd/%.c
241     $(COMPILE.c) -o $@ $<
242     $(CTFCONVERT_O)

244 $(OBJSDIR)/%.o:                $(UTSBASE)/common/fs/fifofs/%.c
245     $(COMPILE.c) -o $@ $<
246     $(CTFCONVERT_O)

248 $(OBJSDIR)/%.o:                $(UTSBASE)/common/fs/hsfs/%.c
249     $(COMPILE.c) -o $@ $<
250     $(CTFCONVERT_O)

252 $(OBJSDIR)/%.o:                $(UTSBASE)/common/fs/lofs/%.c
253     $(COMPILE.c) -o $@ $<
254     $(CTFCONVERT_O)

256 $(OBJSDIR)/%.o:                $(UTSBASE)/common/fs/mntfs/%.c

```

```

257 $(COMPILE.c) -o $@ $<
258 $(CTFCONVERT_O)

260 $(OBJDIR)/%.o: $(UTSBASE)/common/fs/namefs/%.c
261 $(COMPILE.c) -o $@ $<
262 $(CTFCONVERT_O)

264 $(OBJDIR)/%.o: $(UTSBASE)/common/fs/nfs/%.c
265 $(COMPILE.c) -o $@ $<
266 $(CTFCONVERT_O)

268 $(OBJDIR)/%.o: $(COMMONBASE)/smbdrv/%.c
269 $(COMPILE.c) -o $@ $<
270 $(CTFCONVERT_O)

272 $(OBJDIR)/%.o: $(UTSBASE)/common/fs/smbdrv/%.c
273 $(COMPILE.c) -o $@ $<
274 $(CTFCONVERT_O)

276 $(OBJDIR)/%.o: $(UTSBASE)/common/fs/objfs/%.c
277 $(COMPILE.c) -o $@ $<
278 $(CTFCONVERT_O)

280 $(OBJDIR)/%.o: $(UTSBASE)/common/fs/pcfs/%.c
281 $(COMPILE.c) -o $@ $<
282 $(CTFCONVERT_O)

284 $(OBJDIR)/%.o: $(UTSBASE)/common/fs/portfs/%.c
285 $(COMPILE.c) -o $@ $<
286 $(CTFCONVERT_O)

288 $(OBJDIR)/%.o: $(UTSBASE)/common/fs/proc/%.c
289 $(COMPILE.c) -o $@ $<
290 $(CTFCONVERT_O)

292 $(OBJDIR)/%.o: $(UTSBASE)/common/fs/sharefs/%.c
293 $(COMPILE.c) -o $@ $<
294 $(CTFCONVERT_O)

296 $(OBJDIR)/%.o: $(COMMONBASE)/smbclnt/%.c
297 $(COMPILE.c) -o $@ $<
298 $(CTFCONVERT_O)

300 $(OBJDIR)/%.o: $(UTSBASE)/common/fs/smbclnt/net smb/%.c
301 $(COMPILE.c) -o $@ $<
302 $(CTFCONVERT_O)

304 $(OBJDIR)/%.o: $(UTSBASE)/common/fs/smbclnt/smbfs/%.c
305 $(COMPILE.c) -o $@ $<
306 $(CTFCONVERT_O)

308 $(OBJDIR)/%.o: $(UTSBASE)/common/fs/sockfs/%.c
309 $(COMPILE.c) -o $@ $<
310 $(CTFCONVERT_O)

312 $(OBJDIR)/%.o: $(UTSBASE)/common/fs/specfs/%.c
313 $(COMPILE.c) -o $@ $<
314 $(CTFCONVERT_O)

316 $(OBJDIR)/%.o: $(UTSBASE)/common/fs/swapfs/%.c
317 $(COMPILE.c) -o $@ $<
318 $(CTFCONVERT_O)

320 $(OBJDIR)/%.o: $(UTSBASE)/common/fs/tmpfs/%.c
321 $(COMPILE.c) -o $@ $<
322 $(CTFCONVERT_O)

```

```

324 $(OBJDIR)/%.o: $(UTSBASE)/common/fs/udfs/%.c
325 $(COMPILE.c) -o $@ $<
326 $(CTFCONVERT_O)

328 $(OBJDIR)/%.o: $(UTSBASE)/common/fs/ufs/%.c
329 $(COMPILE.c) -o $@ $<
330 $(CTFCONVERT_O)

332 $(OBJDIR)/%.o: $(UTSBASE)/common/io/vscan/%.c
333 $(COMPILE.c) -o $@ $<
334 $(CTFCONVERT_O)

336 $(OBJDIR)/%.o: $(UTSBASE)/common/fs/zfs/%.c
337 $(COMPILE.c) -o $@ $<
338 $(CTFCONVERT_O)

340 $(OBJDIR)/%.o: $(UTSBASE)/common/fs/zut/%.c
341 $(COMPILE.c) -o $@ $<
342 $(CTFCONVERT_O)

344 $(OBJDIR)/%.o: $(COMMONBASE)/xattr/%.c
345 $(COMPILE.c) -o $@ $<
346 $(CTFCONVERT_O)

348 $(OBJDIR)/%.o: $(COMMONBASE)/zfs/%.c
349 $(COMPILE.c) -o $@ $<
350 $(CTFCONVERT_O)

352 $(OBJDIR)/%.o: $(UTSBASE)/common/io/scsi/adapters/pmcs/%.c
353 $(COMPILE.c) -o $@ $<
354 $(CTFCONVERT_O)

356 $(OBJDIR)/%.o: $(UTSBASE)/common/io/scsi/adapters/pmcs/%.bin
357 $(COMPILE.b) -o $@ $<
358 $(CTFCONVERT_O)

360 $(OBJDIR)/%.o: $(COMMONBASE)/fsreparse/%.c
361 $(COMPILE.c) -o $@ $<
362 $(CTFCONVERT_O)

364 KMECHKRB5_BASE=$(UTSBASE)/common/gssapi/mechs/krb5

366 KGSSDFLAGS=-I $(UTSBASE)/common/gssapi/include

368 # Note, KRB5_DEFS can be assigned various preprocessor flags,
369 # typically -D defines on the make invocation. The standard compiler
370 # flags will not be overwritten.
371 KGSSDFLAGS += $(KRB5_DEFS)

373 $(OBJDIR)/%.o: $(UTSBASE)/common/gssapi/%.c
374 $(COMPILE.c) $(KGSSDFLAGS) -o $@ $<
375 $(CTFCONVERT_O)

377 $(OBJDIR)/%.o: $(UTSBASE)/common/gssapi/mechs/dummy/%.c
378 $(COMPILE.c) $(KGSSDFLAGS) -o $@ $<
379 $(CTFCONVERT_O)

381 $(OBJDIR)/%.o: $(KMECHKRB5_BASE)/%.c
382 $(COMPILE.c) $(KGSSDFLAGS) -o $@ $<
383 $(CTFCONVERT_O)

385 $(OBJDIR)/%.o: $(KMECHKRB5_BASE)/crypto/%.c
386 $(COMPILE.c) $(KGSSDFLAGS) -o $@ $<
387 $(CTFCONVERT_O)

```



```

389 $(OBJS_DIR)/%.o: $(KMECHKRB5_BASE)/crypto/des/%.c
390 $(COMPILE.c) $(KGSSDFLAGS) -o $@ $<
391 $(CTFCONVERT_O)

393 $(OBJS_DIR)/%.o: $(KMECHKRB5_BASE)/crypto/arcfour/%.c
394 $(COMPILE.c) $(KGSSDFLAGS) -o $@ $<
395 $(CTFCONVERT_O)

397 $(OBJS_DIR)/%.o: $(KMECHKRB5_BASE)/crypto/dk/%.c
398 $(COMPILE.c) $(KGSSDFLAGS) -o $@ $<
399 $(CTFCONVERT_O)

401 $(OBJS_DIR)/%.o: $(KMECHKRB5_BASE)/crypto/enc_provider/%.c
402 $(COMPILE.c) $(KGSSDFLAGS) -o $@ $<
403 $(CTFCONVERT_O)

405 $(OBJS_DIR)/%.o: $(KMECHKRB5_BASE)/crypto/hash_provider/%.c
406 $(COMPILE.c) $(KGSSDFLAGS) -o $@ $<
407 $(CTFCONVERT_O)

409 $(OBJS_DIR)/%.o: $(KMECHKRB5_BASE)/crypto/keyhash_provider/%.c
410 $(COMPILE.c) $(KGSSDFLAGS) -o $@ $<
411 $(CTFCONVERT_O)

413 $(OBJS_DIR)/%.o: $(KMECHKRB5_BASE)/crypto/raw/%.c
414 $(COMPILE.c) $(KGSSDFLAGS) -o $@ $<
415 $(CTFCONVERT_O)

417 $(OBJS_DIR)/%.o: $(KMECHKRB5_BASE)/crypto/old/%.c
418 $(COMPILE.c) $(KGSSDFLAGS) -o $@ $<
419 $(CTFCONVERT_O)

421 $(OBJS_DIR)/%.o: $(KMECHKRB5_BASE)/krb5/krb/%.c
422 $(COMPILE.c) $(KGSSDFLAGS) -o $@ $<
423 $(CTFCONVERT_O)

425 $(OBJS_DIR)/%.o: $(KMECHKRB5_BASE)/krb5/os/%.c
426 $(COMPILE.c) $(KGSSDFLAGS) -o $@ $<
427 $(CTFCONVERT_O)

429 $(OBJS_DIR)/ser_sctx.o := CPPFLAGS += -DPROVIDE_KERNEL_IMPORT=1

431 $(OBJS_DIR)/%.o: $(KMECHKRB5_BASE)/mech/%.c
432 $(COMPILE.c) $(KGSSDFLAGS) -o $@ $<
433 $(CTFCONVERT_O)

435 $(OBJS_DIR)/%.o: $(KMECHKRB5_BASE)/profile/%.c
436 $(COMPILE.c) $(KGSSDFLAGS) -o $@ $<
437 $(CTFCONVERT_O)

439 $(OBJS_DIR)/%.o: $(UTSBASE)/common/avs/ncall/%.c
440 $(COMPILE.c) -o $@ $<
441 $(CTFCONVERT_O)

443 $(OBJS_DIR)/%.o: $(UTSBASE)/common/avs/ns/dsw/%.c
444 $(COMPILE.c) -o $@ $<
445 $(CTFCONVERT_O)

447 $(OBJS_DIR)/%.o: $(UTSBASE)/common/avs/ns/nsctl/%.c
448 $(COMPILE.c) -o $@ $<
449 $(CTFCONVERT_O)

451 $(OBJS_DIR)/%.o: $(UTSBASE)/common/avs/ns/rdc/%.c
452 $(COMPILE.c) -o $@ $<
453 $(CTFCONVERT_O)

```

```

455 $(OBJS_DIR)/%.o: $(UTSBASE)/common/avs/ns/sdbc/%.c
456 $(COMPILE.c) -o $@ $<
457 $(CTFCONVERT_O)

459 $(OBJS_DIR)/%.o: $(UTSBASE)/common/avs/ns/solaris/%.c
460 $(COMPILE.c) -o $@ $<
461 $(CTFCONVERT_O)

463 $(OBJS_DIR)/%.o: $(UTSBASE)/common/avs/ns/sv/%.c
464 $(COMPILE.c) -o $@ $<
465 $(CTFCONVERT_O)

467 $(OBJS_DIR)/%.o: $(UTSBASE)/common/avs/ns/unistat/%.c
468 $(COMPILE.c) -o $@ $<
469 $(CTFCONVERT_O)

471 $(OBJS_DIR)/%.o: $(UTSBASE)/common/idmap/%.c
472 $(COMPILE.c) -o $@ $<
473 $(CTFCONVERT_O)

475 $(OBJS_DIR)/%.o: $(UTSBASE)/common/inet/%.c
476 $(COMPILE.c) -o $@ $<
477 $(CTFCONVERT_O)

479 $(OBJS_DIR)/%.o: $(UTSBASE)/common/inet/arp/%.c
480 $(COMPILE.c) -o $@ $<
481 $(CTFCONVERT_O)

483 $(OBJS_DIR)/%.o: $(UTSBASE)/common/inet/ip/%.c
484 $(COMPILE.c) -o $@ $<
485 $(CTFCONVERT_O)

487 $(OBJS_DIR)/%.o: $(UTSBASE)/common/inet/ipnet/%.c
488 $(COMPILE.c) -o $@ $<
489 $(CTFCONVERT_O)

491 $(OBJS_DIR)/%.o: $(UTSBASE)/common/inet/iptun/%.c
492 $(COMPILE.c) -o $@ $<
493 $(CTFCONVERT_O)

495 $(OBJS_DIR)/%.o: $(UTSBASE)/common/inet/kssl/%.c
496 $(COMPILE.c) -o $@ $<
497 $(CTFCONVERT_O)

499 $(OBJS_DIR)/%.o: $(UTSBASE)/common/inet/sctp/%.c
500 $(COMPILE.c) -o $@ $<
501 $(CTFCONVERT_O)

503 $(OBJS_DIR)/%.o: $(UTSBASE)/common/inet/tcp/%.c
504 $(COMPILE.c) -o $@ $<
505 $(CTFCONVERT_O)

507 $(OBJS_DIR)/%.o: $(UTSBASE)/common/inet/ilb/%.c
508 $(COMPILE.c) -o $@ $<
509 $(CTFCONVERT_O)

511 $(OBJS_DIR)/%.o: $(UTSBASE)/common/inet/ipf/%.c
512 $(COMPILE.c) -o $@ $<
513 $(CTFCONVERT_O)

515 $(OBJS_DIR)/%.o: $(UTSBASE)/common/inet/ipd/%.c
516 $(COMPILE.c) -o $@ $<
517 $(CTFCONVERT_O)

519 $(OBJS_DIR)/%.o: $(COMMONBASE)/net/patricia/%.c
520 $(COMPILE.c) -o $@ $<

```

```

521     $(CTFCONVERT_O)

523 $(OBJSDIR)/%.o:                $(UTSBASE)/common/inet/udp/%.c
524     $(COMPILE.c) -o $@ $<
525     $(CTFCONVERT_O)

527 $(OBJSDIR)/%.o:                $(UTSBASE)/common/inet/nca/%.c
528     $(COMPILE.c) -o $@ $<
529     $(CTFCONVERT_O)

531 $(OBJSDIR)/%.o:                $(UTSBASE)/common/inet/sockmods/%.c
532     $(COMPILE.c) -o $@ $<
533     $(CTFCONVERT_O)

535 $(OBJSDIR)/%.o:                $(UTSBASE)/common/inet/dlpistub/%.c
536     $(COMPILE.c) -o $@ $<
537     $(CTFCONVERT_O)

539 $(OBJSDIR)/%.o:                $(UTSBASE)/common/io/%.c
540     $(COMPILE.c) -o $@ $<
541     $(CTFCONVERT_O)

543 $(OBJSDIR)/%.o:                $(UTSBASE)/common/io/1394/%.c
544     $(COMPILE.c) -o $@ $<
545     $(CTFCONVERT_O)

547 $(OBJSDIR)/%.o:                $(UTSBASE)/common/io/1394/adapters/%.c
548     $(COMPILE.c) -o $@ $<
549     $(CTFCONVERT_O)

551 $(OBJSDIR)/%.o:                $(UTSBASE)/common/io/1394/targets/av1394/%.c
552     $(COMPILE.c) -o $@ $<
553     $(CTFCONVERT_O)

555 $(OBJSDIR)/%.o:                $(UTSBASE)/common/io/1394/targets/dcam1394/%.c
556     $(COMPILE.c) -o $@ $<
557     $(CTFCONVERT_O)

559 $(OBJSDIR)/%.o:                $(UTSBASE)/common/io/1394/targets/scsal394/%.c
560     $(COMPILE.c) -o $@ $<
561     $(CTFCONVERT_O)

563 $(OBJSDIR)/%.o:                $(UTSBASE)/common/io/sbp2/%.c
564     $(COMPILE.c) -o $@ $<
565     $(CTFCONVERT_O)

567 $(OBJSDIR)/%.o:                $(UTSBASE)/common/io/aac/%.c
568     $(COMPILE.c) -o $@ $<
569     $(CTFCONVERT_O)

571 $(OBJSDIR)/%.o:                $(UTSBASE)/common/io/afe/%.c
572     $(COMPILE.c) -o $@ $<
573     $(CTFCONVERT_O)

575 $(OBJSDIR)/%.o:                $(UTSBASE)/common/io/atge/%.c
576     $(COMPILE.c) -o $@ $<
577     $(CTFCONVERT_O)

579 $(OBJSDIR)/%.o:                $(UTSBASE)/common/io/arn/%.c
580     $(COMPILE.c) -o $@ $<
581     $(CTFCONVERT_O)

583 $(OBJSDIR)/%.o:                $(UTSBASE)/common/io/ath/%.c
584     $(COMPILE.c) -o $@ $<
585     $(CTFCONVERT_O)

```

```

587 $(OBJSDIR)/%.o:                $(UTSBASE)/common/io/atu/%.c
588     $(COMPILE.c) -o $@ $<
589     $(CTFCONVERT_O)

591 $(OBJSDIR)/%.o:                $(UTSBASE)/common/io/audio/impl/%.c
592     $(COMPILE.c) -o $@ $<
593     $(CTFCONVERT_O)

595 $(OBJSDIR)/%.o:                $(UTSBASE)/common/io/audio/ac97/%.c
596     $(COMPILE.c) -o $@ $<
597     $(CTFCONVERT_O)

599 $(OBJSDIR)/%.o:                $(UTSBASE)/common/io/audio/drv/audioens/%.c
600     $(COMPILE.c) -o $@ $<
601     $(CTFCONVERT_O)

603 $(OBJSDIR)/%.o:                $(UTSBASE)/common/io/audio/drv/audioemul0k/%.c
604     $(COMPILE.c) -o $@ $<
605     $(CTFCONVERT_O)

607 $(OBJSDIR)/%.o:                $(UTSBASE)/common/io/audio/drv/audio1575/%.c
608     $(COMPILE.c) -o $@ $<
609     $(CTFCONVERT_O)

611 $(OBJSDIR)/%.o:                $(UTSBASE)/common/io/audio/drv/audio810/%.c
612     $(COMPILE.c) -o $@ $<
613     $(CTFCONVERT_O)

615 $(OBJSDIR)/%.o:                $(UTSBASE)/common/io/audio/drv/audiocmi/%.c
616     $(COMPILE.c) -o $@ $<
617     $(CTFCONVERT_O)

619 $(OBJSDIR)/%.o:                $(UTSBASE)/common/io/audio/drv/audiocmihd/%.c
620     $(COMPILE.c) -o $@ $<
621     $(CTFCONVERT_O)

623 $(OBJSDIR)/%.o:                $(UTSBASE)/common/io/audio/drv/audiohd/%.c
624     $(COMPILE.c) -o $@ $<
625     $(CTFCONVERT_O)

627 $(OBJSDIR)/%.o:                $(UTSBASE)/common/io/audio/drv/audioixp/%.c
628     $(COMPILE.c) -o $@ $<
629     $(CTFCONVERT_O)

631 $(OBJSDIR)/%.o:                $(UTSBASE)/common/io/audio/drv/audiols/%.c
632     $(COMPILE.c) -o $@ $<
633     $(CTFCONVERT_O)

635 $(OBJSDIR)/%.o:                $(UTSBASE)/common/io/audio/drv/audiopci/%.c
636     $(COMPILE.c) -o $@ $<
637     $(CTFCONVERT_O)

639 $(OBJSDIR)/%.o:                $(UTSBASE)/common/io/audio/drv/audiopl6x/%.c
640     $(COMPILE.c) -o $@ $<
641     $(CTFCONVERT_O)

643 $(OBJSDIR)/%.o:                $(UTSBASE)/common/io/audio/drv/audiosolo/%.c
644     $(COMPILE.c) -o $@ $<
645     $(CTFCONVERT_O)

647 $(OBJSDIR)/%.o:                $(UTSBASE)/common/io/audio/drv/audiotots/%.c
648     $(COMPILE.c) -o $@ $<
649     $(CTFCONVERT_O)

651 $(OBJSDIR)/%.o:                $(UTSBASE)/common/io/audio/drv/audiovia823x/%.c
652     $(COMPILE.c) -o $@ $<

```

```

653      $(CTFCONVERT_O)

655 $(OBJS_DIR)/%.o:      $(UTSBASE)/common/io/audio/drv/audiovia97/%.c
656     $(COMPILE.c) -o $@ $<
657     $(CTFCONVERT_O)

659 $(OBJS_DIR)/%.o:      $(UTSBASE)/common/io/bfe/%.c
660     $(COMPILE.c) -o $@ $<
661     $(CTFCONVERT_O)

663 $(OBJS_DIR)/%.o:      $(UTSBASE)/common/io/bge/%.c
664     $(COMPILE.c) -o $@ $<
665     $(CTFCONVERT_O)

667 $(OBJS_DIR)/%.o:      $(UTSBASE)/common/io/blkdev/%.c
668     $(COMPILE.c) -o $@ $<
669     $(CTFCONVERT_O)

671 $(OBJS_DIR)/%.o:      $(UTSBASE)/common/io/bnxex/%.c
672     $(COMPILE.c) -o $@ $<
673     $(CTFCONVERT_O)

675 $(OBJS_DIR)/%.o:      $(UTSBASE)/common/io/bnxex/577xx/common/%.c
676     $(COMPILE.c) -o $@ $<
677     $(CTFCONVERT_O)

679 $(OBJS_DIR)/%.o:      $(UTSBASE)/common/io/bnxex/577xx/drivers/common/e
680     $(COMPILE.c) -o $@ $<
681     $(CTFCONVERT_O)

683 $(OBJS_DIR)/%.o:      $(UTSBASE)/common/io/bnxex/577xx/drivers/common/l
684     $(COMPILE.c) -o $@ $<
685     $(CTFCONVERT_O)

687 $(OBJS_DIR)/%.o:      $(UTSBASE)/common/io/bnxex/577xx/drivers/common/l
688     $(COMPILE.c) -o $@ $<
689     $(CTFCONVERT_O)

691 $(OBJS_DIR)/%.o:      $(UTSBASE)/common/io/bnxex/577xx/drivers/common/l
692     $(COMPILE.c) -o $@ $<
693     $(CTFCONVERT_O)

695 $(OBJS_DIR)/%.o:      $(UTSBASE)/common/io/bnxex/577xx/drivers/common/l
696     $(COMPILE.c) -o $@ $<
697     $(CTFCONVERT_O)

699 $(OBJS_DIR)/%.o:      $(UTSBASE)/common/io/bnxex/577xx/drivers/common/l
700     $(COMPILE.c) -o $@ $<
701     $(CTFCONVERT_O)

703 $(OBJS_DIR)/%.o:      $(UTSBASE)/common/io/bpf/%.c
704     $(COMPILE.c) -o $@ $<
705     $(CTFCONVERT_O)

707 $(OBJS_DIR)/%.o:      $(UTSBASE)/common/io/cardbus/%.c
708     $(COMPILE.c) -o $@ $<
709     $(CTFCONVERT_O)

711 $(OBJS_DIR)/%.o:      $(UTSBASE)/common/io/comstar/stmf/%.c
712     $(COMPILE.c) -o $@ $<
713     $(CTFCONVERT_O)

715 $(OBJS_DIR)/%.o:      $(UTSBASE)/common/io/comstar/port/fct/%.c
716     $(COMPILE.c) -o $@ $<
717     $(CTFCONVERT_O)

```

```

719 $(OBJS_DIR)/%.o:      $(UTSBASE)/common/io/comstar/port/qlt/%.c
720     $(COMPILE.c) -o $@ $<
721     $(CTFCONVERT_O)

723 $(OBJS_DIR)/%.o:      $(UTSBASE)/common/io/comstar/port/srpt/%.c
724     $(COMPILE.c) -o $@ $<
725     $(CTFCONVERT_O)

727 $(OBJS_DIR)/%.o:      $(UTSBASE)/common/io/comstar/port/fcoet/%.c
728     $(COMPILE.c) -o $@ $<
729     $(CTFCONVERT_O)

731 $(OBJS_DIR)/%.o:      $(COMMONBASE)/iscsit/%.c
732     $(COMPILE.c) -o $@ $<
733     $(CTFCONVERT_O)

735 $(OBJS_DIR)/%.o:      $(UTSBASE)/common/io/comstar/port/iscsit/%.c
736     $(COMPILE.c) -o $@ $<
737     $(CTFCONVERT_O)

739 $(OBJS_DIR)/%.o:      $(UTSBASE)/common/io/comstar/port/pppt/%.c
740     $(COMPILE.c) -o $@ $<
741     $(CTFCONVERT_O)

743 $(OBJS_DIR)/%.o:      $(UTSBASE)/common/io/comstar/lu/stmf_sbd/%.c
744     $(COMPILE.c) -o $@ $<
745     $(CTFCONVERT_O)

747 $(OBJS_DIR)/%.o:      $(UTSBASE)/common/io/cpqary3/%.c
748     $(COMPILE.c) -o $@ $<
749     $(CTFCONVERT_O)

751 $(OBJS_DIR)/%.o:      $(UTSBASE)/common/io/dld/%.c
752     $(COMPILE.c) -o $@ $<
753     $(CTFCONVERT_O)

755 $(OBJS_DIR)/%.o:      $(UTSBASE)/common/io/dls/%.c
756     $(COMPILE.c) -o $@ $<
757     $(CTFCONVERT_O)

759 $(OBJS_DIR)/%.o:      $(UTSBASE)/common/io/dmfe/%.c
760     $(COMPILE.c) -o $@ $<
761     $(CTFCONVERT_O)

763 $(OBJS_DIR)/%.o:      $(UTSBASE)/common/io/drm/%.c
764     $(COMPILE.c) -o $@ $<
765     $(CTFCONVERT_O)

767 $(OBJS_DIR)/%.o:      $(UTSBASE)/common/io/efe/%.c
768     $(COMPILE.c) -o $@ $<
769     $(CTFCONVERT_O)

771 $(OBJS_DIR)/%.o:      $(UTSBASE)/common/io/elxl/%.c
772     $(COMPILE.c) -o $@ $<
773     $(CTFCONVERT_O)

775 $(OBJS_DIR)/%.o:      $(UTSBASE)/common/io/fcoe/%.c
776     $(COMPILE.c) -o $@ $<
777     $(CTFCONVERT_O)

779 $(OBJS_DIR)/%.o:      $(UTSBASE)/common/io/hme/%.c
780     $(COMPILE.c) -o $@ $<
781     $(CTFCONVERT_O)

783 $(OBJS_DIR)/%.o:      $(UTSBASE)/common/io/pciex/%.c
784     $(COMPILE.c) -o $@ $<

```

```

785     $(CTFCONVERT_O)

787 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/hotplug/hpcsvc/%.c
788     $(COMPILE.c) -o $@ $<
789     $(CTFCONVERT_O)

791 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/pciex/hotplug/%.c
792     $(COMPILE.c) -o $@ $<
793     $(CTFCONVERT_O)

795 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/hotplug/pcihp/%.c
796     $(COMPILE.c) -o $@ $<
797     $(CTFCONVERT_O)

799 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/ib/clients/rds/%.c
800     $(COMPILE.c) -o $@ $<
801     $(CTFCONVERT_O)

803 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/ib/clients/rdsv3/%.c
804     $(COMPILE.c) -o $@ $<
805     $(CTFCONVERT_O)

807 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/ib/clients/iser/%.c
808     $(COMPILE.c) -o $@ $<
809     $(CTFCONVERT_O)

811 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/ib/clients/ibd/%.c
812     $(COMPILE.c) -o $@ $<
813     $(CTFCONVERT_O)

815 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/ib/clients/eoib/%.c
816     $(COMPILE.c) -o $@ $<
817     $(CTFCONVERT_O)

819 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/ib/clients/of/sol_ofs/%.c
820     $(COMPILE.c) -o $@ $<
821     $(CTFCONVERT_O)

823 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/ib/clients/of/sol_ucma/%.c
824     $(COMPILE.c) -o $@ $<
825     $(CTFCONVERT_O)

827 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/ib/clients/of/sol_umad/%.c
828     $(COMPILE.c) -o $@ $<
829     $(CTFCONVERT_O)

831 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/ib/clients/of/sol_uverbs/%.
832     $(COMPILE.c) -o $@ $<
833     $(CTFCONVERT_O)

835 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/ib/clients/sdp/%.c
836     $(COMPILE.c) -o $@ $<
837     $(CTFCONVERT_O)

839 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/ib/mgt/ibcm/%.c
840     $(COMPILE.c) -o $@ $<
841     $(CTFCONVERT_O)

843 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/ib/mgt/ibdm/%.c
844     $(COMPILE.c) -o $@ $<
845     $(CTFCONVERT_O)

847 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/ib/mgt/ibdma/%.c
848     $(COMPILE.c) -o $@ $<
849     $(CTFCONVERT_O)

```

```

851 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/ib/mgt/ibmf/%.c
852     $(COMPILE.c) -o $@ $<
853     $(CTFCONVERT_O)

855 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/ib/ibnex/%.c
856     $(COMPILE.c) -o $@ $<
857     $(CTFCONVERT_O)

859 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/ib/ibt1/%.c
860     $(COMPILE.c) -o $@ $<
861     $(CTFCONVERT_O)

863 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/ib/adapters/tavor/%.c
864     $(COMPILE.c) -o $@ $<
865     $(CTFCONVERT_O)

867 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/ib/adapters/hermon/%.c
868     $(COMPILE.c) -o $@ $<
869     $(CTFCONVERT_O)

871 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/ib/clients/daplt/%.c
872     $(COMPILE.c) -o $@ $<
873     $(CTFCONVERT_O)

875 $(OBJS_DIR)/%.o:                $(COMMONBASE)/iscsi/%.c
876     $(COMPILE.c) -o $@ $<
877     $(CTFCONVERT_O)

879 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/idm/%.c
880     $(COMPILE.c) -o $@ $<
881     $(CTFCONVERT_O)

883 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/ipw/%.c
884     $(COMPILE.c) -o $@ $<
885     $(CTFCONVERT_O)

887 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/iwh/%.c
888     $(COMPILE.c) -o $@ $<
889     $(CTFCONVERT_O)

891 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/iwi/%.c
892     $(COMPILE.c) -o $@ $<
893     $(CTFCONVERT_O)

895 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/iwk/%.c
896     $(COMPILE.c) -o $@ $<
897     $(CTFCONVERT_O)

899 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/iwp/%.c
900     $(COMPILE.c) -o $@ $<
901     $(CTFCONVERT_O)

903 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/kb8042/%.c
904     $(COMPILE.c) -o $@ $<
905     $(CTFCONVERT_O)

907 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/kbtrans/%.c
908     $(COMPILE.c) -o $@ $<
909     $(CTFCONVERT_O)

911 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/ksocket/%.c
912     $(COMPILE.c) -o $@ $<
913     $(CTFCONVERT_O)

915 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/aggr/%.c
916     $(COMPILE.c) -o $@ $<

```

```

917      $(CTFCONVERT_O)

919 $(OBJS_DIR)/%.o:      $(UTSBASE)/common/io/lp/%.c
920     $(COMPILE.c) -o $@ $<
921     $(CTFCONVERT_O)

923 $(OBJS_DIR)/%.o:      $(UTSBASE)/common/io/lvm/hotspares/%.c
924     $(COMPILE.c) -o $@ $<
925     $(CTFCONVERT_O)

927 $(OBJS_DIR)/%.o:      $(UTSBASE)/common/io/lvm/md/%.c
928     $(COMPILE.c) -o $@ $<
929     $(CTFCONVERT_O)

931 $(OBJS_DIR)/%.o:      $(UTSBASE)/common/io/lvm/mirror/%.c
932     $(COMPILE.c) -o $@ $<
933     $(CTFCONVERT_O)

935 $(OBJS_DIR)/%.o:      $(UTSBASE)/common/io/lvm/notify/%.c
936     $(COMPILE.c) -o $@ $<
937     $(CTFCONVERT_O)

939 $(OBJS_DIR)/%.o:      $(UTSBASE)/common/io/lvm/raid/%.c
940     $(COMPILE.c) -o $@ $<
941     $(CTFCONVERT_O)

943 $(OBJS_DIR)/%.o:      $(UTSBASE)/common/io/lvm/softpart/%.c
944     $(COMPILE.c) -o $@ $<
945     $(CTFCONVERT_O)

947 $(OBJS_DIR)/%.o:      $(UTSBASE)/common/io/lvm/stripe/%.c
948     $(COMPILE.c) -o $@ $<
949     $(CTFCONVERT_O)

951 $(OBJS_DIR)/%.o:      $(UTSBASE)/common/io/lvm/trans/%.c
952     $(COMPILE.c) -o $@ $<
953     $(CTFCONVERT_O)

955 $(OBJS_DIR)/%.o:      $(UTSBASE)/common/io/mac/%.c
956     $(COMPILE.c) -o $@ $<
957     $(CTFCONVERT_O)

959 $(OBJS_DIR)/%.o:      $(UTSBASE)/common/io/mac/plugins/%.c
960     $(COMPILE.c) -o $@ $<
961     $(CTFCONVERT_O)

963 $(OBJS_DIR)/%.o:      $(UTSBASE)/common/io/mega_sas/%.c
964     $(COMPILE.c) -o $@ $<
965     $(CTFCONVERT_O)

967 $(OBJS_DIR)/%.o:      $(UTSBASE)/common/io/mii/%.c
968     $(COMPILE.c) -o $@ $<
969     $(CTFCONVERT_O)

971 $(OBJS_DIR)/%.o:      $(UTSBASE)/common/io/mr_sas/%.c
972     $(COMPILE.c) -o $@ $<
973     $(CTFCONVERT_O)

975 $(OBJS_DIR)/%.o:      $(UTSBASE)/common/io/scsi/adapters/mpt_sas/%.c
976     $(COMPILE.c) -o $@ $<
977     $(CTFCONVERT_O)

979 $(OBJS_DIR)/%.o:      $(UTSBASE)/common/io/mxfe/%.c
980     $(COMPILE.c) -o $@ $<
981     $(CTFCONVERT_O)

```

```

983 $(OBJS_DIR)/%.o:      $(UTSBASE)/common/io/mwl/%.c
984     $(COMPILE.c) -o $@ $<
985     $(CTFCONVERT_O)

987 $(OBJS_DIR)/%.o:      $(UTSBASE)/common/io/mwl/mwl_fw/%.c
988     $(COMPILE.c) -o $@ $<
989     $(CTFCONVERT_O)

991 $(OBJS_DIR)/%.o:      $(UTSBASE)/common/io/net80211/%.c
992     $(COMPILE.c) -o $@ $<
993     $(CTFCONVERT_O)

995 $(OBJS_DIR)/%.o:      $(UTSBASE)/common/io/nge/%.c
996     $(COMPILE.c) -o $@ $<
997     $(CTFCONVERT_O)

999 $(OBJS_DIR)/%.o:      $(UTSBASE)/common/io/nvme/%.c
1000    $(COMPILE.c) -o $@ $<
1001    $(CTFCONVERT_O)

1003 $(OBJS_DIR)/%.o:      $(UTSBASE)/common/io/nxge/%.c
1004    $(COMPILE.c) -o $@ $<
1005    $(CTFCONVERT_O)

1007 $(OBJS_DIR)/%.o:      $(UTSBASE)/common/io/nxge/npi/%.c
1008    $(COMPILE.c) -o $@ $<
1009    $(CTFCONVERT_O)

1011 $(OBJS_DIR)/%.o:      $(UTSBASE)/common/io/nxge/%.s
1012    $(COMPILE.s) -o $@ $<

1014 $(OBJS_DIR)/%.o:      $(UTSBASE)/common/io/pci-ide/%.c
1015    $(COMPILE.c) -o $@ $<
1016    $(CTFCONVERT_O)

1018 $(OBJS_DIR)/%.o:      $(UTSBASE)/common/io/pcn/%.c
1019    $(COMPILE.c) -o $@ $<
1020    $(CTFCONVERT_O)

1022 $(OBJS_DIR)/%.o:      $(UTSBASE)/common/io/ppp/sppp/%.c
1023    $(COMPILE.c) -o $@ $<
1024    $(CTFCONVERT_O)

1026 $(OBJS_DIR)/%.o:      $(UTSBASE)/common/io/ppp/spppasyn/%.c
1027    $(COMPILE.c) -o $@ $<
1028    $(CTFCONVERT_O)

1030 $(OBJS_DIR)/%.o:      $(UTSBASE)/common/io/ppp/sppptun/%.c
1031    $(COMPILE.c) -o $@ $<
1032    $(CTFCONVERT_O)

1034 $(OBJS_DIR)/%.o:      $(UTSBASE)/common/io/ral/%.c
1035    $(COMPILE.c) -o $@ $<
1036    $(CTFCONVERT_O)

1038 $(OBJS_DIR)/%.o:      $(UTSBASE)/common/io/rge/%.c
1039    $(COMPILE.c) -o $@ $<
1040    $(CTFCONVERT_O)

1042 $(OBJS_DIR)/%.o:      $(UTSBASE)/common/io/rtls/%.c
1043    $(COMPILE.c) -o $@ $<
1044    $(CTFCONVERT_O)

1046 $(OBJS_DIR)/%.o:      $(UTSBASE)/common/io/rsm/%.c
1047    $(COMPILE.c) -o $@ $<
1048    $(CTFCONVERT_O)

```

```

1050 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/rtw/%.c
1051     $(COMPILE.c) -o $@ $<
1052     $(CTFCONVERT_O)

1054 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/rum/%.c
1055     $(COMPILE.c) -o $@ $<
1056     $(CTFCONVERT_O)

1058 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/rwd/%.c
1059     $(COMPILE.c) -o $@ $<
1060     $(CTFCONVERT_O)

1062 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/rwn/%.c
1063     $(COMPILE.c) -o $@ $<
1064     $(CTFCONVERT_O)

1066 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/sata/adapters/ahci/%.c
1067     $(COMPILE.c) -o $@ $<
1068     $(CTFCONVERT_O)

1070 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/sata/adapters/nv_sata/%.c
1071     $(COMPILE.c) -o $@ $<
1072     $(CTFCONVERT_O)

1074 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/sata/adapters/si3124/%.c
1075     $(COMPILE.c) -o $@ $<
1076     $(CTFCONVERT_O)

1078 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/sata/impl/%.c
1079     $(COMPILE.c) -o $@ $<
1080     $(CTFCONVERT_O)

1082 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/scsi/conf/%.c
1083     $(COMPILE.c) -o $@ $<
1084     $(CTFCONVERT_O)

1086 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/scsi/impl/%.c
1087     $(COMPILE.c) -o $@ $<
1088     $(CTFCONVERT_O)

1090 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/scsi/targets/%.c
1091     $(COMPILE.c) -o $@ $<
1092     $(CTFCONVERT_O)

1094 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/scsi/adapters/%.c
1095     $(COMPILE.c) -o $@ $<
1096     $(CTFCONVERT_O)

1098 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/scsi/adapters/blk2scsa/%.c
1099     $(COMPILE.c) -o $@ $<
1100     $(CTFCONVERT_O)

1102 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/scsi/adapters/scsi_vhci/%.c
1103     $(COMPILE.c) -o $@ $<
1104     $(CTFCONVERT_O)

1106 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/scsi/adapters/scsi_vhci/fop
1107     $(COMPILE.c) -o $@ $<
1108     $(CTFCONVERT_O)

1110 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/fibre-channel/ulp/%.c
1111     $(COMPILE.c) -o $@ $<
1112     $(CTFCONVERT_O)

1114 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/fibre-channel/impl/%.c

```

```

1115     $(COMPILE.c) -o $@ $<
1116     $(CTFCONVERT_O)

1118 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/fibre-channel/fca/qlc/%.c
1119     $(COMPILE.c) -o $@ $<
1120     $(CTFCONVERT_O)

1122 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/fibre-channel/fca/qlge/%.c
1123     $(COMPILE.c) -o $@ $<
1124     $(CTFCONVERT_O)

1126 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/fibre-channel/fca/emlxs/%.c
1127     $(COMPILE.c) -o $@ $<
1128     $(CTFCONVERT_O)

1130 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/fibre-channel/fca/oce/%.c
1131     $(COMPILE.c) -o $@ $<
1132     $(CTFCONVERT_O)

1134 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/fibre-channel/fca/fcoei/%.c
1135     $(COMPILE.c) -o $@ $<
1136     $(CTFCONVERT_O)

1138 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/sdcard/adapters/sdhost/%.c
1139     $(COMPILE.c) -o $@ $<
1140     $(CTFCONVERT_O)

1142 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/sdcard/impl/%.c
1143     $(COMPILE.c) -o $@ $<
1144     $(CTFCONVERT_O)

1146 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/sdcard/targets/sdcard/%.c
1147     $(COMPILE.c) -o $@ $<
1148     $(CTFCONVERT_O)

1150 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/sfe/%.c
1151     $(COMPILE.c) -o $@ $<
1152     $(CTFCONVERT_O)

1154 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/simnet/%.c
1155     $(COMPILE.c) -o $@ $<
1156     $(CTFCONVERT_O)

1158 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/softmac/%.c
1159     $(COMPILE.c) -o $@ $<
1160     $(CTFCONVERT_O)

1162 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/uath/%.c
1163     $(COMPILE.c) -o $@ $<
1164     $(CTFCONVERT_O)

1166 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/uath/uath_fw/%.c
1167     $(COMPILE.c) -o $@ $<
1168     $(CTFCONVERT_O)

1170 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/ural/%.c
1171     $(COMPILE.c) -o $@ $<
1172     $(CTFCONVERT_O)

1174 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/urtw/%.c
1175     $(COMPILE.c) -o $@ $<
1176     $(CTFCONVERT_O)

1178 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/clients/audio/usb_ac/%.
1179     $(COMPILE.c) -o $@ $<
1180     $(CTFCONVERT_O)

```

```

1182 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/clients/audio/usb_as/%.
1183 $(COMPILE.c) -o $@ $<
1184 $(CTFCONVERT_O)

1186 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/clients/audio/usb_ah/%.
1187 $(COMPILE.c) -o $@ $<
1188 $(CTFCONVERT_O)

1190 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/clients/usbskel/%.c
1191 $(COMPILE.c) -o $@ $<
1192 $(CTFCONVERT_O)

1194 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/clients/video/usbvc/%.c
1195 $(COMPILE.c) -o $@ $<
1196 $(CTFCONVERT_O)

1198 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/clients/hid/%.c
1199 $(COMPILE.c) -o $@ $<
1200 $(CTFCONVERT_O)

1202 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/clients/hidparser/%.c
1203 $(COMPILE.c) -o $@ $<
1204 $(CTFCONVERT_O)

1206 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/clients/printer/%.c
1207 $(COMPILE.c) -o $@ $<
1208 $(CTFCONVERT_O)

1210 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/clients/usbkbm/%.c
1211 $(COMPILE.c) -o $@ $<
1212 $(CTFCONVERT_O)

1214 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/clients/usbms/%.c
1215 $(COMPILE.c) -o $@ $<
1216 $(CTFCONVERT_O)

1218 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/clients/usbinput/usbwcm
1219 $(COMPILE.c) -o $@ $<
1220 $(CTFCONVERT_O)

1222 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/clients/ugen/%.c
1223 $(COMPILE.c) -o $@ $<
1224 $(CTFCONVERT_O)

1226 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/clients/usbser/%.c
1227 $(COMPILE.c) -o $@ $<
1228 $(CTFCONVERT_O)

1230 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/clients/usbser/usbsacm/
1231 $(COMPILE.c) -o $@ $<
1232 $(CTFCONVERT_O)

1234 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/clients/usbser/usbftdi/
1235 $(COMPILE.c) -o $@ $<
1236 $(CTFCONVERT_O)

1238 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/clients/usbser/usbser_k
1239 $(COMPILE.c) -o $@ $<
1240 $(CTFCONVERT_O)

1242 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/clients/usbser/usbsprl/
1243 $(COMPILE.c) -o $@ $<
1244 $(CTFCONVERT_O)

1246 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/clients/usbecm/%.c

```

```

1247 $(COMPILE.c) -o $@ $<
1248 $(CTFCONVERT_O)

1250 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/hcd/openhci/%.c
1251 $(COMPILE.c) -o $@ $<
1252 $(CTFCONVERT_O)

1254 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/hcd/ehci/%.c
1255 $(COMPILE.c) -o $@ $<
1256 $(CTFCONVERT_O)

1258 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/hcd/uhci/%.c
1259 $(COMPILE.c) -I../common -o $@ $<
1260 $(CTFCONVERT_O)

1262 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/hubd/%.c
1263 $(COMPILE.c) -o $@ $<
1264 $(CTFCONVERT_O)

1266 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/scsa2usb/%.c
1267 $(COMPILE.c) -o $@ $<
1268 $(CTFCONVERT_O)

1270 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/usb_mid/%.c
1271 $(COMPILE.c) -o $@ $<
1272 $(CTFCONVERT_O)

1274 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/usb_ia/%.c
1275 $(COMPILE.c) -o $@ $<
1276 $(CTFCONVERT_O)

1278 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/usba/%.c
1279 $(COMPILE.c) -o $@ $<
1280 $(CTFCONVERT_O)

1282 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/usb/usba10/%.c
1283 $(COMPILE.c) -o $@ $<
1284 $(CTFCONVERT_O)

1286 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/vuidmice/%.c
1287 $(COMPILE.c) -o $@ $<
1288 $(CTFCONVERT_O)

1290 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/vnic/%.c
1291 $(COMPILE.c) -o $@ $<
1292 $(CTFCONVERT_O)

1294 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/wpi/%.c
1295 $(COMPILE.c) -o $@ $<
1296 $(CTFCONVERT_O)

1298 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/zyd/%.c
1299 $(COMPILE.c) -o $@ $<
1300 $(CTFCONVERT_O)

1302 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/chxge/com/%.c
1303 $(COMPILE.c) -o $@ $<
1304 $(CTFCONVERT_O)

1306 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/chxge/%.c
1307 $(COMPILE.c) -o $@ $<
1308 $(CTFCONVERT_O)

1310 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/cxgbe/common/%.c
1311 $(COMPILE.c) -o $@ $<
1312 $(CTFCONVERT_O)

```

```

1314 $(OBJS_DIR)/%.o: $(UTSBASE)/common/io/cxgbe/shared/%.c
1315 $(COMPILE.c) -o $@ $<
1316 $(CTFCONVERT_O)

1318 $(OBJS_DIR)/%.o: $(UTSBASE)/common/io/cxgbe/firmware/%.c
1319 $(COMPILE.c) -o $@ $<
1320 $(CTFCONVERT_O)

1322 $(OBJS_DIR)/%.o: $(UTSBASE)/common/io/cxgbe/t4nex/%.c
1323 $(COMPILE.c) -o $@ $<
1324 $(CTFCONVERT_O)

1326 $(OBJS_DIR)/%.o: $(UTSBASE)/common/io/cxgbe/cxgbe/%.c
1327 $(COMPILE.c) -o $@ $<
1328 $(CTFCONVERT_O)

1330 $(OBJS_DIR)/%.o: $(UTSBASE)/common/io/ixgb/%.c
1331 $(COMPILE.c) -o $@ $<
1332 $(CTFCONVERT_O)

1334 $(OBJS_DIR)/%.o: $(UTSBASE)/common/io/xge/drv/%.c
1335 $(COMPILE.c) -o $@ $<
1336 $(CTFCONVERT_O)

1338 $(OBJS_DIR)/%.o: $(UTSBASE)/common/io/xge/hal/xgehal/%.c
1339 $(COMPILE.c) -o $@ $<
1340 $(CTFCONVERT_O)

1342 $(OBJS_DIR)/%.o: $(UTSBASE)/common/io/e1000api/%.c
1343 $(COMPILE.c) -o $@ $<
1344 $(CTFCONVERT_O)

1346 $(OBJS_DIR)/%.o: $(UTSBASE)/common/io/e1000g/%.c
1347 $(COMPILE.c) -o $@ $<
1348 $(CTFCONVERT_O)

1350 $(OBJS_DIR)/%.o: $(UTSBASE)/common/io/igb/%.c
1351 $(COMPILE.c) -o $@ $<
1352 $(CTFCONVERT_O)

1354 $(OBJS_DIR)/%.o: $(UTSBASE)/common/io/iprb/%.c
1355 $(COMPILE.c) -o $@ $<
1356 $(CTFCONVERT_O)

1358 $(OBJS_DIR)/%.o: $(UTSBASE)/common/io/ixgbe/%.c
1359 $(COMPILE.c) -o $@ $<
1360 $(CTFCONVERT_O)

1362 $(OBJS_DIR)/%.o: $(UTSBASE)/common/io/ixgbe/core/%.c
1363 $(COMPILE.c) -o $@ $<
1364 $(CTFCONVERT_O)

1366 $(OBJS_DIR)/%.o: $(UTSBASE)/common/io/ntxn/%.c
1367 $(COMPILE.c) -o $@ $<
1368 $(CTFCONVERT_O)

1370 $(OBJS_DIR)/%.o: $(UTSBASE)/common/io/myri10ge/drv/%.c
1371 $(COMPILE.c) -o $@ $<
1372 $(CTFCONVERT_O)

1374 $(OBJS_DIR)/%.o: $(UTSBASE)/common/ipp/%.c
1375 $(COMPILE.c) -o $@ $<
1376 $(CTFCONVERT_O)

1378 $(OBJS_DIR)/%.o: $(UTSBASE)/common/ipp/ipgpc/%.c

```

```

1379 $(COMPILE.c) -o $@ $<
1380 $(CTFCONVERT_O)

1382 $(OBJS_DIR)/%.o: $(UTSBASE)/common/ipp/dlcosmk/%.c
1383 $(COMPILE.c) -o $@ $<
1384 $(CTFCONVERT_O)

1386 $(OBJS_DIR)/%.o: $(UTSBASE)/common/ipp/flowacct/%.c
1387 $(COMPILE.c) -o $@ $<
1388 $(CTFCONVERT_O)

1390 $(OBJS_DIR)/%.o: $(UTSBASE)/common/ipp/dscpmk/%.c
1391 $(COMPILE.c) -o $@ $<
1392 $(CTFCONVERT_O)

1394 $(OBJS_DIR)/%.o: $(UTSBASE)/common/ipp/meters/%.c
1395 $(COMPILE.c) -o $@ $<
1396 $(CTFCONVERT_O)

1398 $(OBJS_DIR)/%.o: $(UTSBASE)/common/kiconv/kiconv_emea/%.c
1399 $(COMPILE.c) -o $@ $<
1400 $(CTFCONVERT_O)

1402 $(OBJS_DIR)/%.o: $(UTSBASE)/common/kiconv/kiconv_ja/%.c
1403 $(COMPILE.c) -o $@ $<
1404 $(CTFCONVERT_O)

1406 $(OBJS_DIR)/%.o: $(UTSBASE)/common/kiconv/kiconv_ko/%.c
1407 $(COMPILE.c) -o $@ $<
1408 $(CTFCONVERT_O)

1410 $(OBJS_DIR)/%.o: $(UTSBASE)/common/kiconv/kiconv_sc/%.c
1411 $(COMPILE.c) -o $@ $<
1412 $(CTFCONVERT_O)

1414 $(OBJS_DIR)/%.o: $(UTSBASE)/common/kiconv/kiconv_tc/%.c
1415 $(COMPILE.c) -o $@ $<
1416 $(CTFCONVERT_O)

1418 $(OBJS_DIR)/%.o: $(UTSBASE)/common/klm/%.c
1419 $(COMPILE.c) -o $@ $<
1420 $(CTFCONVERT_O)

1422 $(OBJS_DIR)/%.o: $(UTSBASE)/common/kmdb/%.c
1423 $(COMPILE.c) -o $@ $<
1424 $(CTFCONVERT_O)

1426 $(OBJS_DIR)/%.o: $(UTSBASE)/common/ktli/%.c
1427 $(COMPILE.c) -o $@ $<
1428 $(CTFCONVERT_O)

1430 $(OBJS_DIR)/%.o: $(UTSBASE)/common/io/scsi/adapters/iscsi/%.c
1431 $(COMPILE.c) -o $@ $<
1432 $(CTFCONVERT_O)

1434 $(OBJS_DIR)/%.o: $(COMMONBASE)/iscsi/%.c
1435 $(COMPILE.c) -o $@ $<
1436 $(CTFCONVERT_O)

1438 $(OBJS_DIR)/%.o: $(UTSBASE)/common/inet/kifconf/%.c
1439 $(COMPILE.c) -o $@ $<
1440 $(CTFCONVERT_O)

1442 $(OBJS_DIR)/%.o: $(UTSBASE)/common/io/vr/%.c
1443 $(COMPILE.c) -o $@ $<
1444 $(CTFCONVERT_O)

```



```

1446 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/yge/%.c
1447 $(COMPILE.c) -o $@ $<
1448 $(CTFCONVERT_O)

1450 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/sfxge/%.c
1451 $(COMPILE.c) -o $@ $<
1452 $(CTFCONVERT_O)

1454 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/sfxge/common/%.c
1455 $(COMPILE.c) -o $@ $<
1456 $(CTFCONVERT_O)

1458 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/skd/%.c
1459 $(COMPILE.c) -o $@ $<
1460 $(CTFCONVERT_O)

1462 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/virtio/%.c
1463 $(COMPILE.c) -o $@ $<
1464 $(CTFCONVERT_O)

1466 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/vioblk/%.c
1467 $(COMPILE.c) -o $@ $<
1468 $(CTFCONVERT_O)

1470 $(OBJSDIR)/%.o: $(UTSBASE)/common/io/vioif/%.c
1471 $(COMPILE.c) -o $@ $<
1472 $(CTFCONVERT_O)
1473 #
1474 # krtld must refer to its own bzero/bcopy until the kernel is fully linked
1475 #
1476 $(OBJSDIR)/bootrd.o := CPPFLAGS += -DKOBJ_OVERRIDES
1477 $(OBJSDIR)/doreloc.o := CPPFLAGS += -DKOBJ_OVERRIDES
1478 $(OBJSDIR)/kobj.o := CPPFLAGS += -DKOBJ_OVERRIDES
1479 $(OBJSDIR)/kobj_boot.o := CPPFLAGS += -DKOBJ_OVERRIDES
1480 $(OBJSDIR)/kobj_bootflags.o := CPPFLAGS += -DKOBJ_OVERRIDES
1481 $(OBJSDIR)/kobj_convrelstr.o := CPPFLAGS += -DKOBJ_OVERRIDES
1482 $(OBJSDIR)/kobj_isa.o := CPPFLAGS += -DKOBJ_OVERRIDES
1483 $(OBJSDIR)/kobj_kdi.o := CPPFLAGS += -DKOBJ_OVERRIDES
1484 $(OBJSDIR)/kobj_lm.o := CPPFLAGS += -DKOBJ_OVERRIDES
1485 $(OBJSDIR)/kobj_reloc.o := CPPFLAGS += -DKOBJ_OVERRIDES
1486 $(OBJSDIR)/kobj_stubs.o := CPPFLAGS += -DKOBJ_OVERRIDES
1487 $(OBJSDIR)/kobj_subr.o := CPPFLAGS += -DKOBJ_OVERRIDES

1489 $(OBJSDIR)/%.o: $(UTSBASE)/common/krtld/%.c
1490 $(COMPILE.c) -o $@ $<
1491 $(CTFCONVERT_O)

1493 $(OBJSDIR)/%.o: $(COMMONBASE)/list/%.c
1494 $(COMPILE.c) -o $@ $<
1495 $(CTFCONVERT_O)

1497 $(OBJSDIR)/%.o: $(COMMONBASE)/lvm/%.c
1498 $(COMPILE.c) -o $@ $<
1499 $(CTFCONVERT_O)

1501 $(OBJSDIR)/%.o: $(COMMONBASE)/lzma/%.c
1502 $(COMPILE.c) -o $@ $<
1503 $(CTFCONVERT_O)

1505 $(OBJSDIR)/%.o: $(COMMONBASE)/crypto/md4/%.c
1506 $(COMPILE.c) -o $@ $<
1507 $(CTFCONVERT_O)

1509 $(OBJSDIR)/%.o: $(COMMONBASE)/crypto/md5/%.c
1510 $(COMPILE.c) -o $@ $<

```

```

1511 $(CTFCONVERT_O)

1513 $(OBJSDIR)/%.o: $(COMMONBASE)/net/dhcp/%.c
1514 $(COMPILE.c) -o $@ $<
1515 $(CTFCONVERT_O)

1517 $(OBJSDIR)/%.o: $(COMMONBASE)/nvpair/%.c
1518 $(COMPILE.c) -o $@ $<
1519 $(CTFCONVERT_O)

1521 $(OBJSDIR)/%.o: $(UTSBASE)/common/os/%.c
1522 $(COMPILE.c) -o $@ $<
1523 $(CTFCONVERT_O)

1525 $(OBJSDIR)/%.o: $(UTSBASE)/common/pcmcia/cis/%.c
1526 $(COMPILE.c) -o $@ $<
1527 $(CTFCONVERT_O)

1529 $(OBJSDIR)/%.o: $(UTSBASE)/common/pcmcia/cs/%.c
1530 $(COMPILE.c) -o $@ $<
1531 $(CTFCONVERT_O)

1533 $(OBJSDIR)/%.o: $(UTSBASE)/common/pcmcia/nexus/%.c
1534 $(COMPILE.c) -o $@ $<
1535 $(CTFCONVERT_O)

1537 $(OBJSDIR)/%.o: $(UTSBASE)/common/pcmcia/pcs/%.c
1538 $(COMPILE.c) -o $@ $<
1539 $(CTFCONVERT_O)

1541 $(OBJSDIR)/%.o: $(UTSBASE)/common/rpc/%.c
1542 $(COMPILE.c) -o $@ $<
1543 $(CTFCONVERT_O)

1545 $(OBJSDIR)/%.o: $(UTSBASE)/common/rpc/sec/%.c
1546 $(COMPILE.c) -o $@ $<
1547 $(CTFCONVERT_O)

1549 $(OBJSDIR)/%.o: $(UTSBASE)/common/rpc/sec_gss/%.c
1550 $(COMPILE.c) -o $@ $<
1551 $(CTFCONVERT_O)

1553 $(OBJSDIR)/%.o: $(COMMONBASE)/crypto/edonr/%.c
1554 $(COMPILE.c) -o $@ $<
1555 $(CTFCONVERT_O)

1557 $(OBJSDIR)/%.o: $(COMMONBASE)/crypto/shal/%.c
1558 $(COMPILE.c) -o $@ $<
1559 $(CTFCONVERT_O)

1561 $(OBJSDIR)/%.o: $(COMMONBASE)/crypto/sha2/%.c
1562 $(COMPILE.c) -o $@ $<
1563 $(CTFCONVERT_O)

1565 $(OBJSDIR)/%.o: $(COMMONBASE)/crypto/skein/%.c
1566 $(COMPILE.c) -o $@ $<
1567 $(CTFCONVERT_O)

1569 $(OBJSDIR)/%.o: $(UTSBASE)/common/syscall/%.c
1570 $(COMPILE.c) -o $@ $<
1571 $(CTFCONVERT_O)

1573 $(OBJSDIR)/%.o: $(UTSBASE)/common/tnf/%.c
1574 $(COMPILE.c) -o $@ $<
1575 $(CTFCONVERT_O)

```

```

1577 $(OBJS_DIR)/%.o: $(COMMONBASE)/tsol/%.c
1578 $(COMPILE.c) -o $@ $<
1579 $(CTFCONVERT_O)

1581 $(OBJS_DIR)/%.o: $(COMMONBASE)/util/%.c
1582 $(COMPILE.c) -o $@ $<
1583 $(CTFCONVERT_O)

1585 $(OBJS_DIR)/%.o: $(COMMONBASE)/unicode/%.c
1586 $(COMPILE.c) -o $@ $<
1587 $(CTFCONVERT_O)

1589 $(OBJS_DIR)/%.o: $(UTSBASE)/common/vm/%.c
1590 $(COMPILE.c) -o $@ $<
1591 $(CTFCONVERT_O)

1593 $(OBJS_DIR)/%.o: $(UTSBASE)/common/zmod/%.c
1594 $(COMPILE.c) -o $@ $<
1595 $(CTFCONVERT_O)

1597 $(OBJS_DIR)/zlib_obj.o: $(ZLIB_OBJS:%=$(OBJS_DIR)/%)
1598 $(LD) -r -Breduce -M$(UTSBASE)/common/zmod/mapfile -o $@ \
1599 $(ZLIB_OBJS:%=$(OBJS_DIR)/%)
1600 $(CTFMERGE) -t -f -L VERSION -o $@ $(ZLIB_OBJS:%=$(OBJS_DIR)/%)

1602 $(OBJS_DIR)/%.o: $(UTSBASE)/common/io/hxge/%.c
1603 $(COMPILE.c) -o $@ $<
1604 $(CTFCONVERT_O)

1606 $(OBJS_DIR)/%.o: $(UTSBASE)/common/io/tpm/%.c
1607 $(COMPILE.c) -o $@ $<
1608 $(CTFCONVERT_O)

1610 $(OBJS_DIR)/%.o: $(UTSBASE)/common/io/tpm/%.s
1611 $(COMPILE.s) -o $@ $<

1613 $(OBJS_DIR)/bz2%.o: $(COMMONBASE)/bzip2/%.c
1614 $(COMPILE.c) -o $@ -I$(COMMONBASE)/bzip2 $<
1615 $(CTFCONVERT_O)

1617 BZ2LINT = -erroff=%all -I$(UTSBASE)/common/bzip2

1619 $(LINTS_DIR)/bz2%.ln: $(COMMONBASE)/bzip2/%.c
1620 @($LHEAD) $(LINT.c) -C $(LINTS_DIR)/`basename $@ .ln` $(BZ2LINT) $< $(

1622 #
1623 # Section lb: Lint `objects`
1624 #
1625 $(LINTS_DIR)/%.ln: $(COMMONBASE)/crypto/aes/%.c
1626 @($LHEAD) $(LINT.c) $< $(LTAIL))

1628 $(LINTS_DIR)/%.ln: $(COMMONBASE)/crypto/arcfour/%.c
1629 @($LHEAD) $(LINT.c) $< $(LTAIL))

1631 $(LINTS_DIR)/%.ln: $(COMMONBASE)/crypto/blowfish/%.c
1632 @($LHEAD) $(LINT.c) $< $(LTAIL))

1634 $(LINTS_DIR)/%.ln: $(COMMONBASE)/crypto/ecc/%.c
1635 @($LHEAD) $(LINT.c) $< $(LTAIL))

1637 $(LINTS_DIR)/%.ln: $(COMMONBASE)/crypto/modes/%.c
1638 @($LHEAD) $(LINT.c) $< $(LTAIL))

1640 $(LINTS_DIR)/%.ln: $(COMMONBASE)/crypto/padding/%.c
1641 @($LHEAD) $(LINT.c) $< $(LTAIL))

```

```

1643 $(LINTS_DIR)/%.ln: $(COMMONBASE)/crypto/rng/%.c
1644 @($LHEAD) $(LINT.c) $< $(LTAIL))

1646 $(LINTS_DIR)/%.ln: $(COMMONBASE)/crypto/rsa/%.c
1647 @($LHEAD) $(LINT.c) $< $(LTAIL))

1649 $(LINTS_DIR)/%.ln: $(COMMONBASE)/bignum/%.c
1650 @($LHEAD) $(LINT.c) $< $(LTAIL))

1652 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/bignum/%.c
1653 @($LHEAD) $(LINT.c) $< $(LTAIL))

1655 $(LINTS_DIR)/%.ln: $(COMMONBASE)/mpi/%.c
1656 @($LHEAD) $(LINT.c) $< $(LTAIL))

1658 $(LINTS_DIR)/%.ln: $(COMMONBASE)/acl/%.c
1659 @($LHEAD) $(LINT.c) $< $(LTAIL))

1661 $(LINTS_DIR)/%.ln: $(COMMONBASE)/avl/%.c
1662 @($LHEAD) $(LINT.c) $< $(LTAIL))

1664 $(LINTS_DIR)/%.ln: $(COMMONBASE)/ucode/%.c
1665 @($LHEAD) $(LINT.c) $< $(LTAIL))

1667 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/brand/snl/%.c
1668 @($LHEAD) $(LINT.c) $< $(LTAIL))

1670 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/brand/solaris10/%.c
1671 @($LHEAD) $(LINT.c) $< $(LTAIL))

1673 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/c2/%.c
1674 @($LHEAD) $(LINT.c) $< $(LTAIL))

1676 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/conf/%.c
1677 @($LHEAD) $(LINT.c) $< $(LTAIL))

1679 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/contract/%.c
1680 @($LHEAD) $(LINT.c) $< $(LTAIL))

1682 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/cpr/%.c
1683 @($LHEAD) $(LINT.c) $< $(LTAIL))

1685 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/ctf/%.c
1686 @($LHEAD) $(LINT.c) $< $(LTAIL))

1688 $(LINTS_DIR)/%.ln: $(COMMONBASE)/ctf/%.c
1689 @($LHEAD) $(LINT.c) $< $(LTAIL))

1691 $(LINTS_DIR)/%.ln: $(COMMONBASE)/pci/%.c
1692 @($LHEAD) $(LINT.c) $< $(LTAIL))

1694 $(LINTS_DIR)/%.ln: $(COMMONBASE)/devid/%.c
1695 @($LHEAD) $(LINT.c) $< $(LTAIL))

1697 $(LINTS_DIR)/%.ln: $(COMMONBASE)/crypto/des/%.c
1698 @($LHEAD) $(LINT.c) $< $(LTAIL))

1700 $(LINTS_DIR)/%.ln: $(COMMONBASE)/secflags/%.c
1701 @($LHEAD) $(LINT.c) $< $(LTAIL))

1703 #endif /* ! codereview */
1704 $(LINTS_DIR)/%.ln: $(COMMONBASE)/smbios/%.c
1705 @($LHEAD) $(LINT.c) $< $(LTAIL))

1707 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/avs/ncall/%.c
1708 @($LHEAD) $(LINT.c) $< $(LTAIL))

```

```

1710 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/avs/ns/dsw/%.c
1711     @$(LHEAD) $(LINT.c) $< $(LTAIL))

1713 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/avs/ns/nsctl/%.c
1714     @$(LHEAD) $(LINT.c) $< $(LTAIL))

1716 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/avs/ns/rdc/%.c
1717     @$(LHEAD) $(LINT.c) $< $(LTAIL))

1719 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/avs/ns/sdbc/%.c
1720     @$(LHEAD) $(LINT.c) $< $(LTAIL))

1722 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/avs/ns/solaris/%.c
1723     @$(LHEAD) $(LINT.c) $< $(LTAIL))

1725 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/avs/ns/sv/%.c
1726     @$(LHEAD) $(LINT.c) $< $(LTAIL))

1728 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/avs/ns/unistat/%.c
1729     @$(LHEAD) $(LINT.c) $< $(LTAIL))

1731 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/des/%.c
1732     @$(LHEAD) $(LINT.c) $< $(LTAIL))

1734 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/crypto/api/%.c
1735     @$(LHEAD) $(LINT.c) $< $(LTAIL))

1737 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/crypto/core/%.c
1738     @$(LHEAD) $(LINT.c) $< $(LTAIL))

1740 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/crypto/io/%.c
1741     @$(LHEAD) $(LINT.c) $< $(LTAIL))

1743 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/crypto/spi/%.c
1744     @$(LHEAD) $(LINT.c) $< $(LTAIL))

1746 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/disp/%.c
1747     @$(LHEAD) $(LINT.c) $< $(LTAIL))

1749 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/dtrace/%.c
1750     @$(LHEAD) $(LINT.c) $< $(LTAIL))

1752 $(LINTS_DIR)/%.ln:      $(COMMONBASE)/exacct/%.c
1753     @$(LHEAD) $(LINT.c) $< $(LTAIL))

1755 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/exec/aout/%.c
1756     @$(LHEAD) $(LINT.c) $< $(LTAIL))

1758 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/exec/elf/%.c
1759     @$(LHEAD) $(LINT.c) $< $(LTAIL))

1761 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/exec/intp/%.c
1762     @$(LHEAD) $(LINT.c) $< $(LTAIL))

1764 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/exec/shbin/%.c
1765     @$(LHEAD) $(LINT.c) $< $(LTAIL))

1767 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/exec/java/%.c
1768     @$(LHEAD) $(LINT.c) $< $(LTAIL))

1770 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/%.c
1771     @$(LHEAD) $(LINT.c) $< $(LTAIL))

1773 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/autofs/%.c
1774     @$(LHEAD) $(LINT.c) $< $(LTAIL))

```

```

1776 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/bootfs/%.c
1777     @$(LHEAD) $(LINT.c) $< $(LTAIL))

1779 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/ctfs/%.c
1780     @$(LHEAD) $(LINT.c) $< $(LTAIL))

1782 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/doorfs/%.c
1783     @$(LHEAD) $(LINT.c) $< $(LTAIL))

1785 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/dcfs/%.c
1786     @$(LHEAD) $(LINT.c) $< $(LTAIL))

1788 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/devfs/%.c
1789     @$(LHEAD) $(LINT.c) $< $(LTAIL))

1791 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/dev/%.c
1792     @$(LHEAD) $(LINT.c) $< $(LTAIL))

1794 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/fd/%.c
1795     @$(LHEAD) $(LINT.c) $< $(LTAIL))

1797 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/fifofs/%.c
1798     @$(LHEAD) $(LINT.c) $< $(LTAIL))

1800 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/hsfs/%.c
1801     @$(LHEAD) $(LINT.c) $< $(LTAIL))

1803 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/lofs/%.c
1804     @$(LHEAD) $(LINT.c) $< $(LTAIL))

1806 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/mntfs/%.c
1807     @$(LHEAD) $(LINT.c) $< $(LTAIL))

1809 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/namefs/%.c
1810     @$(LHEAD) $(LINT.c) $< $(LTAIL))

1812 $(LINTS_DIR)/%.ln:      $(COMMONBASE)/smbsrv/%.c
1813     @$(LHEAD) $(LINT.c) $< $(LTAIL))

1815 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/smbsrv/%.c
1816     @$(LHEAD) $(LINT.c) $< $(LTAIL))

1818 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/nfs/%.c
1819     @$(LHEAD) $(LINT.c) $< $(LTAIL))

1821 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/objfs/%.c
1822     @$(LHEAD) $(LINT.c) $< $(LTAIL))

1824 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/pcfs/%.c
1825     @$(LHEAD) $(LINT.c) $< $(LTAIL))

1827 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/portfs/%.c
1828     @$(LHEAD) $(LINT.c) $< $(LTAIL))

1830 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/proc/%.c
1831     @$(LHEAD) $(LINT.c) $< $(LTAIL))

1833 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/sharefs/%.c
1834     @$(LHEAD) $(LINT.c) $< $(LTAIL))

1836 $(LINTS_DIR)/%.ln:      $(COMMONBASE)/smblnt/%.c
1837     @$(LHEAD) $(LINT.c) $< $(LTAIL))

1839 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/fs/smbclnt/netsmb/%.c
1840     @$(LHEAD) $(LINT.c) $< $(LTAIL))

```

```

1842 $(LINTSDIR)/%.ln:          $(UTSBASE)/common/fs/smbclnt/smbfs/%.c
1843     @$(LHEAD) $(LINT.c) $< $(LTAIL))

1845 $(LINTSDIR)/%.ln:          $(UTSBASE)/common/fs/sockfs/%.c
1846     @$(LHEAD) $(LINT.c) $< $(LTAIL))

1848 $(LINTSDIR)/%.ln:          $(UTSBASE)/common/fs/specfs/%.c
1849     @$(LHEAD) $(LINT.c) $< $(LTAIL))

1851 $(LINTSDIR)/%.ln:          $(UTSBASE)/common/fs/swapfs/%.c
1852     @$(LHEAD) $(LINT.c) $< $(LTAIL))

1854 $(LINTSDIR)/%.ln:          $(UTSBASE)/common/fs/tmpfs/%.c
1855     @$(LHEAD) $(LINT.c) $< $(LTAIL))

1857 $(LINTSDIR)/%.ln:          $(UTSBASE)/common/fs/udfs/%.c
1858     @$(LHEAD) $(LINT.c) $< $(LTAIL))

1860 $(LINTSDIR)/%.ln:          $(UTSBASE)/common/fs/ufs/%.c
1861     @$(LHEAD) $(LINT.c) $< $(LTAIL))

1863 $(LINTSDIR)/%.ln:          $(UTSBASE)/common/fs/ufs_log/%.c
1864     @$(LHEAD) $(LINT.c) $< $(LTAIL))

1866 $(LINTSDIR)/%.ln:          $(UTSBASE)/common/io/vscan/%.c
1867     @$(LHEAD) $(LINT.c) $< $(LTAIL))

1869 $(LINTSDIR)/%.ln:          $(UTSBASE)/common/fs/zfs/%.c
1870     @$(LHEAD) $(LINT.c) $< $(LTAIL))

1872 $(LINTSDIR)/%.ln:          $(UTSBASE)/common/fs/zut/%.c
1873     @$(LHEAD) $(LINT.c) $< $(LTAIL))

1875 $(LINTSDIR)/%.ln:          $(COMMONBASE)/xattr/%.c
1876     @$(LHEAD) $(LINT.c) $< $(LTAIL))

1878 $(LINTSDIR)/%.ln:          $(COMMONBASE)/zfs/%.c
1879     @$(LHEAD) $(LINT.c) $< $(LTAIL))

1881 $(LINTSDIR)/%.ln:          $(UTSBASE)/common/gssapi/%.c
1882     @$(LHEAD) $(LINT.c) $(KGSSDFLAGS) $< $(LTAIL))

1884 $(LINTSDIR)/%.ln:          $(UTSBASE)/common/gssapi/mechs/dummy/%.c
1885     @$(LHEAD) $(LINT.c) $(KGSSDFLAGS) $< $(LTAIL))

1887 $(LINTSDIR)/%.ln:          $(KMECHKRB5_BASE)/%.c
1888     @$(LHEAD) $(LINT.c) $(KGSSDFLAGS) $< $(LTAIL))

1890 $(LINTSDIR)/%.ln:          $(KMECHKRB5_BASE)/crypto/%.c
1891     @$(LHEAD) $(LINT.c) $(KGSSDFLAGS) $< $(LTAIL))

1893 $(LINTSDIR)/%.ln:          $(KMECHKRB5_BASE)/crypto/des/%.c
1894     @$(LHEAD) $(LINT.c) $(KGSSDFLAGS) $< $(LTAIL))

1896 $(LINTSDIR)/%.ln:          $(KMECHKRB5_BASE)/crypto/dk/%.c
1897     @$(LHEAD) $(LINT.c) $(KGSSDFLAGS) $< $(LTAIL))

1899 $(LINTSDIR)/%.ln:          $(KMECHKRB5_BASE)/crypto/os/%.c
1900     @$(LHEAD) $(LINT.c) $(KGSSDFLAGS) $< $(LTAIL))

1902 $(LINTSDIR)/%.ln:          $(KMECHKRB5_BASE)/crypto/arcfour/%.c
1903     @$(LHEAD) $(LINT.c) $(KGSSDFLAGS) $< $(LTAIL))

1905 $(LINTSDIR)/%.ln:          $(KMECHKRB5_BASE)/crypto/enc_provider/%.c
1906     @$(LHEAD) $(LINT.c) $(KGSSDFLAGS) $< $(LTAIL))

```

```

1908 $(LINTSDIR)/%.ln:          $(KMECHKRB5_BASE)/crypto/hash_provider/%.c
1909     @$(LHEAD) $(LINT.c) $(KGSSDFLAGS) $< $(LTAIL))

1911 $(LINTSDIR)/%.ln:          $(KMECHKRB5_BASE)/crypto/keyhash_provider/%.c
1912     @$(LHEAD) $(LINT.c) $(KGSSDFLAGS) $< $(LTAIL))

1914 $(LINTSDIR)/%.ln:          $(KMECHKRB5_BASE)/crypto/raw/%.c
1915     @$(LHEAD) $(LINT.c) $(KGSSDFLAGS) $< $(LTAIL))

1917 $(LINTSDIR)/%.ln:          $(KMECHKRB5_BASE)/crypto/old/%.c
1918     @$(LHEAD) $(LINT.c) $(KGSSDFLAGS) $< $(LTAIL))

1920 $(LINTSDIR)/%.ln:          $(KMECHKRB5_BASE)/krb5/krb/%.c
1921     @$(LHEAD) $(LINT.c) $(KGSSDFLAGS) $< $(LTAIL))

1923 $(LINTSDIR)/%.ln:          $(KMECHKRB5_BASE)/krb5/os/%.c
1924     @$(LHEAD) $(LINT.c) $(KGSSDFLAGS) $< $(LTAIL))

1926 $(LINTSDIR)/%.ln:          $(KMECHKRB5_BASE)/mech/%.c
1927     @$(LHEAD) $(LINT.c) $(KGSSDFLAGS) $< $(LTAIL))

1929 $(LINTSDIR)/%.ln:          $(UTSBASE)/common/ldap/%.c
1930     @$(LHEAD) $(LINT.c) $< $(LTAIL))

1932 $(LINTSDIR)/%.ln:          $(UTSBASE)/common/inet/%.c
1933     @$(LHEAD) $(LINT.c) $< $(LTAIL))

1935 $(LINTSDIR)/%.ln:          $(UTSBASE)/common/inet/sockmods/%.c
1936     @$(LHEAD) $(LINT.c) $< $(LTAIL))

1938 $(LINTSDIR)/%.ln:          $(UTSBASE)/common/inet/arp/%.c
1939     @$(LHEAD) $(LINT.c) $< $(LTAIL))

1941 $(LINTSDIR)/%.ln:          $(UTSBASE)/common/inet/ip/%.c
1942     @$(LHEAD) $(LINT.c) $< $(LTAIL))

1944 $(LINTSDIR)/%.ln:          $(UTSBASE)/common/inet/ipnet/%.c
1945     @$(LHEAD) $(LINT.c) $< $(LTAIL))

1947 $(LINTSDIR)/%.ln:          $(UTSBASE)/common/inet/iptun/%.c
1948     @$(LHEAD) $(LINT.c) $< $(LTAIL))

1950 $(LINTSDIR)/%.ln:          $(UTSBASE)/common/inet/ipd/%.c
1951     @$(LHEAD) $(LINT.c) $< $(LTAIL))

1953 $(LINTSDIR)/%.ln:          $(UTSBASE)/common/inet/ipf/%.c
1954     @$(LHEAD) $(LINT.c) $(IPFFLAGS) $< $(LTAIL))

1956 $(LINTSDIR)/%.ln:          $(UTSBASE)/common/inet/kssl/%.c
1957     @$(LHEAD) $(LINT.c) $< $(LTAIL))

1959 $(LINTSDIR)/%.ln:          $(COMMONBASE)/net/patricia/%.c
1960     @$(LHEAD) $(LINT.c) $(IPFFLAGS) $< $(LTAIL))

1962 $(LINTSDIR)/%.ln:          $(UTSBASE)/common/inet/udp/%.c
1963     @$(LHEAD) $(LINT.c) $< $(LTAIL))

1965 $(LINTSDIR)/%.ln:          $(UTSBASE)/common/inet/sctp/%.c
1966     @$(LHEAD) $(LINT.c) $< $(LTAIL))

1968 $(LINTSDIR)/%.ln:          $(UTSBASE)/common/inet/tcp/%.c
1969     @$(LHEAD) $(LINT.c) $< $(LTAIL))

1971 $(LINTSDIR)/%.ln:          $(UTSBASE)/common/inet/ilb/%.c
1972     @$(LHEAD) $(LINT.c) $< $(LTAIL))

```

```

1974 $(LINTSDIR)/%.ln: $(UTSBASE)/common/inet/nca/%.c
1975     @$(LHEAD) $(LINT.c) $< $(LTAIL))

1977 $(LINTSDIR)/%.ln: $(UTSBASE)/common/inet/dlpistub/%.c
1978     @$(LHEAD) $(LINT.c) $< $(LTAIL))

1980 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/%.c
1981     @$(LHEAD) $(LINT.c) $< $(LTAIL))

1983 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/1394/%.c
1984     @$(LHEAD) $(LINT.c) $< $(LTAIL))

1986 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/1394/adapters/%.c
1987     @$(LHEAD) $(LINT.c) $< $(LTAIL))

1989 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/1394/targets/av1394/%.c
1990     @$(LHEAD) $(LINT.c) $< $(LTAIL))

1992 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/1394/targets/dcam1394/%.c
1993     @$(LHEAD) $(LINT.c) $< $(LTAIL))

1995 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/1394/targets/scsal394/%.c
1996     @$(LHEAD) $(LINT.c) $< $(LTAIL))

1998 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/sbp2/%.c
1999     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2001 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/aac/%.c
2002     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2004 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/afe/%.c
2005     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2007 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/atge/%.c
2008     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2010 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/arn/%.c
2011     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2013 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/ath/%.c
2014     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2016 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/atu/%.c
2017     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2019 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/audio/impl/%.c
2020     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2022 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/audio/ac97/%.c
2023     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2025 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/audio/drv/audio1575/%.c
2026     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2028 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/audio/drv/audio810/%.c
2029     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2031 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/audio/drv/audiocmi/%.c
2032     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2034 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/audio/drv/audiocmihd/%.c
2035     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2037 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/audio/drv/audioens/%.c
2038     @$(LHEAD) $(LINT.c) $< $(LTAIL))

```

```

2040 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/audio/drv/audioemu10k/%.c
2041     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2043 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/audio/drv/audiohd/%.c
2044     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2046 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/audio/drv/audioixp/%.c
2047     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2049 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/audio/drv/audiols/%.c
2050     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2052 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/audio/drv/audiopci/%.c
2053     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2055 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/audio/drv/audiop16x/%.c
2056     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2058 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/audio/drv/audiosolo/%.c
2059     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2061 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/audio/drv/audiots/%.c
2062     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2064 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/audio/drv/audiovia823x/%.c
2065     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2067 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/audio/drv/audiovia97/%.c
2068     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2070 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/bfe/%.c
2071     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2073 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/bpf/%.c
2074     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2076 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/bge/%.c
2077     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2079 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/blkdev/%.c
2080     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2082 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/bnxe/%.c
2083     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2085 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/bnxe/577xx/common/%.c
2086     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2088 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/bnxe/577xx/drivers/common/e
2089     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2091 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/bnxe/577xx/drivers/common/l
2092     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2094 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/bnxe/577xx/drivers/common/l
2095     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2097 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/bnxe/577xx/drivers/common/l
2098     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2100 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/bnxe/577xx/drivers/common/l
2101     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2103 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/bnxe/577xx/drivers/common/l
2104     @$(LHEAD) $(LINT.c) $< $(LTAIL))

```

```

2106 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/cardbus/%.c
2107     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2109 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/comstar/lu/stmf_sbd/%.c
2110     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2112 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/comstar/port/fct/%.c
2113     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2115 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/comstar/port/qlt/%.c
2116     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2118 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/comstar/port/srpt/%.c
2119     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2121 $(LINTS_DIR)/%.ln: $(COMMONBASE)/iscsit/%.c
2122     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2124 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/comstar/port/fcoet/%.c
2125     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2127 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/comstar/port/iscsit/%.c
2128     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2130 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/comstar/port/pppt/%.c
2131     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2133 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/comstar/stmf/%.c
2134     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2136 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/cpgary3/%.c
2137     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2139 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/dld/%.c
2140     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2142 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/dls/%.c
2143     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2145 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/dmfe/%.c
2146     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2148 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/drm/%.c
2149     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2151 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/efe/%.c
2152     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2154 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/elx1/%.c
2155     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2157 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/fcoe/%.c
2158     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2160 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/hme/%.c
2161     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2163 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/pciex/%.c
2164     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2166 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/hotplug/hpcsvc/%.c
2167     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2169 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/pciex/hotplug/%.c
2170     @$(LHEAD) $(LINT.c) $< $(LTAIL))

```

```

2172 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/hotplug/pcihp/%.c
2173     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2175 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/ib/clients/rds/%.c
2176     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2178 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/ib/clients/rds3/%.c
2179     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2181 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/ib/clients/iser/%.c
2182     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2184 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/ib/clients/ibd/%.c
2185     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2187 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/ib/clients/eoib/%.c
2188     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2190 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/ib/clients/of/sol_ofs/%.c
2191     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2193 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/ib/clients/of/sol_ucma/%.c
2194     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2196 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/ib/clients/of/sol_umad/%.c
2197     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2199 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/ib/clients/of/sol_uverbs/%.c
2200     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2202 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/ib/clients/sdp/%.c
2203     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2205 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/ib/mgt/ibcm/%.c
2206     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2208 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/ib/mgt/ibdm/%.c
2209     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2211 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/ib/mgt/ibdma/%.c
2212     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2214 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/ib/mgt/ibmf/%.c
2215     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2217 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/ib/ibnex/%.c
2218     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2220 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/ib/ibt1/%.c
2221     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2223 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/ib/adapters/tavor/%.c
2224     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2226 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/ib/adapters/hermon/%.c
2227     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2229 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/ib/clients/daplt/%.c
2230     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2232 $(LINTS_DIR)/%.ln: $(COMMONBASE)/iscsi/%.c
2233     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2235 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/idm/%.c
2236     @$(LHEAD) $(LINT.c) $< $(LTAIL))

```

```

2238 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/ipw/%.c
2239     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2241 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/iwh/%.c
2242     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2244 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/iwi/%.c
2245     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2247 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/iwk/%.c
2248     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2250 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/iwp/%.c
2251     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2253 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/kb8042/%.c
2254     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2256 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/kbtrans/%.c
2257     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2259 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/ksocket/%.c
2260     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2262 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/aggr/%.c
2263     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2265 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/lp/%.c
2266     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2268 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/lvm/hotspares/%.c
2269     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2271 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/lvm/md/%.c
2272     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2274 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/lvm/mirror/%.c
2275     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2277 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/lvm/raid/%.c
2278     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2280 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/lvm/softpart/%.c
2281     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2283 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/lvm/striped/%.c
2284     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2286 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/lvm/notify/%.c
2287     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2289 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/lvm/trans/%.c
2290     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2292 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/mac/%.c
2293     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2295 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/mac/plugins/%.c
2296     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2298 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/mega_sas/%.c
2299     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2301 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/mii/%.c
2302     @$(LHEAD) $(LINT.c) $< $(LTAIL))

```

```

2304 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/mr_sas/%.c
2305     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2307 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/scsi/adapters/mpt_sas/%.c
2308     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2310 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/mxufe/%.c
2311     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2313 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/mwl/%.c
2314     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2316 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/mwl/mwl_fw/%.c
2317     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2319 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/net80211/%.c
2320     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2322 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/nge/%.c
2323     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2325 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/nvme/%.c
2326     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2328 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/nxge/%.c
2329     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2331 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/nxge/.s
2332     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2334 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/nxge/npi/%.c
2335     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2337 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/pci-ide/%.c
2338     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2340 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/pcn/%.c
2341     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2343 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/ppp/sppp/%.c
2344     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2346 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/ppp/sppasyn/%.c
2347     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2349 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/ppp/spptun/%.c
2350     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2352 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/ral/%.c
2353     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2355 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/rge/%.c
2356     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2358 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/rtls/%.c
2359     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2361 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/rsm/%.c
2362     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2364 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/rtw/%.c
2365     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2367 $(LINTS_DIR)/%.ln: $(UTSBASE)/common/io/rum/%.c
2368     @$(LHEAD) $(LINT.c) $< $(LTAIL))

```

```

2370 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/rwd/%.c
2371     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2373 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/rwn/%.c
2374     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2376 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/sata/adapters/ahci/%.c
2377     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2379 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/sata/adapters/nv_sata/%.c
2380     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2382 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/sata/adapters/si3124/%.c
2383     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2385 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/sata/impl/%.c
2386     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2388 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/scsi/adapters/%.c
2389     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2391 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/scsi/adapters/blk2scsa/%.c
2392     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2394 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/scsi/adapters/pmcs/%.c
2395     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2397 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/scsi/adapters/scsi_vhci/%.c
2398     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2400 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/scsi/adapters/scsi_vhci/fop
2401     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2403 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/fibre-channel/ulp/%.c
2404     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2406 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/fibre-channel/impl/%.c
2407     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2409 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/fibre-channel/fca/qlc/%.c
2410     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2412 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/fibre-channel/fca/qlge/%.c
2413     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2415 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/fibre-channel/fca/emlxs/%.c
2416     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2418 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/fibre-channel/fca/oce/%.c
2419     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2421 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/fibre-channel/fca/fcoei/%.c
2422     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2424 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/scsi/conf/%.c
2425     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2427 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/scsi/impl/%.c
2428     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2430 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/scsi/targets/%.c
2431     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2433 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/sdcard/adapters/sdhost/%.c
2434     @$(LHEAD) $(LINT.c) $< $(LTAIL))

```

```

2436 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/sdcard/impl/%.c
2437     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2439 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/sdcard/targets/sdcard/%.c
2440     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2442 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/sfe/%.c
2443     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2445 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/simnet/%.c
2446     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2448 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/softmac/%.c
2449     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2451 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/uath/%.c
2452     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2454 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/uath/uath_fw/%.c
2455     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2457 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/ural/%.c
2458     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2460 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/urtw/%.c
2461     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2463 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/usb/clients/audio/usb_ac/%.
2464     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2466 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/usb/clients/audio/usb_as/%.
2467     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2469 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/usb/clients/audio/usb_ah/%.
2470     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2472 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/usb/clients/usbskel/%.c
2473     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2475 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/usb/clients/video/usbvc/%.c
2476     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2478 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/usb/clients/hid/%.c
2479     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2481 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/usb/clients/hidparser/%.c
2482     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2484 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/usb/clients/usbkbm/%.c
2485     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2487 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/usb/clients/usbms/%.c
2488     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2490 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/usb/clients/usbinput/usbwcm
2491     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2493 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/usb/clients/ugen/%.c
2494     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2496 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/usb/clients/printer/%.c
2497     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2499 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/usb/clients/usbser/%.c
2500     @$(LHEAD) $(LINT.c) $< $(LTAIL))

```



```

2502 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/usb/clients/usbser/usbsacm/
2503     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2505 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/usb/clients/usbser/usbftdi/
2506     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2508 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/usb/clients/usbser/usbser_k
2509     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2511 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/usb/clients/usbser/usbsprl/
2512     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2514 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/usb/clients/usbecm/%.c
2515     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2517 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/usb/hcd/openhci/%.c
2518     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2520 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/usb/hcd/ehci/%.c
2521     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2523 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/usb/hcd/uhci/%.c
2524     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2526 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/usb/hubd/%.c
2527     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2529 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/usb/scsa2usb/%.c
2530     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2532 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/usb/usb_mid/%.c
2533     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2535 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/usb/usb_ia/%.c
2536     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2538 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/usb/usba/%.c
2539     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2541 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/usb/usba10/%.c
2542     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2544 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/vuidmice/%.c
2545     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2547 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/vnic/%.c
2548     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2550 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/wpi/%.c
2551     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2553 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/zyd/%.c
2554     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2556 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/chxge/com/%.c
2557     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2559 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/chxge/%.c
2560     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2562 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/cxgbe/common/%.c
2563     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2565 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/cxgbe/shared/%.c
2566     @$(LHEAD) $(LINT.c) $< $(LTAIL))

```

```

2568 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/cxgbe/firmware/%.c
2569     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2571 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/cxgbe/t4nex/%.c
2572     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2574 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/cxgbe/cxgbe/%.c
2575     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2577 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/ixgb/%.c
2578     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2580 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/xge/drv/%.c
2581     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2583 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/xge/hal/xgehal/%.c
2584     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2586 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/e1000g/%.c
2587     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2589 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/e1000api/%.c
2590     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2592 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/igb/%.c
2593     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2595 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/iprb/%.c
2596     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2598 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/ixgbe/%.c
2599     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2601 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/ixgbe/core/%.c
2602     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2604 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/ntxn/%.c
2605     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2607 $(LINTSDIR)/%.ln: $(UTSBASE)/common/io/myril0ge/drv/%.c
2608     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2610 $(LINTSDIR)/%.ln: $(UTSBASE)/common/ipp/%.c
2611     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2613 $(LINTSDIR)/%.ln: $(UTSBASE)/common/ipp/ipgpc/%.c
2614     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2616 $(LINTSDIR)/%.ln: $(UTSBASE)/common/ipp/dlcosmk/%.c
2617     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2619 $(LINTSDIR)/%.ln: $(UTSBASE)/common/ipp/flowacct/%.c
2620     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2622 $(LINTSDIR)/%.ln: $(UTSBASE)/common/ipp/dscpmk/%.c
2623     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2625 $(LINTSDIR)/%.ln: $(UTSBASE)/common/ipp/meters/%.c
2626     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2628 $(LINTSDIR)/%.ln: $(UTSBASE)/common/kiconv/kiconv_emea/%.c
2629     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2631 $(LINTSDIR)/%.ln: $(UTSBASE)/common/kiconv/kiconv_ja/%.c
2632     @$(LHEAD) $(LINT.c) $< $(LTAIL))

```

```

2634 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/kiconv/kiconv_ko/%.c
2635     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2637 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/kiconv/kiconv_sc/%.c
2638     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2640 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/kiconv/kiconv_tc/%.c
2641     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2643 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/klm/%.c
2644     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2646 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/kmdb/%.c
2647     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2649 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/krtld/%.c
2650     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2652 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/ktli/%.c
2653     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2655 $(LINTS_DIR)/%.ln:      $(COMMONBASE)/list/%.c
2656     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2658 $(LINTS_DIR)/%.ln:      $(COMMONBASE)/lvm/%.c
2659     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2661 $(LINTS_DIR)/%.ln:      $(COMMONBASE)/lzma/%.c
2662     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2664 $(LINTS_DIR)/%.ln:      $(COMMONBASE)/crypto/md4/%.c
2665     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2667 $(LINTS_DIR)/%.ln:      $(COMMONBASE)/crypto/md5/%.c
2668     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2670 $(LINTS_DIR)/%.ln:      $(COMMONBASE)/net/dhcp/%.c
2671     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2673 $(LINTS_DIR)/%.ln:      $(COMMONBASE)/nvpair/%.c
2674     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2676 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/os/%.c
2677     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2679 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/rpc/%.c
2680     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2682 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/pcmcia/cs/%.c
2683     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2685 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/pcmcia/cis/%.c
2686     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2688 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/pcmcia/nexus/%.c
2689     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2691 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/pcmcia/pcs/%.c
2692     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2694 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/rpc/%.c
2695     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2697 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/rpc/sec/%.c
2698     @$(LHEAD) $(LINT.c) $< $(LTAIL))

```

```

2700 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/rpc/sec_gss/%.c
2701     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2703 $(LINTS_DIR)/%.ln:      $(COMMONBASE)/crypto/edonr/%.c
2704     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2706 $(LINTS_DIR)/%.ln:      $(COMMONBASE)/crypto/shal/%.c
2707     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2709 $(LINTS_DIR)/%.ln:      $(COMMONBASE)/crypto/sha2/%.c
2710     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2712 $(LINTS_DIR)/%.ln:      $(COMMONBASE)/crypto/skein/%.c
2713     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2715 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/syscall/%.c
2716     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2718 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/tnf/%.c
2719     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2721 $(LINTS_DIR)/%.ln:      $(COMMONBASE)/tsol/%.c
2722     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2724 $(LINTS_DIR)/%.ln:      $(COMMONBASE)/util/%.c
2725     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2727 $(LINTS_DIR)/%.ln:      $(COMMONBASE)/unicode/%.c
2728     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2730 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/vm/%.c
2731     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2733 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/scsi/adapters/iscsi/%.c
2734     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2736 $(LINTS_DIR)/%.ln:      $(COMMONBASE)/iscsi/%.c
2737     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2739 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/inet/kifconf/%.c
2740     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2742 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/virtio/%.c
2743     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2745 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/vioblk/%.c
2746     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2748 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/vioif/%.c
2749     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2751 ZMODLINTFLAGS = -eroff=E_CONSTANT_CONDITION

2753 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/zmod/%.c
2754     @$(LHEAD) $(LINT.c) $(ZMODLINTFLAGS) $< $(LTAIL))

2756 $(LINTS_DIR)/zlib_obj.ln: $(ZLIB_OBJS:%.o=$(LINTS_DIR)/%.ln) \
2757     $(UTSBASE)/common/zmod/zlib_lint.c
2758     @$(LHEAD) $(LINT.c) -C $(LINTS_DIR)/zlib_obj \
2759     $(UTSBASE)/common/zmod/zlib_lint.c $(LTAIL))

2761 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/hxge/%.c
2762     @$(LHEAD) $(LINT.c) $< $(LTAIL))

2764 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/tpm/%.c

```

```
2765      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2767 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/tpm/%.s
2768      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2770 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/vr/%.c
2771      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2773 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/yge/%.c
2774      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2776 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/sfxge/%.c
2777      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2779 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/sfxge/common/%.c
2780      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2782 $(LINTS_DIR)/%.ln:      $(UTSBASE)/common/io/skd/%.c
2783      @$(LHEAD) $(LINT.c) $< $(LTAIL))
2785 $(LINTS_DIR)/%.ln:      $(COMMONBASE)/fsreparse/%.c
2786      @$(LHEAD) $(LINT.c) $< $(LTAIL))
```

```

*****
53406 Thu Jun 30 22:03:11 2016
new/usr/src/uts/common/c2/audit.c
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
_____unchanged_portion_omitted_____

709 /*
710 * ROUTINE:      AUDIT_CLOSEF
711 * PURPOSE:
712 * CALLBY:      CLOSEF
713 * NOTE:
714 * release per file audit resources when file structure is being released.
715 *
716 * IMPORTANT NOTE: Since we generate an audit record here, we may sleep
717 * on the audit queue if it becomes full. This means
718 * audit_closef can not be called when f_count == 0. Since
719 * f_count == 0 indicates the file structure is free, another
720 * process could attempt to use the file while we were still
721 * asleep waiting on the audit queue. This would cause the
722 * per file audit data to be corrupted when we finally do
723 * wakeup.
724 * TODO:
725 * QUESTION:
726 */

728 void
729 audit_closef(struct file *fp)
730 {
731     /* AUDIT_CLOSEF */
732     f_audit_data_t *fad;
733     t_audit_data_t *tad;
734     int success;
735     au_state_t estate;
736     struct vnode *vp;
737     token_t *ad = NULL;
738     struct vattr attr;
739     au_emod_t evmod = 0;
740     const auditinfo_addr_t *ainfo;
741     cred_t *cr;
742     au_kcontext_t *kctx = GET_KCTX_PZ;
743     uint32_t auditing;
744     boolean_t audit_attr = B_FALSE;

745     fad = F2A(fp);
746     estate = kctx->aук_ets[AUE_CLOSE];
747     tad = U2A(u);
748     cr = CRED();

750     /* audit record already generated by system call envelope */
751     if (tad->tad_event == AUE_CLOSE) {
752         /* so close audit event will have bits set */
753         tad->tad_evmod |= (au_emod_t)fad->fad_flags;
754         return;
755     }

757     /* if auditing not enabled, then don't generate an audit record */
758     auditing = (tad->tad_audit == AUC_UNSET) ?
759         kctx->aук_auditstate : tad->tad_audit;
760     if (auditing & ~(AUC_AUDITING | AUC_INIT_AUDIT | AUC_NOSPACE))
761         return;

763     ainfo = crgetauinfo(cr);

```

```

764     if (ainfo == NULL)
765         return;

767     success = ainfo->ai_mask.as_success & estate;

769     /* not selected for this event */
770     if (success == 0)
771         return;

773     /*
774     * can't use audit_attributes here since we use a private audit area
775     * to build the audit record instead of the one off the thread.
776     */
777     if ((vp = fp->f_vnode) != NULL) {
778         attr.va_mask = AT_ALL;
779         if (VOP_GETATTR(vp, &attr, 0, CRED(), NULL) == 0) {
780             if ((fp->f_flag & FWRITE) == 0 &&
781                 object_is_public(&attr)) {
782                 /*
783                 * When write was not used and the file can be
784                 * considered public, then skip the audit.
785                 */
786                 return;
787             }
788             audit_attr = B_TRUE;
789         }
790     }

792     evmod = (au_emod_t)fad->fad_flags;
793     if (fad->fad_aupath != NULL) {
794         au_write((caddr_t *)&(ad), au_to_path(fad->fad_aupath));
795     } else {
796 #ifdef _LP64
797         au_write((caddr_t *)&(ad), au_to_arg64(
798             1, "no path: fp", (uint64_t)fp));
799 #else
800         au_write((caddr_t *)&(ad), au_to_arg32(
801             1, "no path: fp", (uint32_t)fp));
802 #endif
803     }

805     if (audit_attr) {
806         au_write((caddr_t *)&(ad), au_to_attr(&attr));
807         audit_sec_attributes((caddr_t *)&(ad), vp);
808     }

810     /* Add subject information */
811     AUDIT_SETSUBJ((caddr_t *)&(ad), cr, ainfo, kctx);

813     /* add a return token */
814     add_return_token((caddr_t *)&(ad), tad->tad_scid, 0, 0);

816     AS_INC(as_generated, 1, kctx);
817     AS_INC(as_kernel, 1, kctx);

819     /*
820     * Close up everything
821     * Note: path space recovery handled by normal system
822     * call envelope if not at last close.
823     * Note there is no failure at this point since
824     * this represents closes due to exit of process,
825     * thus we always indicate successful closes.
826     */
827     au_close(kctx, (caddr_t *)&(ad), AU_OK | AU_DEFER,
828         AUE_CLOSE, evmod, NULL);
829 }
_____unchanged_portion_omitted_____

```

```

1599 /*ARGSUSED*/
1600 void
1601 audit_fdsend(int fd, struct file *fp, int error)
1602 audit_fdsend(fd, fp, error)
1603     int fd;
1604     struct file *fp;
1605     int error;          /* ignore for now */
1606 {
1607     t_audit_data_t *tad; /* current thread */
1608     f_audit_data_t *fad; /* per file audit structure */
1609     struct vnode *vp;    /* for file attributes */
1610
1611     /* is this system call being audited */
1612     tad = U2A(u);
1613     ASSERT(tad != (t_audit_data_t *)0);
1614     if (!tad->tad_flag)
1615         return;
1616
1617     fad = F2A(fp);
1618
1619     /* add path and file attributes */
1620     if (fad != NULL && fad->fad_aupath != NULL) {
1621         au_uwrite(au_to_arg32(0, "send fd", (uint32_t)fd));
1622         au_uwrite(au_to_path(fad->fad_aupath));
1623     } else {
1624         au_uwrite(au_to_arg32(0, "send fd", (uint32_t)fd));
1625 #ifdef _LP64
1626         au_uwrite(au_to_arg64(0, "no path", (uint64_t)fp));
1627 #else
1628         au_uwrite(au_to_arg32(0, "no path", (uint32_t)fp));
1629 #endif
1630     }
1631     vp = fp->f_vnode; /* include vnode attributes */
1632     audit_attributes(vp);
1633 }
1634
1635 unchanged portion omitted
1636
1637 /*
1638 * Audit the psecflags() system call; the set name, current value, and delta
1639 * are put in the audit trail.
1640 */
1641 void
1642 audit_psecflags(proc_t *p,
1643     psecflagwhich_t which,
1644     const secflagdelta_t *psd)
1645 {
1646     t_audit_data_t *tad;
1647     secflagset_t new;
1648     const secflagset_t *old;
1649     const char *s;
1650     cred_t *cr;
1651     pid_t pid;
1652     const auditinfo_addr_t *ainfo;
1653     const psecflags_t *psec = &p->p_secflags;
1654
1655     tad = U2A(u);
1656
1657     if (tad->tad_flag == 0)
1658         return;
1659
1660     switch (which) {
1661     case PSF_EFFECTIVE:
1662         s = "effective";
1663         old = &psec->psf_effective;
1664         break;

```

```

1700     case PSF_INHERIT:
1701         s = "inherit";
1702         old = &psec->psf_inherit;
1703         break;
1704     case PSF_LOWER:
1705         s = "lower";
1706         old = &psec->psf_lower;
1707         break;
1708     case PSF_UPPER:
1709         s = "upper";
1710         old = &psec->psf_upper;
1711         break;
1712     }
1713
1714     secflags_copy(&new, old);
1715     secflags_apply_delta(&new, psd);
1716
1717     au_uwrite(au_to_secflags(s, *old));
1718     au_uwrite(au_to_secflags(s, new));
1719
1720     ASSERT(mutex_owned(&p->p_lock));
1721     mutex_enter(&p->p_crlock);
1722
1723     pid = p->p_pid;
1724     crhold(cr = p->p_cred);
1725     mutex_exit(&p->p_crlock);
1726
1727     if ((ainfo = crgetauinfo(cr)) == NULL) {
1728         crfree(cr);
1729         return;
1730     }
1731
1732     AUDIT_SETPROC_GENERIC(&(u_ad), cr, ainfo, pid);
1733
1734     crfree(cr);
1735 }
1736
1737 /*
1738 #endif /* ! codereview */
1739 * Audit the setpriv() system call; the operation, the set name and
1740 * the current value as well as the set argument are put in the
1741 * audit trail.
1742 */
1743 void
1744 audit_setpriv(int op, int set, const priv_set_t *newpriv, const cred_t *ocr)
1745 {
1746     t_audit_data_t *tad;
1747     const priv_set_t *oldpriv;
1748     priv_set_t report;
1749     const char *setname;
1750
1751     tad = U2A(u);
1752
1753     if (tad->tad_flag == 0)
1754         return;
1755
1756     oldpriv = priv_getset(ocr, set);
1757
1758     /* Generate the actual record, include the before and after */
1759     au_uwrite(au_to_arg32(2, "op", op));
1760     setname = priv_getsetbyname(set);
1761
1762     switch (op) {
1763     case PRIV_OFF:
1764         /* Report privileges actually switched off */
1765         report = *oldpriv;

```

```

1766     priv_intersect(newpriv, &report);
1767     au_uwrite(au_to_privset(setname, &report, AUT_PRIV, 0));
1768     break;
1769 case PRIV_ON:
1770     /* Report privileges actually switched on */
1771     report = *oldpriv;
1772     priv_inverse(&report);
1773     priv_intersect(newpriv, &report);
1774     au_uwrite(au_to_privset(setname, &report, AUT_PRIV, 0));
1775     break;
1776 case PRIV_SET:
1777     /* Report before and after */
1778     au_uwrite(au_to_privset(setname, oldpriv, AUT_PRIV, 0));
1779     au_uwrite(au_to_privset(setname, newpriv, AUT_PRIV, 0));
1780     break;
1781 }
1782 }

1784 /*
1785  * Dump the full device policy setting in the audit trail.
1786  */
1787 void
1788 audit_devpolicy(int nitems, const devplcysys_t *items)
1789 {
1790     t_audit_data_t *tad;
1791     int i;

1793     tad = U2A(u);

1795     if (tad->tad_flag == 0)
1796         return;

1798     for (i = 0; i < nitems; i++) {
1799         au_uwrite(au_to_arg32(2, "major", items[i].dps_maj));
1800         if (items[i].dps_minornm[0] == '\0') {
1801             au_uwrite(au_to_arg32(2, "lomin", items[i].dps_lomin));
1802             au_uwrite(au_to_arg32(2, "himin", items[i].dps_himin));
1803         } else
1804             au_uwrite(au_to_text(items[i].dps_minornm));

1806         au_uwrite(au_to_privset("read", &items[i].dps_rdp,
1807             AUT_PRIV, 0));
1808         au_uwrite(au_to_privset("write", &items[i].dps_wrp,
1809             AUT_PRIV, 0));
1810     }
1811 }

1813 /*ARGSUSED*/
1814 void
1815 audit_fdrecv(int fd, struct file *fp)
1816 audit_fdrecv(fd, fp)
1817     int fd;
1818     struct file *fp;
1819 {
1820     t_audit_data_t *tad; /* current thread */
1821     f_audit_data_t *fad; /* per file audit structure */
1822     struct vnode *vp; /* for file attributes */

1824     /* is this system call being audited */
1825     tad = U2A(u);
1826     ASSERT(tad != (t_audit_data_t *)0);
1827     if (!tad->tad_flag)
1828         return;

1829     fad = F2A(fp);

```

```

1829     /* add path and file attributes */
1830     if (fad != NULL && fad->fad_aupath != NULL) {
1831         au_uwrite(au_to_arg32(0, "recv fd", (uint32_t)fd));
1832         au_uwrite(au_to_path(fad->fad_aupath));
1833     } else {
1834         au_uwrite(au_to_arg32(0, "recv fd", (uint32_t)fd));
1835 #ifdef _LP64
1836         au_uwrite(au_to_arg64(0, "no path", (uint64_t)fp));
1837 #else
1838         au_uwrite(au_to_arg32(0, "no path", (uint32_t)fp));
1839 #endif
1840     }
1841     vp = fp->f_vnode; /* include vnode attributes */
1842     audit_attributes(vp);
1843 }
_____unchanged_portion_omitted_

```

new/usr/src/uts/common/c2/audit.h

1

```
*****
17213 Thu Jun 30 22:03:13 2016
new/usr/src/uts/common/c2/audit.h
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
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 (c) 1992, 2010, Oracle and/or its affiliates. All rights reserved.
23  */

25 /*
26  * This file contains the declarations of the various data structures
27  * used by the auditing module(s).
28  */

30 #ifndef _BSM_AUDIT_H
31 #define _BSM_AUDIT_H

33 #ifdef __cplusplus
34 extern "C" {
35 #endif

38 #include <sys/shm.h> /* for shmids structure */
39 #include <sys/sem.h> /* for semids structure */
40 #include <sys/msg.h> /* for msqid_ds structure */
41 #include <sys/atomic.h> /* using atomics */
42 #include <sys/secflags.h>
43 #endif /* ! codereview */

45 /*
46  * Audit conditions, statements regarding what's to be done with
47  * audit records. None of the "global state" is returned by an
48  * auditconfig -getcond call. AUC_NOSPACE no longer seems used.
49  */
50 /* global state */
51 #define AUC_UNSET 0 /* on/off hasn't been decided */
52 #define AUC_ENABLED 1 /* loaded and enabled */
53 /* pseudo state used in libbsm */
54 #define AUC_DISABLED 0x100 /* c2audit module is excluded */
55 /* local zone state */
56 #define AUC_AUDITING 0x1 /* audit daemon is active */
57 #define AUC_NOAUDIT 0x2 /* audit daemon is not active */
58 #define AUC_INIT_AUDIT 0x4 /* audit ready but auditd has not run */
```

new/usr/src/uts/common/c2/audit.h

2

```
59 #define AUC_NOSPACE 0x8 /* audit enabled, no space for audit records */

61 /*
62  * The user id -2 is never audited - in fact, a setaudit(AU_NOAUDITID)
63  * will turn off auditing.
64  */
65 #define AU_NOAUDITID ((au_id_t)-2)

67 /*
68  * success/failure bits for asynchronous events
69  */

71 #define AUM_SUCC 1 /* use the system success preselection mask */
72 #define AUM_FAIL 2 /* use the system failure preselection mask */

75 /*
76  * Defines for event modifier field
77  */
78 #define PAD_READ 0x0001 /* object read */
79 #define PAD_WRITE 0x0002 /* object write */
80 #define PAD_NONATTR 0x4000 /* non-attributable event */
81 #define PAD_FAILURE 0x8000 /* fail audit event */
82 #define PAD_SPRIVUSE 0x0080 /* successfully used privileged */
83 #define PAD_FPRIVUSE 0x0100 /* failed use of privileged */

85 /*
86  * Some typedefs for the fundamentals
87  */
88 typedef uint_t au_asid_t;
89 typedef uint_t au_class_t;
90 typedef ushort_t au_event_t;
91 typedef ushort_t au_emod_t;
92 typedef uid_t au_id_t;

94 /*
95  * An audit event mask.
96  */
97 #define AU_MASK_ALL 0xFFFFFFFF /* all bits on for unsigned int */
98 #define AU_MASK_NONE 0x0 /* all bits off = no:invalid class */

100 struct au_mask {
101     unsigned int am_success; /* success bits */
102     unsigned int am_failure; /* failure bits */
103 };
104 typedef struct au_mask au_mask_t;
105 #define as_success am_success
106 #define as_failure am_failure

108 /*
109  * The structure of the terminal ID (ipV4)
110  */
111 struct au_tid {
112     dev_t port;
113     uint_t machine;
114 };

116 #if defined(_SYSCALL32)
117 struct au_tid32 {
118     uint_t port;
119     uint_t machine;
120 };

122 typedef struct au_tid32 au_tid32_t;
123 #endif
```

```

125 typedef struct au_tid au_tid_t;
127 /*
128 * The structure of the terminal ID (ipv6)
129 */
130 struct au_tid_addr {
131     dev_t at_port;
132     uint_t at_type;
133     uint_t at_addr[4];
134 };
136 struct au_port_s {
137     uint32_t at_major; /* major # */
138     uint32_t at_minor; /* minor # */
139 };
140 typedef struct au_port_s au_port_t;
142 struct au_tid_addr64 {
143     au_port_t at_port;
144     uint_t at_type;
145     uint_t at_addr[4];
146 };
147 typedef struct au_tid_addr64 au_tid64_addr_t;
149 #if defined(_SYSCALL32)
150 struct au_tid_addr32 {
151     uint_t at_port;
152     uint_t at_type;
153     uint_t at_addr[4];
154 };
156 typedef struct au_tid_addr32 au_tid32_addr_t;
157 #endif
159 typedef struct au_tid_addr au_tid_addr_t;
161 struct au_ip {
162     uint16_t at_r_port; /* remote port */
163     uint16_t at_l_port; /* local port */
164     uint32_t at_type; /* AU_IPv4,... */
165     uint32_t at_addr[4]; /* remote IP */
166 };
167 typedef struct au_ip au_ip_t;
169 /*
170 * Generic network address structure
171 */
172 struct au_generic_tid {
173     uchar_t gt_type; /* AU_IPADR, AU_DEVICE,... */
174     union {
175         au_ip_t at_ip;
176         au_port_t at_dev;
177     } gt_addr;
178 };
179 typedef struct au_generic_tid au_generic_tid_t;
181 /*
182 * au_generic_tid_t gt_type values
183 * 0 is reserved for uninitialized data
184 */
185 #define AU_IPADR 1
186 #define AU_ETHER 2
187 #define AU_DEVICE 3
189 /*
190 * at_type values - address length used to identify address type

```

```

191 */
192 #define AU_IPv4 4 /* ipv4 type IP address */
193 #define AU_IPv6 16 /* ipv6 type IP address */
195 /*
196 * Compatability with SunOS 4.x BSM module
197 */
198 * New code should not contain audit_state_t,
199 * au_state_t, nor au_termid as these types
200 * may go away in future releases.
201 *
202 * typedef new-5.x-bsm-name old-4.x-bsm-name
203 */
205 typedef au_class_t au_state_t;
206 typedef au_mask_t audit_state_t;
207 typedef au_id_t auid_t;
208 #define ai_state ai_mask;
210 /*
211 * Opcodes for bsm system calls
212 */
214 #define BSM_GETAUID 19
215 #define BSM_SETAUID 20
216 #define BSM_GETAUDIT 21
217 #define BSM_SETAUDIT 22
218 /* 23 OBSOLETE */
219 /* 24 OBSOLETE */
220 #define BSM_AUDIT 25
221 /* 26 OBSOLETE */
222 /* 27 EOL announced for Sol 10 */
223 /* 28 OBSOLETE */
224 #define BSM_AUDITCTL 29
225 /* 30 OBSOLETE */
226 /* 31 OBSOLETE */
227 /* 32 OBSOLETE */
228 /* 33 OBSOLETE */
229 /* 34 OBSOLETE */
230 #define BSM_GETAUDIT_ADDR 35
231 #define BSM_SETAUDIT_ADDR 36
232 #define BSM_AUDITDOOR 37
234 /*
235 * auditon(2) commands
236 */
237 #define A_GETPOLICY 2 /* get audit policy */
238 #define A_SETPOLICY 3 /* set audit policy */
239 #define A_GETKMASK 4 /* get non-attributable event audit mask */
240 #define A_SETKMASK 5 /* set non-attributable event audit mask */
241 #define A_GETQCTRL 6 /* get kernel audit queue ctrl parameters */
242 #define A_SETQCTRL 7 /* set kernel audit queue ctrl parameters */
243 #define A_GETCWD 8 /* get process current working directory */
244 #define A_GETCAR 9 /* get process current active root */
245 #define A_GETSTAT 12 /* get audit statistics */
246 #define A_SETSTAT 13 /* (re)set audit statistics */
247 #define A_SETUMASK 14 /* set preselection mask for procs with auid */
248 #define A_SETSMASK 15 /* set preselection mask for procs with asid */
249 #define A_GETCOND 20 /* get audit system on/off condition */
250 #define A_SETCOND 21 /* set audit system on/off condition */
251 #define A_GETCLASS 22 /* get audit event to class mapping */
252 #define A_SETCLASS 23 /* set audit event to class mapping */
253 #define A_GETPINFO 24 /* get audit info for an arbitrary pid */
254 #define A_SETPMASK 25 /* set preselection mask for an given pid */
255 #define A_GETPINFO_ADDR 28 /* get audit info for an arbitrary pid */
256 #define A_GETKAUDIT 29 /* get kernel audit characteristics */

```



```

257 #define A_SETKAUDIT    30    /* set kernel audit characteristics */
258 #define A_GETAMASK    31    /* set user default audit event mask */
259 #define A_SETAMASK    32    /* get user default audit event mask */

261 /*
262  * Audit Policy parameters (32 bits)
263  */
264 #define AUDIT_CNT      0x0001 /* do NOT sleep undelivered synch events */
265 #define AUDIT_AHLT     0x0002 /* HALT machine on undelivered async event */
266 #define AUDIT_ARGV    0x0004 /* include argv with execv system call events */
267 #define AUDIT_ARGE    0x0008 /* include arge with execv system call events */
268 #define AUDIT_SEQ     0x0010 /* include sequence attribute */
269 #define AUDIT_GROUP   0x0040 /* include group attribute with each record */
270 #define AUDIT_TRAIL   0x0080 /* include trailer token */
271 #define AUDIT_PATH    0x0100 /* allow multiple paths per event */
272 #define AUDIT_SCNT    0x0200 /* sleep user events but not kernel events */
273 #define AUDIT_PUBLIC  0x0400 /* audit even "public" files */
274 #define AUDIT_ZONENAME 0x0800 /* emit zonename token */
275 #define AUDIT_PERZONE 0x1000 /* auditd and audit queue for each zone */
276 #define AUDIT_WINDATA_DOWN 0x2000 /* include paste downgraded data */
277 #define AUDIT_WINDATA_UP 0x4000 /* include paste upgraded data */

279 /*
280  * If AUDIT_GLOBAL changes, corresponding changes are required in
281  * audit_syscalls.c's setpolicy().
282  */
283 #define AUDIT_GLOBAL   (AUDIT_AHLT | AUDIT_PERZONE)
284 #define AUDIT_LOCAL    (AUDIT_CNT | AUDIT_ARGV | AUDIT_ARGE | \
285     AUDIT_SEQ | AUDIT_GROUP | AUDIT_TRAIL | AUDIT_PATH | \
286     AUDIT_PUBLIC | AUDIT_SCNT | AUDIT_ZONENAME | \
287     AUDIT_WINDATA_DOWN | AUDIT_WINDATA_UP)

289 /*
290  * Kernel audit queue control parameters
291  */
292  * audit record recording blocks at hiwater # undelivered records
293  * audit record recording resumes at lowwater # undelivered audit records
294  * bufisz determines how big the data xfers will be to the audit trail
295  */
296 struct au_qctrl {
297     size_t aq_hiwater;    /* kernel audit queue, high water mark */
298     size_t aq_lowater;   /* kernel audit queue, low water mark */
299     size_t aq_bufisz;    /* kernel audit queue, write size to trail */
300     clock_t aq_delay;    /* delay before flushing audit queue */
301 };

303 #if defined(_SYSCALL32)
304 struct au_qctrl32 {
305     size32_t aq_hiwater;
306     size32_t aq_lowater;
307     size32_t aq_bufisz;
308     clock32_t aq_delay;
309 };
310 #endif

313 /*
314  * default values of hiwater and lowater (note hi > lo)
315  */
316 #define AQ_HIWATER    100
317 #define AQ_MAXHIGH    100000
318 #define AQ_LOWATER    10
319 #define AQ_BUFISZ     8192
320 #define AQ_MAXBUFSZ   1048576
321 #define AQ_DELAY      20
322 #define AQ_MAXDELAY   20000

```

```

324 struct auditinfo {
325     au_id_t      ai_auid;
326     au_mask_t    ai_mask;
327     au_tid_t     ai_termid;
328     au_asid_t    ai_asid;
329 };

331 #if defined(_SYSCALL32)
332 struct auditinfo32 {
333     au_id_t      ai_auid;
334     au_mask_t    ai_mask;
335     au_tid32_t   ai_termid;
336     au_asid_t    ai_asid;
337 };

339 typedef struct auditinfo32 auditinfo32_t;
340 #endif

342 typedef struct auditinfo auditinfo_t;

344 struct k_auditinfo_addr {
345     au_id_t      ai_auid;
346     au_mask_t    ai_amask;    /* user default preselection mask */
347     au_mask_t    ai_namask;   /* non-attributable mask */
348     au_tid_addr_t ai_termid;
349     au_asid_t    ai_asid;
350 };
351 typedef struct k_auditinfo_addr k_auditinfo_addr_t;

353 struct auditinfo_addr {
354     au_id_t      ai_auid;
355     au_mask_t    ai_mask;
356     au_tid_addr_t ai_termid;
357     au_asid_t    ai_asid;
358 };

360 struct auditinfo_addr64 {
361     au_id_t      ai_auid;
362     au_mask_t    ai_mask;
363     au_tid64_addr_t ai_termid;
364     au_asid_t    ai_asid;
365 };
366 typedef struct auditinfo_addr64 auditinfo64_addr_t;

368 #if defined(_SYSCALL32)
369 struct auditinfo_addr32 {
370     au_id_t      ai_auid;
371     au_mask_t    ai_mask;
372     au_tid32_addr_t ai_termid;
373     au_asid_t    ai_asid;
374 };

376 typedef struct auditinfo_addr32 auditinfo32_addr_t;
377 #endif

379 typedef struct auditinfo_addr auditinfo_addr_t;

381 struct auditpinfo {
382     pid_t        ap_pid;
383     au_id_t      ap_auid;
384     au_mask_t    ap_mask;
385     au_tid_t     ap_termid;
386     au_asid_t    ap_asid;
387 };

```

```

389 #if defined(_SYSCALL32)
390 struct auditpinfo32 {
391     pid_t      ap_pid;
392     au_id_t    ap_auid;
393     au_mask_t  ap_mask;
394     au_tid32_t ap_termid;
395     au_asid_t  ap_asid;
396 };
397 #endif

400 struct auditpinfo_addr {
401     pid_t      ap_pid;
402     au_id_t    ap_auid;
403     au_mask_t  ap_mask;
404     au_tid_addr_t ap_termid;
405     au_asid_t  ap_asid;
406 };

408 #if defined(_SYSCALL32)
409 struct auditpinfo_addr32 {
410     pid_t      ap_pid;
411     au_id_t    ap_auid;
412     au_mask_t  ap_mask;
413     au_tid32_addr_t ap_termid;
414     au_asid_t  ap_asid;
415 };
416 #endif

419 struct au_evclass_map {
420     au_event_t  ec_number;
421     au_class_t  ec_class;
422 };
423 typedef struct au_evclass_map au_evclass_map_t;

425 /*
426  * Audit stat structures (used to be in audit_stat.h
427  */

429 struct audit_stat {
430     unsigned int as_version;          /* version of kernel audit code */
431     unsigned int as_numevent;        /* number of kernel audit events */
432     uint32_t as_generated;           /* # records processed */
433     uint32_t as_nonattrib;           /* # non-attributed records produced */
434     uint32_t as_kernel;              /* # records produced by kernel */
435     uint32_t as_audit;               /* # records processed by audit(2) */
436     uint32_t as_auditctl;            /* # records processed by auditctl(2) */
437     uint32_t as_enqueue;             /* # records put onto audit queue */
438     uint32_t as_written;              /* # records written to audit trail */
439     uint32_t as_wblocked;             /* # times write blked on audit queue */
440     uint32_t as_rblocked;             /* # times read blked on audit queue */
441     uint32_t as_dropped;              /* # of dropped audit records */
442     uint32_t as_totalsize;           /* total number bytes of audit data */
443     uint32_t as_memused;              /* no longer used */
444 };
445 typedef struct audit_stat au_stat_t;

447 /* get kernel audit context dependent on AUDIT_PERZONE policy */
448 #define GET_KCTX_PZ      (audit_policy & AUDIT_PERZONE) ? \
449     curproc->p_zone->zone_audit_kctxt : \
450     global_zone->zone_audit_kctxt
451 /* get kernel audit context of global zone */
452 #define GET_KCTX_GZ      global_zone->zone_audit_kctxt
453 /* get kernel audit context of non-global zone */
454 #define GET_KCTX_NGZ     curproc->p_zone->zone_audit_kctxt

```

```

456 #define AS_INC(a, b, c) atomic_add_32(&(c->auk_statistics.a), (b))
457 #define AS_DEC(a, b, c) atomic_add_32(&(c->auk_statistics.a), -(b))

459 /*
460  * audit token IPC types (shm, sem, msg) [for ipc attribute]
461  */

463 #define AT_IPC_MSG      ((char)1)      /* message IPC id */
464 #define AT_IPC_SEM     ((char)2)      /* semaphore IPC id */
465 #define AT_IPC_SHM     ((char)3)      /* shared memory IPC id */

467 #if defined(_KERNEL)

469 #ifdef __cplusplus
470 }
471 #endif

473 #include <sys/types.h>
474 #include <sys/model.h>
475 #include <sys/proc.h>
476 #include <sys/stream.h>
477 #include <sys/stropts.h>
478 #include <sys/file.h>
479 #include <sys/pathname.h>
480 #include <sys/vnode.h>
481 #include <sys/system.h>
482 #include <netinet/in.h>
483 #include <c2/audit_door_infc.h>
484 #include <sys/crypto/ioctladmin.h>
485 #include <sys/netstack.h>
486 #include <sys/zone.h>

488 #ifdef __cplusplus
489 extern "C" {
490 #endif

492 struct fcntla;
493 struct t_audit_data;
494 struct audit_path;
495 struct priv_set;
496 struct devplcysys;

498 struct auditcalls {
499     long code;
500     long a1;
501     long a2;
502     long a3;
503     long a4;
504     long a5;
505 };

507 int audit(caddr_t, int);
508 int auditsys(struct auditcalls *, union rval *); /* fake stub */
509 void audit_cryptoadm(int, char *, crypto_mech_name_t *,
510     uint_t, uint_t, uint32_t, int);
511 void audit_init(void);
512 void audit_init_module(void);
513 void audit_newproc(struct proc *);
514 void audit_pfree(struct proc *);
515 void audit_thread_create(kthread_id_t);
516 void audit_thread_free(kthread_id_t);
517 int audit_savepath(struct pathname *, struct vnode *, struct vnode *,
518     int, cred_t *);
519 void audit_anchorpath(struct pathname *, int);
520 void audit_symlink(struct pathname *, struct pathname *);

```

```

521 void    audit_symlink_create(struct vnode *, char *, char *, int);
522 int     object_is_public(struct vattr *);
523 void    audit_attributes(struct vnode *);
524 void    audit_falloc(struct file *);
525 void    audit_unfalloc(struct file *);
526 void    audit_exit(int, int);
527 void    audit_core_start(int);
528 void    audit_core_finish(int);
529 void    audit_strgetmsg(struct vnode *, struct strbuf *, struct strbuf *,
530        unsigned char *, int *, int);
531 void    audit_strputmsg(struct vnode *, struct strbuf *, struct strbuf *,
532        unsigned char, int, int);
533 void    audit_closef(struct file *);
534 void    audit_setf(struct file *, int);
535 void    audit_reboot(void);
536 void    audit_vncreate_start(void);
537 void    audit_setfsat_path(int argnum);
538 void    audit_vncreate_finish(struct vnode *, int);
539 void    audit_exec(const char *, const char *, ssize_t, ssize_t, cred_t *);
540 void    audit_enterprom(int);
541 void    audit_exitprom(int);
542 void    audit_chdirec(struct vnode *, struct vnode **);
543 void    audit_socket(int, struct queue *, struct msgb *, int);
544 int     audit_start(unsigned int, unsigned int, uint32_t, int, klpw_t *);
545 void    audit_finish(unsigned int, unsigned int, int, union rval *);
546 int     audit_async_start(label_t *, au_event_t, int);
547 void    audit_async_finish(caddr_t *, au_event_t, au_emod_t, timestruc_t *);
548 void    audit_async_discard_backend(void *);
549 void    audit_async_done(caddr_t *, int);
550 void    audit_async_drop(caddr_t *, int);

```

```

552 #ifndef AUK_CONTEXT_T
553 #define AUK_CONTEXT_T
554 typedef struct au_kcontext au_kcontext_t;
555 #endif

```

```

557 /* Zone audit context setup routine */
558 void au_zone_setup(void);

```

```

560 /*
561  * c2audit module states
562  */
563 #define C2AUDIT_DISABLED    0 /* c2audit module excluded in /etc/system */
564 #define C2AUDIT_UNLOADED    1 /* c2audit module not loaded */
565 #define C2AUDIT_LOADED     2 /* c2audit module loaded */

```

```

567 uint32_t    audit_getstate(void);
568 int         au_zone_getstate(const au_kcontext_t *);

```

```

570 /* The audit mask defining in which case is auditing enabled */
571 #define AU_AUDIT_MASK    (AUC_AUDITING | AUC_NOSPACE)

```

```

573 /*
574  * Get the given zone audit status. zcontext != NULL serves
575  * as a protection when c2audit module is not loaded.
576  */
577 #define AU_ZONE_AUDITING(zcontext)    \
578     (audit_active == C2AUDIT_LOADED && \
579      ((AU_AUDIT_MASK) & au_zone_getstate((zcontext))))

```

```

581 /*
582  * Get auditing status
583  */
584 #define AU_AUDITING() (audit_getstate())

```

```

586 int     audit_success(au_kcontext_t *, struct t_audit_data *, int, cred_t *);

```

```

587 int     auditme(au_kcontext_t *, struct t_audit_data *, au_state_t);
588 void    audit_fixpath(struct audit_path *, int);
589 void    audit_ipc(int, int, void *);
590 void    audit_ipcget(int, void *);
591 void    audit_fdsend(int, struct file *, int);
592 void    audit_fdrecv(int, struct file *);
593 void    audit_priv(int, const struct priv_set *, int);
594 void    audit_setppriv(int, int, const struct priv_set *, const cred_t *);
595 void    audit_psecflags(proc_t *, psecflagwhich_t,
596        const secflagdelta_t *);
597 #endif /* !codereview */
598 void    audit_devpolicy(int, const struct devplcysys *);
599 void    audit_update_context(proc_t *, cred_t *);
600 void    audit_kssl(int, void *, int);
601 void    audit_pf_policy(int, cred_t *, netstack_t *, char *, boolean_t, int,
602        pid_t);
603 void    audit_sec_attributes(caddr_t *, struct vnode *);

605 #endif

607 #ifdef __cplusplus
608 }
609 #endif

611 #endif /* _BSM_AUDIT_H */

```

```

*****
130912 Thu Jun 30 22:03:14 2016
new/usr/src/uts/common/c2/audit_event.c
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */

22 /*
23  * Copyright (c) 1992, 2010, Oracle and/or its affiliates. All rights reserved.
24  * Copyright (c) 2011 Bayard G. Bell. All rights reserved.
25 */

27 /*
28  * This file contains the audit event table used to control the production
29  * of audit records for each system call.
30 */

32 #include <sys/policy.h>
33 #include <sys/cred.h>
34 #include <sys/types.h>
35 #include <sys/system.h>
36 #include <sys/systeminfo.h>    /* for sysinfo auditing */
37 #include <sys/utsname.h>       /* for sysinfo auditing */
38 #include <sys/proc.h>
39 #include <sys/vnode.h>
40 #include <sys/mman.h>           /* for mmap(2) auditing etc. */
41 #include <sys/fcntl.h>
42 #include <sys/modctl.h>        /* for modctl auditing */
43 #include <sys/vnode.h>
44 #include <sys/user.h>
45 #include <sys/types.h>
46 #include <sys/processor.h>
47 #include <sys/procset.h>
48 #include <sys/acl.h>
49 #include <sys/ipc.h>
50 #include <sys/door.h>
51 #include <sys/sem.h>
52 #include <sys/msg.h>
53 #include <sys/shm.h>
54 #include <sys/kmem.h>
55 #include <sys/file.h>          /* for accept */
56 #include <sys/utssys.h>        /* for fuser */
57 #include <sys/tsol/label.h>
58 #include <sys/tsol/tndb.h>

```

```

59 #include <sys/tsol/tsyscall.h>
60 #include <c2/audit.h>
61 #include <c2/audit_kernel.h>
62 #include <c2/audit_events.h>
63 #include <c2/audit_record.h>
64 #include <sys/procset.h>
65 #include <nfs/mount.h>
66 #include <sys/param.h>
67 #include <sys/debug.h>
68 #include <sys/sysmacros.h>
69 #include <sys/stream.h>
70 #include <sys/strsubr.h>
71 #include <sys/stropts.h>
72 #include <sys/tihdr.h>
73 #include <sys/socket.h>
74 #include <sys/socketvar.h>
75 #include <sys/vfs_opreg.h>
76 #include <fs/sockfs/sockcommon.h>
77 #include <netinet/in.h>
78 #include <sys/ddi.h>
79 #include <sys/port_impl.h>
80 #include <sys/secflags.h>
81 #endif /* ! codereview */

83 static au_event_t    aui_fchowmat(au_event_t);
84 static au_event_t    aui_fchmodat(au_event_t);
85 static au_event_t    aui_open(au_event_t);
86 static au_event_t    aui_openat(au_event_t);
87 static au_event_t    aui_unlinkat(au_event_t);
88 static au_event_t    aui_fstatat(au_event_t);
89 static au_event_t    aui_msgsys(au_event_t);
90 static au_event_t    aui_shmsys(au_event_t);
91 static au_event_t    aui_semsys(au_event_t);
92 static au_event_t    aui_utssys(au_event_t);
93 static au_event_t    aui_fcntl(au_event_t);
94 static au_event_t    aui_execve(au_event_t);
95 static au_event_t    aui_memcntl(au_event_t);
96 static au_event_t    aui_sysinfo(au_event_t);
97 static au_event_t    aui_portfs(au_event_t);
98 static au_event_t    aui_auditsys(au_event_t);
99 static au_event_t    aui_modctl(au_event_t);
100 static au_event_t    aui_acl(au_event_t);
101 static au_event_t    aui_doorfs(au_event_t);
102 static au_event_t    aui_privsys(au_event_t);
103 static au_event_t    aui_forksys(au_event_t);
104 static au_event_t    aui_labelsys(au_event_t);
105 static au_event_t    aui_setpgrp(au_event_t);

108 #endif /* ! codereview */
109 static void    aus_exit(struct t_audit_data *);
110 static void    aus_open(struct t_audit_data *);
111 static void    aus_openat(struct t_audit_data *);
112 static void    aus_acl(struct t_audit_data *);
113 static void    aus_acct(struct t_audit_data *);
114 static void    aus_chown(struct t_audit_data *);
115 static void    aus_fchown(struct t_audit_data *);
116 static void    aus_lchown(struct t_audit_data *);
117 static void    aus_fchowmat(struct t_audit_data *);
118 static void    aus_chmod(struct t_audit_data *);
119 static void    aus_facl(struct t_audit_data *);
120 static void    aus_fchmod(struct t_audit_data *);
121 static void    aus_fchmodat(struct t_audit_data *);
122 static void    aus_fcntl(struct t_audit_data *);
123 static void    aus_mkdir(struct t_audit_data *);
124 static void    aus_mkdirat(struct t_audit_data *);

```

```

125 static void aus_mknod(struct t_audit_data *);
126 static void aus_mknodat(struct t_audit_data *);
127 static void aus_mount(struct t_audit_data *);
128 static void aus_umount2(struct t_audit_data *);
129 static void aus_msgsys(struct t_audit_data *);
130 static void aus_semsys(struct t_audit_data *);
131 static void aus_close(struct t_audit_data *);
132 static void aus_fstatfs(struct t_audit_data *);
133 static void aus_setgid(struct t_audit_data *);
134 static void aus_setpggrp(struct t_audit_data *);
135 static void aus_setuid(struct t_audit_data *);
136 static void aus_shmsys(struct t_audit_data *);
137 static void aus_doorfs(struct t_audit_data *);
138 static void aus_ioctl(struct t_audit_data *);
139 static void aus_memcntl(struct t_audit_data *);
140 static void aus_mmap(struct t_audit_data *);
141 static void aus_munmap(struct t_audit_data *);
142 static void aus_priocntlsys(struct t_audit_data *);
143 static void aus_setegid(struct t_audit_data *);
144 static void aus_setgroups(struct t_audit_data *);
145 static void aus_seteuid(struct t_audit_data *);
146 static void aus_putmsg(struct t_audit_data *);
147 static void aus_putpmsg(struct t_audit_data *);
148 static void aus_getmsg(struct t_audit_data *);
149 static void aus_getpmsg(struct t_audit_data *);
150 static void aus_auditsys(struct t_audit_data *);
151 static void aus_sysinfo(struct t_audit_data *);
152 static void aus_modctl(struct t_audit_data *);
153 static void aus_kill(struct t_audit_data *);
154 static void aus_setregid(struct t_audit_data *);
155 static void aus_setreuid(struct t_audit_data *);
156 static void aus_labelsys(struct t_audit_data *);

158 static void auf_mknod(struct t_audit_data *, int, rval_t *);
159 static void auf_mknodat(struct t_audit_data *, int, rval_t *);
160 static void auf_msgsys(struct t_audit_data *, int, rval_t *);
161 static void auf_semsys(struct t_audit_data *, int, rval_t *);
162 static void auf_shmsys(struct t_audit_data *, int, rval_t *);
163 static void auf_read(struct t_audit_data *, int, rval_t *);
164 static void auf_write(struct t_audit_data *, int, rval_t *);

166 static void aus_sigqueue(struct t_audit_data *);
167 static void aus_p_online(struct t_audit_data *);
168 static void aus_processor_bind(struct t_audit_data *);
169 static void aus_inst_sync(struct t_audit_data *);
170 static void aus_brandsys(struct t_audit_data *);

172 static void auf_accept(struct t_audit_data *, int, rval_t *);

174 static void auf_bind(struct t_audit_data *, int, rval_t *);
175 static void auf_connect(struct t_audit_data *, int, rval_t *);
176 static void aus_shutdown(struct t_audit_data *);
177 static void auf_setsockopt(struct t_audit_data *, int, rval_t *);
178 static void aus_sockconfig(struct t_audit_data *);
179 static void auf_rcv(struct t_audit_data *, int, rval_t *);
180 static void auf_rcvmsg(struct t_audit_data *, int, rval_t *);
181 static void auf_send(struct t_audit_data *, int, rval_t *);
182 static void auf_sendmsg(struct t_audit_data *, int, rval_t *);
183 static void auf_rcvfrom(struct t_audit_data *, int, rval_t *);
184 static void auf_sendto(struct t_audit_data *, int, rval_t *);
185 static void aus_socket(struct t_audit_data *);
186 /*
187 * This table contains mapping information for converting system call numbers
188 * to audit event IDs. In several cases it is necessary to map a single system
189 * call to several events.
190 */

```

```

192 #define aui_null      NULL      /* NULL initialize function */
193 #define aus_null     NULL      /* NULL start function */
194 #define auf_null     NULL      /* NULL finish function */

196 struct audit_s2e audit_s2e[] =
197 {
198 /*
199  * -----
200  * INITIAL      AUDIT      START      SYSTEM
201  * PROCESSING  EVENT      PROCESSING  CALL
202  * -----
203  *
204  * FINISH      EVENT
205  * PROCESSING  CONTROL
206  * -----
207 aui_null,      AUE_NULL,      aus_null,      /* 0 unused (indirect) */
208                auf_null,      0,
209 aui_null,      AUE_EXIT,      aus_exit,      /* 1 exit */
210                auf_null,      S2E_NPT,
211 aui_null,      AUE_PSECFLAGS, aus_null,      /* 2 psecflags */
212                auf_null,      aus_null,      /* 2 (loadable) was forkall */
213 aui_null,      AUE_READ,      aus_null,      /* 3 read */
214                auf_read,      S2E_PUB,
215 aui_null,      AUE_WRITE,      aus_null,      /* 4 write */
216                auf_write,      0,
217 aui_open,      AUE_OPEN,      aus_open,      /* 5 open */
218                auf_null,      S2E_SP,
219 aui_null,      AUE_CLOSE,      aus_close,     /* 6 close */
220                auf_null,      0,
221 aui_null,      AUE_LINK,      aus_null,      /* 7 linkat */
222                auf_null,      0,
223 aui_null,      AUE_NULL,      aus_null,      /* 8 (loadable) was creat */
224                auf_null,      0,
225 aui_null,      AUE_LINK,      aus_null,      /* 9 link */
226                auf_null,      0,
227 aui_null,      AUE_UNLINK,   aus_null,      /* 10 unlink */
228                auf_null,      0,
229 aui_null,      AUE_SYMLINK,   aus_null,      /* 11 symlinkat */
230                auf_null,      0,
231 aui_null,      AUE_CHDIR,      aus_null,      /* 12 chdir */
232                auf_null,      S2E_SP,
233 aui_null,      AUE_NULL,      aus_null,      /* 13 time */
234                auf_null,      0,
235 aui_null,      AUE_MKNOD,      aus_mknod,     /* 14 mknod */
236                auf_mknod,     S2E_MLD,
237 aui_null,      AUE_CHMOD,      aus_chmod,     /* 15 chmod */
238                auf_null,      0,
239 aui_null,      AUE_CHOWN,      aus_chown,     /* 16 chown */
240                auf_null,      0,
241 aui_null,      AUE_NULL,      aus_null,      /* 17 brk */
242                auf_null,      0,
243 aui_null,      AUE_STAT,      aus_null,      /* 18 stat */
244                auf_null,      S2E_PUB,
245 aui_null,      AUE_NULL,      aus_null,      /* 19 lseek */
246                auf_null,      0,
247 aui_null,      AUE_NULL,      aus_null,      /* 20 getpid */
248                auf_null,      0,
249 aui_null,      AUE_MOUNT,      aus_mount,     /* 21 mount */
250                auf_null,      S2E_MLD,
251 aui_null,      AUE_READLINK,  aus_null,      /* 22 readlinkat */
252                auf_null,      S2E_PUB,
253 aui_null,      AUE_SETUID,      aus_setuid,    /* 23 setuid */
254                auf_null,      0,
255 aui_null,      AUE_NULL,      aus_null,      /* 24 getuid */

```

```

256         auf_null,          0,
257 aui_null,    AUE_STIME,    aus_null,    /* 25 stime */
258         auf_null,          0,
259 aui_null,    AUE_NULL,     aus_null,    /* 26 pcsample */
260         auf_null,          0,
261 aui_null,    AUE_NULL,     aus_null,    /* 27 alarm */
262         auf_null,          0,
263 aui_null,    AUE_NULL,     aus_null,    /* 28 fstat */
264         auf_null,          0,
265 aui_null,    AUE_NULL,     aus_null,    /* 29 pause */
266         auf_null,          0,
267 aui_null,    AUE_NULL,     aus_null,    /* 30 (loadable) was utime */
268         auf_null,          0,
269 aui_null,    AUE_NULL,     aus_null,    /* 31 stty (TIOCSETP-audit?) */
270         auf_null,          0,
271 aui_null,    AUE_NULL,     aus_null,    /* 32 gtty */
272         auf_null,          0,
273 aui_null,    AUE_ACCESS,   aus_null,    /* 33 access */
274         auf_null,          S2E_PUB,
275 aui_null,    AUE_NICE,     aus_null,    /* 34 nice */
276         auf_null,          0,
277 aui_null,    AUE_STATFS,   aus_null,    /* 35 statfs */
278         auf_null,          S2E_PUB,
279 aui_null,    AUE_NULL,     aus_null,    /* 36 sync */
280         auf_null,          0,
281 aui_null,    AUE_KILL,     aus_kill,    /* 37 kill */
282         auf_null,          0,
283 aui_null,    AUE_FSTATFS,  aus_fstatfs, /* 38 fstatfs */
284         auf_null,          S2E_PUB,
285 aui_setpgrp, AUE_SETPGRP,   aus_setpgrp, /* 39 setpgrp */
286         auf_null,          0,
287 aui_null,    AUE_NULL,     aus_null,    /* 40 uucopystr */
288         auf_null,          0,
289 aui_null,    AUE_NULL,     aus_null,    /* 41 (loadable) was dup */
290         auf_null,          0,
291 aui_null,    AUE_PIPE,     aus_null,    /* 42 (loadable) pipe */
292         auf_null,          0,
293 aui_null,    AUE_NULL,     aus_null,    /* 43 times */
294         auf_null,          0,
295 aui_null,    AUE_NULL,     aus_null,    /* 44 profil */
296         auf_null,          0,
297 aui_null,    AUE_ACCESS,   aus_null,    /* 45 faccessat */
298         auf_null,          S2E_PUB,
299 aui_null,    AUE_SETGID,   aus_setgid, /* 46 setgid */
300         auf_null,          0,
301 aui_null,    AUE_NULL,     aus_null,    /* 47 getgid */
302         auf_null,          0,
303 aui_null,    AUE_MKNOD,   aus_mknodat, /* 48 mknodat */
304         auf_mknodat,      S2E_MLD,
305 aui_msgsys,  AUE_MSGSYS,   aus_msgsys, /* 49 (loadable) msgsys */
306         auf_msgsys,      0,
307 #if defined(__x86)
308 aui_null,    AUE_NULL,     aus_null,    /* 50 sysi86 */
309         auf_null,          0,
310 #else
311 aui_null,    AUE_NULL,     aus_null,    /* 50 (loadable) was sys3b */
312         auf_null,          0,
313 #endif /* __x86 */
314 aui_null,    AUE_ACCT,     aus_acct,   /* 51 (loadable) sysacct */
315         auf_null,          0,
316 aui_shmsys,  AUE_SHMSYS,   aus_shmsys, /* 52 (loadable) shmsys */
317         auf_shmsys,      0,
318 aui_semsys,  AUE_SEMSYS,   aus_semsys, /* 53 (loadable) semsys */
319         auf_semsys,      0,
320 aui_null,    AUE_IOCTL,   aus_ioctl,  /* 54 ioctl */
321         auf_null,          0,

```

```

322 aui_null,    AUE_NULL,     aus_null,    /* 55 uadmin */
323         auf_null,          0,
324 aui_fchownat, AUE_NULL,     aus_fchownat, /* 56 fchownat */
325         auf_null,          0,
326 aui_utssys,  AUE_FUSERS,   aus_null,    /* 57 utssys */
327         auf_null,          0,
328 aui_null,    AUE_NULL,     aus_null,    /* 58 fsync */
329         auf_null,          0,
330 aui_execve,  AUE_EXECVE,   aus_null,    /* 59 exece */
331         auf_null,          S2E_MLD,
332 aui_null,    AUE_NULL,     aus_null,    /* 60 umask */
333         auf_null,          0,
334 aui_null,    AUE_CHROOT,   aus_null,    /* 61 chroot */
335         auf_null,          S2E_SP,
336 aui_fcntl,   AUE_FCNTL,   aus_fcntl,   /* 62 fcntl */
337         auf_null,          0,
338 aui_null,    AUE_NULL,     aus_null,    /* 63 ulimit */
339         auf_null,          0,
340 aui_null,    AUE_RENAME,   aus_null,    /* 64 renameat */
341         auf_null,          0,
342 aui_unlinkat, AUE_NULL,     aus_null,    /* 65 unlinkat */
343         auf_null,          0,
344 aui_fstatat, AUE_NULL,     aus_null,    /* 66 fstatat */
345         auf_null,          S2E_PUB,
346 aui_fstatat, AUE_NULL,     aus_null,    /* 67 fstatat64 */
347         auf_null,          S2E_PUB,
348 aui_openat,  AUE_OPEN,     aus_openat, /* 68 openat */
349         auf_null,          S2E_SP,
350 aui_openat,  AUE_OPEN,     aus_openat, /* 69 openat64 */
351         auf_null,          S2E_SP,
352 aui_null,    AUE_NULL,     aus_null,    /* 70 tasksys */
353         auf_null,          0,
354 aui_null,    AUE_NULL,     aus_null,    /* 71 (loadable) acctctl */
355         auf_null,          0,
356 aui_null,    AUE_NULL,     aus_null,    /* 72 (loadable) exacct */
357         auf_null,          0,
358 aui_null,    AUE_NULL,     aus_null,    /* 73 getpagesizes */
359         auf_null,          0,
360 aui_null,    AUE_NULL,     aus_null,    /* 74 rctlsys */
361         auf_null,          0,
362 aui_null,    AUE_NULL,     aus_null,    /* 75 sidsys */
363         auf_null,          0,
364 aui_null,    AUE_NULL,     aus_null,    /* 76 (loadable) was fsat */
365         auf_null,          0,
366 aui_null,    AUE_NULL,     aus_null,    /* 77 syslwp_park */
367         auf_null,          0,
368 aui_null,    AUE_NULL,     aus_null,    /* 78 sendfilev */
369         auf_null,          0,
370 aui_null,    AUE_RMDIR,   aus_null,    /* 79 rmdir */
371         auf_null,          0,
372 aui_null,    AUE_MKDIR,   aus_mkdir,   /* 80 mkdir */
373         auf_null,          0,
374 aui_null,    AUE_NULL,     aus_null,    /* 81 getdents */
375         auf_null,          0,
376 aui_privsys, AUE_NULL,     aus_null,    /* 82 privsys */
377         auf_null,          0,
378 aui_null,    AUE_NULL,     aus_null,    /* 83 ucredsys */
379         auf_null,          0,
380 aui_null,    AUE_NULL,     aus_null,    /* 84 sysfs */
381         auf_null,          0,
382 aui_null,    AUE_GETMSG,   aus_getmsg, /* 85 getmsg */
383         auf_null,          0,
384 aui_null,    AUE_PUTMSG,   aus_putmsg, /* 86 putmsg */
385         auf_null,          0,
386 aui_null,    AUE_NULL,     aus_null,    /* 87 (loadable) was poll */
387         auf_null,          0,

```

```

388 aui_null,      AUE_LSTAT,      aus_null,      /* 88 lstat */
389 aui_null,      auf_null,      S2E_PUB,
390 aui_null,      AUE_SYMLINK,   aus_null,      /* 89 symlink */
391 aui_null,      auf_null,      0,
392 aui_null,      AUE_READLINK, aus_null,      /* 90 readlink */
393 aui_null,      auf_null,      S2E_PUB,
394 aui_null,      AUE_SETGROUPS, aus_setgroups, /* 91 setgroups */
395 aui_null,      auf_null,      0,
396 aui_null,      AUE_NULL,      aus_null,      /* 92 getgroups */
397 aui_null,      auf_null,      0,
398 aui_null,      AUE_FCHMOD,    aus_fchmod,    /* 93 fchmod */
399 aui_null,      auf_null,      0,
400 aui_null,      AUE_FCHOWN,    aus_fchown,    /* 94 fchown */
401 aui_null,      auf_null,      0,
402 aui_null,      AUE_NULL,      aus_null,      /* 95 sigprocmask */
403 aui_null,      auf_null,      0,
404 aui_null,      AUE_NULL,      aus_null,      /* 96 sigsuspend */
405 aui_null,      auf_null,      0,
406 aui_null,      AUE_NULL,      aus_null,      /* 97 sigaltstack */
407 aui_null,      auf_null,      0,
408 aui_null,      AUE_NULL,      aus_null,      /* 98 sigaction */
409 aui_null,      auf_null,      0,
410 aui_null,      AUE_NULL,      aus_null,      /* 99 sigpending */
411 aui_null,      auf_null,      0,
412 aui_null,      AUE_NULL,      aus_null,      /* 100 setcontext */
413 aui_null,      auf_null,      0,
414 aui_fchmodat,  AUE_NULL,      aus_fchmodat, /* 101 fchmodat */
415 aui_null,      auf_null,      0,
416 aui_null,      AUE_MKDIR,     aus_mkdirat,   /* 102 mkdirat */
417 aui_null,      auf_null,      0,
418 aui_null,      AUE_STATVFS,   aus_null,      /* 103 statvfs */
419 aui_null,      auf_null,      S2E_PUB,
420 aui_null,      AUE_NULL,      aus_null,      /* 104 fstatvfs */
421 aui_null,      auf_null,      0,
422 aui_null,      AUE_NULL,      aus_null,      /* 105 getloadavg */
423 aui_null,      auf_null,      0,
424 aui_null,      AUE_NULL,      aus_null,      /* 106 nfssys */
425 aui_null,      auf_null,      0,
426 aui_null,      AUE_NULL,      aus_null,      /* 107 waitsys */
427 aui_null,      auf_null,      0,
428 aui_null,      AUE_NULL,      aus_null,      /* 108 sigsendsys */
429 aui_null,      auf_null,      0,
430 #if defined(__x86)
431 aui_null,      AUE_NULL,      aus_null,      /* 109 hrtsys */
432 aui_null,      auf_null,      0,
433 #else
434 aui_null,      AUE_NULL,      aus_null,      /* 109 (loadable) */
435 aui_null,      auf_null,      0,
436 #endif /* __x86 */
437 aui_null,      AUE_UTIMES,    aus_null,      /* 110 utimesys */
438 aui_null,      auf_null,      0,
439 aui_null,      AUE_NULL,      aus_null,      /* 111 sigresend */
440 aui_null,      auf_null,      0,
441 aui_null,      AUE_PRIOCNLSYS, aus_priocntlsys, /* 112 priocntlsys */
442 aui_null,      auf_null,      0,
443 aui_null,      AUE_PATHCONF, aus_null,      /* 113 pathconf */
444 aui_null,      auf_null,      S2E_PUB,
445 aui_null,      AUE_NULL,      aus_null,      /* 114 mincore */
446 aui_null,      auf_null,      0,
447 aui_null,      AUE_MMMap,     aus_mmap,      /* 115 mmap */
448 aui_null,      auf_null,      0,
449 aui_null,      AUE_NULL,      aus_null,      /* 116 mprotect */
450 aui_null,      auf_null,      0,
451 aui_null,      AUE_MUNMAP,    aus_munmap,    /* 117 munmap */
452 aui_null,      auf_null,      0,
453 aui_null,      AUE_NULL,      aus_null,      /* 118 fpathconf */

```

```

454 aui_null,      auf_null,      0,
455 aui_null,      AUE_VFORK,     aus_null,      /* 119 vfork */
456 aui_null,      auf_null,      0,
457 aui_null,      AUE_FCHDIR,    aus_null,      /* 120 fchdir */
458 aui_null,      auf_null,      0,
459 aui_null,      AUE_READ,      aus_null,      /* 121 readv */
460 aui_null,      auf_read,      S2E_PUB,
461 aui_null,      AUE_WRITE,     aus_null,      /* 122 writev */
462 aui_null,      auf_write,     0,
463 aui_null,      AUE_NULL,      aus_null,      /* 123 (loadable) was xstat */
464 aui_null,      auf_null,      0,
465 aui_null,      AUE_NULL,      aus_null,      /* 124 (loadable) was lxstat */
466 aui_null,      auf_null,      0,
467 aui_null,      AUE_NULL,      aus_null,      /* 125 (loadable) was fxstat */
468 aui_null,      auf_null,      0,
469 aui_null,      AUE_NULL,      aus_null,      /* 126 (loadable) was xmknod */
470 aui_null,      auf_null,      0,
471 aui_null,      AUE_NULL,      aus_null,      /* 127 mmapobj */
472 aui_null,      auf_null,      0,
473 aui_null,      AUE_SETRLIMIT, aus_null,      /* 128 setrlimit */
474 aui_null,      auf_null,      0,
475 aui_null,      AUE_NULL,      aus_null,      /* 129 getrlimit */
476 aui_null,      auf_null,      0,
477 aui_null,      AUE_LCHOWN,    aus_lchown,    /* 130 lchown */
478 aui_null,      auf_null,      0,
479 aui_memcntl,  AUE_MEMCNTL,   aus_memcntl,   /* 131 memcntl */
480 aui_null,      auf_null,      0,
481 aui_null,      AUE_GETPMSG,   aus_getpmsg,   /* 132 getpmsg */
482 aui_null,      auf_null,      0,
483 aui_null,      AUE_PUTPMSG,   aus_putpmsg,   /* 133 putpmsg */
484 aui_null,      auf_null,      0,
485 aui_null,      AUE_RENAME,    aus_null,      /* 134 rename */
486 aui_null,      auf_null,      0,
487 aui_null,      AUE_NULL,      aus_null,      /* 135 uname */
488 aui_null,      auf_null,      0,
489 aui_null,      AUE_SETEGID,   aus_setegid,   /* 136 setegid */
490 aui_null,      auf_null,      0,
491 aui_null,      AUE_NULL,      aus_null,      /* 137 sysconfig */
492 aui_null,      auf_null,      0,
493 aui_null,      AUE_ADJTIME,   aus_null,      /* 138 adjtime */
494 aui_null,      auf_null,      0,
495 aui_sysinfo,  AUE_SYSINFO,   aus_sysinfo,   /* 139 systeminfo */
496 aui_null,      auf_null,      0,
497 aui_null,      AUE_NULL,      aus_null,      /* 140 (loadable) sharefs */
498 aui_null,      auf_null,      0,
499 aui_null,      AUE_SETEUID,   aus_seteuid,   /* 141 seteuid */
500 aui_null,      auf_null,      0,
501 aui_forksys,  AUE_NULL,      aus_null,      /* 142 forksys */
502 aui_null,      auf_null,      0,
503 aui_null,      AUE_NULL,      aus_null,      /* 143 (loadable) was fork1 */
504 aui_null,      auf_null,      0,
505 aui_null,      AUE_NULL,      aus_null,      /* 144 sigwait */
506 aui_null,      auf_null,      0,
507 aui_null,      AUE_NULL,      aus_null,      /* 145 lwp_info */
508 aui_null,      auf_null,      0,
509 aui_null,      AUE_NULL,      aus_null,      /* 146 yield */
510 aui_null,      auf_null,      0,
511 aui_null,      AUE_NULL,      aus_null,      /* 147 (loadable) */
512 aui_null,      auf_null,      0, /* was lwp_sema_wait */
513 aui_null,      auf_null,      0,
514 aui_null,      AUE_NULL,      aus_null,      /* 148 lwp_sema_post */
515 aui_null,      auf_null,      0,
516 aui_null,      AUE_NULL,      aus_null,      /* 149 lwp_sema_trywait */
517 aui_null,      auf_null,      0,
518 aui_null,      AUE_NULL,      aus_null,      /* 150 lwp_detach */
519 aui_null,      auf_null,      0,

```

```

520 aui_null,      AUE_NULL,      aus_null,      /* 151 corectl */
521 aui_null,      auf_null,      0,
522 aui_modctl,   AUE_MODCTL,   aus_modctl,   /* 152 modctl */
523 aui_null,      auf_null,      0,
524 aui_null,      AUE_FCHROOT, aus_null,     /* 153 fchroot */
525 aui_null,      auf_null,      0,
526 aui_null,      AUE_NULL,     aus_null,     /* 154 (loadable) was utimes */
527 aui_null,      auf_null,      0,
528 aui_null,      AUE_NULL,     aus_null,     /* 155 vhangup */
529 aui_null,      auf_null,      0,
530 aui_null,      AUE_NULL,     aus_null,     /* 156 gettimeofday */
531 aui_null,      auf_null,      0,
532 aui_null,      AUE_NULL,     aus_null,     /* 157 getitimer */
533 aui_null,      auf_null,      0,
534 aui_null,      AUE_NULL,     aus_null,     /* 158 setitimer */
535 aui_null,      auf_null,      0,
536 aui_null,      AUE_NULL,     aus_null,     /* 159 lwp_create */
537 aui_null,      auf_null,      0,
538 aui_null,      AUE_NULL,     aus_null,     /* 160 lwp_exit */
539 aui_null,      auf_null,      0,
540 aui_null,      AUE_NULL,     aus_null,     /* 161 lwp_suspend */
541 aui_null,      auf_null,      0,
542 aui_null,      AUE_NULL,     aus_null,     /* 162 lwp_continue */
543 aui_null,      auf_null,      0,
544 aui_null,      AUE_NULL,     aus_null,     /* 163 lwp_kill */
545 aui_null,      auf_null,      0,
546 aui_null,      AUE_NULL,     aus_null,     /* 164 lwp_self */
547 aui_null,      auf_null,      0,
548 aui_null,      AUE_NULL,     aus_null,     /* 165 lwp_sigmask */
549 aui_null,      auf_null,      0,
550 aui_null,      AUE_NULL,     aus_null,     /* 166 lwp_private */
551 aui_null,      auf_null,      0,
552 aui_null,      AUE_NULL,     aus_null,     /* 167 lwp_wait */
553 aui_null,      auf_null,      0,
554 aui_null,      AUE_NULL,     aus_null,     /* 168 lwp_mutex_wakeup */
555 aui_null,      auf_null,      0,
556 aui_null,      AUE_NULL,     aus_null,     /* 169 (loadable) */
557 aui_null,      auf_null,      0, /* 169 (loadable) */
558 aui_null,      auf_null,      0, /* was lwp_mutex_lock */
559 aui_null,      AUE_NULL,     aus_null,     /* 170 lwp_cond_wait */
560 aui_null,      auf_null,      0,
561 aui_null,      AUE_NULL,     aus_null,     /* 171 lwp_cond_signal */
562 aui_null,      auf_null,      0,
563 aui_null,      AUE_NULL,     aus_null,     /* 172 lwp_cond_broadcast */
564 aui_null,      auf_null,      0,
565 aui_null,      AUE_READ,     aus_null,     /* 173 pread */
566 aui_null,      auf_read,     S2E_PUB,     /* 174 pwrite */
567 aui_null,      AUE_WRITE,    aus_null,     /* 175 llseek */
568 aui_null,      auf_write,    0,
569 aui_null,      AUE_NULL,     aus_null,     /* 176 (loadable) inst_sync */
570 aui_null,      auf_null,      0,
571 aui_null,      AUE_INST_SYNC, aus_inst_sync, /* 176 (loadable) inst_sync */
572 aui_null,      auf_null,      0,
573 aui_null,      AUE_BRANDSYS, aus_brandsys, /* 177 brandsys */
574 aui_null,      auf_null,      0,
575 aui_null,      AUE_NULL,     aus_null,     /* 178 (loadable) kaio */
576 aui_null,      auf_null,      0,
577 aui_null,      AUE_NULL,     aus_null,     /* 179 (loadable) cpc */
578 aui_null,      auf_null,      0,
579 aui_null,      AUE_NULL,     aus_null,     /* 180 lgrpsys */
580 aui_null,      auf_null,      0,
581 aui_null,      AUE_NULL,     aus_null,     /* 181 rusagesys */
582 aui_null,      auf_null,      0,
583 aui_portfs,   AUE_PORTFS,   aus_null,     /* 182 (loadable) portfs */
584 aui_null,      auf_null,     S2E_MLD,
585 aui_null,      AUE_NULL,     aus_null,     /* 183 pollsys */

```

```

586 aui_null,      auf_null,      0,
587 aui_labelsys, AUE_NULL,     aus_labelsys, /* 184 labelsys */
588 aui_labelsys, auf_null,      0,
589 aui_acl,       AUE_ACLSET,   aus_acl,      /* 185 acl */
590 aui_labelsys, auf_null,      0,
591 aui_auditsys, AUE_AUDITSYS, aus_auditsys, /* 186 auditsys */
592 aui_labelsys, auf_null,      0,
593 aui_processor_bind, AUE_PROCESSOR_BIND, aus_processor_bind, /* 187 processor_bind */
594 aui_labelsys, auf_null,      0,
595 aui_processor_info, AUE_NULL,     aus_null,     /* 188 processor_info */
596 aui_labelsys, auf_null,      0,
597 aui_p_online,  AUE_P_ONLINE, aus_p_online, /* 189 p_online */
598 aui_labelsys, auf_null,      0,
599 aui_sigqueue,  AUE_NULL,     aus_sigqueue, /* 190 sigqueue */
600 aui_labelsys, auf_null,      0,
601 aui_clock_gettime, AUE_NULL,     aus_null,     /* 191 clock_gettime */
602 aui_labelsys, auf_null,      0,
603 aui_clock_settime, AUE_CLOCK_SETTIME, aus_null,     /* 192 clock_settime */
604 aui_labelsys, auf_null,      0,
605 aui_clock_getres, AUE_NULL,     aus_null,     /* 193 clock_getres */
606 aui_labelsys, auf_null,      0,
607 aui_timer_create, AUE_NULL,     aus_null,     /* 194 timer_create */
608 aui_labelsys, auf_null,      0,
609 aui_timer_delete, AUE_NULL,     aus_null,     /* 195 timer_delete */
610 aui_labelsys, auf_null,      0,
611 aui_timer_settime, AUE_NULL,     aus_null,     /* 196 timer_settime */
612 aui_labelsys, auf_null,      0,
613 aui_timer_gettime, AUE_NULL,     aus_null,     /* 197 timer_gettime */
614 aui_labelsys, auf_null,      0,
615 aui_timer_getoverrun, AUE_NULL,     aus_null,     /* 198 timer_getoverrun */
616 aui_labelsys, auf_null,      0,
617 aui_nanosleep, AUE_NULL,     aus_null,     /* 199 nanosleep */
618 aui_labelsys, auf_null,      0,
619 aui_acl,       AUE_FACLSET,   aus_fac1,     /* 200 fac1 */
620 aui_labelsys, auf_null,      0,
621 aui_doorfs,    AUE_DOORFS,   aus_doorfs,   /* 201 (loadable) doorfs */
622 aui_labelsys, auf_null,      0,
623 aui_setreuid,  AUE_SETREUID, aus_setreuid, /* 202 setreuid */
624 aui_labelsys, auf_null,      0,
625 aui_setregid,  AUE_SETREGID, aus_setregid, /* 203 setregid */
626 aui_labelsys, auf_null,      0,
627 aui_install_utrap, AUE_NULL,     aus_null,     /* 204 install_utrap */
628 aui_labelsys, auf_null,      0,
629 aui_notify,    AUE_NULL,     aus_null,     /* 205 notify */
630 aui_labelsys, auf_null,      0,
631 aui_schedctl,  AUE_NULL,     aus_null,     /* 206 schedctl */
632 aui_labelsys, auf_null,      0,
633 aui_pset,      AUE_NULL,     aus_null,     /* 207 (loadable) pset */
634 aui_labelsys, auf_null,      0,
635 aui_sparc_utrap_install, AUE_NULL,     aus_null,     /* 208 sparc_utrap_install */
636 aui_labelsys, auf_null,      0,
637 aui_resolvepath, AUE_NULL,     aus_null,     /* 209 resolvepath */
638 aui_labelsys, auf_null,      0,
639 aui_lwp_mutex_timedlock, AUE_NULL,     aus_null,     /* 210 lwp_mutex_timedlock */
640 aui_labelsys, auf_null,      0,
641 aui_lwp_sema_timedwait, AUE_NULL,     aus_null,     /* 211 lwp_sema_timedwait */
642 aui_labelsys, auf_null,      0,
643 aui_lwp_rwlock_sys, AUE_NULL,     aus_null,     /* 212 lwp_rwlock_sys */
644 aui_labelsys, auf_null,      0,
645 aui_getdents64, AUE_NULL,     aus_null,     /* 213 getdents64 */
646 aui_labelsys, auf_null,      0,
647 aui_mmap64,    AUE_MMAP,     aus_mmap,     /* 214 mmap64 */
648 aui_labelsys, auf_null,      0,
649 aui_stat64,    AUE_STAT,     aus_null,     /* 215 stat64 */
650 aui_labelsys, auf_null,     S2E_PUB,
651 aui_labelsys, AUE_LSTAT,     aus_null,     /* 216 lstat64 */

```



```

652         auf_null,      S2E_PUB,
653 aui_null,    AUE_NULL,  aus_null,    /* 217 fstat64 */
654         auf_null,      0,
655 aui_null,    AUE_STATVFS, aus_null,    /* 218 statvfs64 */
656         auf_null,      S2E_PUB,
657 aui_null,    AUE_NULL,  aus_null,    /* 219 fstatvfs64 */
658         auf_null,      0,
659 aui_null,    AUE_SETRLIMIT, aus_null,    /* 220 setrlimit64 */
660         auf_null,      0,
661 aui_null,    AUE_NULL,  aus_null,    /* 221 getrlimit64 */
662         auf_null,      0,
663 aui_null,    AUE_READ,   aus_null,    /* 222 pread64 */
664         auf_read,      S2E_PUB,
665 aui_null,    AUE_WRITE,  aus_null,    /* 223 pwrite64 */
666         auf_write,     0,
667 aui_null,    AUE_NULL,  aus_null,    /* 224 (loadable) was creat64 */
668         auf_null,      0,
669 aui_open,   AUE_OPEN,   aus_open,    /* 225 open64 */
670         auf_null,      S2E_SP,
671 aui_null,    AUE_NULL,  aus_null,    /* 226 (loadable) rpcsys */
672         auf_null,      0,
673 aui_null,    AUE_NULL,  aus_null,    /* 227 zone */
674         auf_null,      0,
675 aui_null,    AUE_NULL,  aus_null,    /* 228 (loadable) autofssys */
676         auf_null,      0,
677 aui_null,    AUE_NULL,  aus_null,    /* 229 getcwd */
678         auf_null,      0,
679 aui_null,    AUE_SOCKET, aus_socket,  /* 230 so_socket */
680         auf_null,      0,
681 aui_null,    AUE_NULL,  aus_null,    /* 231 so_socketpair */
682         auf_null,      0,
683 aui_null,    AUE_BIND,   aus_null,    /* 232 bind */
684         auf_bind,      0,
685 aui_null,    AUE_NULL,  aus_null,    /* 233 listen */
686         auf_null,      0,
687 aui_null,    AUE_ACCEPT, aus_null,    /* 234 accept */
688         auf_accept,    0,
689 aui_null,    AUE_CONNECT, aus_null,    /* 235 connect */
690         auf_connect,   0,
691 aui_null,    AUE_SHUTDOWN, aus_shutdown, /* 236 shutdown */
692         auf_null,      0,
693 aui_null,    AUE_READ,   aus_null,    /* 237 recv */
694         auf_recv,      0,
695 aui_null,    AUE_RECVFROM, aus_null,    /* 238 recvfrom */
696         auf_recvfrom,  0,
697 aui_null,    AUE_RECVMSG, aus_null,    /* 239 recvmsg */
698         auf_recvmsg,   0,
699 aui_null,    AUE_WRITE,  aus_null,    /* 240 send */
700         auf_send,      0,
701 aui_null,    AUE_SENDMSG, aus_null,    /* 241 sendmsg */
702         auf_sendmsg,   0,
703 aui_null,    AUE_SENDTO, aus_null,    /* 242 sendto */
704         auf_sendto,    0,
705 aui_null,    AUE_NULL,  aus_null,    /* 243 getpeername */
706         auf_null,      0,
707 aui_null,    AUE_NULL,  aus_null,    /* 244 getsockname */
708         auf_null,      0,
709 aui_null,    AUE_NULL,  aus_null,    /* 245 getsockopt */
710         auf_null,      0,
711 aui_null,    AUE_SETSOCKOPT, aus_null,    /* 246 setsockopt */
712         auf_setsockopt, 0,
713 aui_null,    AUE_SOCKCONFIG, aus_sockconfig, /* 247 sockconfig */
714         auf_null,      0,
715 aui_null,    AUE_NULL,  aus_null,    /* 248 ntp_gettime */
716         auf_null,      0,
717 aui_null,    AUE_NTP_ADJTIME, aus_null,    /* 249 ntp_adjtime */

```

```

718         auf_null,      0,
719 aui_null,    AUE_NULL,  aus_null,    /* 250 lwp_mutex_unlock */
720         auf_null,      0,
721 aui_null,    AUE_NULL,  aus_null,    /* 251 lwp_mutex_trylock */
722         auf_null,      0,
723 aui_null,    AUE_NULL,  aus_null,    /* 252 lwp_mutex_register */
724         auf_null,      0,
725 aui_null,    AUE_NULL,  aus_null,    /* 253 cladm */
726         auf_null,      0,
727 aui_null,    AUE_NULL,  aus_null,    /* 254 uucopy */
728         auf_null,      0,
729 aui_null,    AUE_UMOUNT2, aus_umount2, /* 255 umount2 */
730         auf_null,      0
731 };
_____ unchanged_portion_omitted _____

751 #endif /* ! codereview */
752 /* acct start function */
753 /*ARGSUSED*/
754 static void
755 aus_acct(struct t_audit_data *tad)
756 {
757     klpw_t *clwp = ttolwp(curthread);
758     uintptr_t fname;

760     struct a {
761         long    fname;          /* char ** */
762     } *uap = (struct a *)clwp->lwp_ap;

764     fname = (uintptr_t)uap->fname;

766     if (fname == 0)
767         au_uwrite(au_to_arg32(1, "accounting off", (uint32_t)0));
768 }

770 /* chown start function */
771 /*ARGSUSED*/
772 static void
773 aus_chown(struct t_audit_data *tad)
774 {
775     klpw_t *clwp = ttolwp(curthread);
776     uint32_t uid, gid;

778     struct a {
779         long    fname;          /* char ** */
780         long    uid;
781         long    gid;
782     } *uap = (struct a *)clwp->lwp_ap;

784     uid = (uint32_t)uap->uid;
785     gid = (uint32_t)uap->gid;

787     au_uwrite(au_to_arg32(2, "new file uid", uid));
788     au_uwrite(au_to_arg32(3, "new file gid", gid));
789 }

791 /* fchown start function */
792 /*ARGSUSED*/
793 static void
794 aus_fchown(struct t_audit_data *tad)
795 {
796     klpw_t *clwp = ttolwp(curthread);
797     uint32_t uid, gid, fd;
798     struct file *fp;
799     struct vnode *vp;

```

```

800     struct f_audit_data *fad;

802     struct a {
803         long fd;
804         long uid;
805         long gid;
806     } *uap = (struct a *)clwp->lwp_ap;

808     fd = (uint32_t)uap->fd;
809     uid = (uint32_t)uap->uid;
810     gid = (uint32_t)uap->gid;

812     au_uwrite(au_to_arg32(2, "new file uid", uid));
813     au_uwrite(au_to_arg32(3, "new file gid", gid));

815     /*
816      * convert file pointer to file descriptor
817      * Note: fd ref count incremented here.
818      */
819     if ((fp = getf(fd)) == NULL)
820         return;

822     /* get path from file struct here */
823     fad = F2A(fp);
824     if (fad->fad_aupath != NULL) {
825         au_uwrite(au_to_path(fad->fad_aupath));
826     } else {
827         au_uwrite(au_to_arg32(1, "no path: fd", fd));
828     }

830     vp = fp->f_vnode;
831     audit_attributes(vp);

833     /* decrement file descriptor reference count */
834     releasef(fd);
835 }

837 /*ARGSUSED*/
838 static void
839 aus_lchown(struct t_audit_data *tad)
840 {
841     klpw_t *clwp = ttolwp(curthread);
842     uint32_t uid, gid;

845     struct a {
846         long     fname;          /* char * */
847         long     uid;
848         long     gid;
849     } *uap = (struct a *)clwp->lwp_ap;

851     uid = (uint32_t)uap->uid;
852     gid = (uint32_t)uap->gid;

854     au_uwrite(au_to_arg32(2, "new file uid", uid));
855     au_uwrite(au_to_arg32(3, "new file gid", gid));
856 }

858 static au_event_t
859 aui_fchownat(au_event_t e)
860 {
861     klpw_t *clwp = ttolwp(curthread);

863     struct a {
864         long     fd;
865         long     fname;          /* char * */

```

```

866         long     uid;
867         long     gid;
868         long     flags;
869     } *uap = (struct a *)clwp->lwp_ap;

871     if (uap->fname == NULL)
872         e = AUE_FCHOWN;
873     else if (uap->flags & AT_SYMLINK_NOFOLLOW)
874         e = AUE_LCHOWN;
875     else
876         e = AUE_CHOWN;

878     return (e);
879 }

881 /*ARGSUSED*/
882 static void
883 aus_fchownat(struct t_audit_data *tad)
884 {
885     klpw_t *clwp = ttolwp(curthread);
886     uint32_t uid, gid;

888     struct a {
889         long     fd;
890         long     fname;          /* char * */
891         long     uid;
892         long     gid;
893         long     flags;
894     } *uap = (struct a *)clwp->lwp_ap;

896     uid = (uint32_t)uap->uid;
897     gid = (uint32_t)uap->gid;

899     au_uwrite(au_to_arg32(3, "new file uid", uid));
900     au_uwrite(au_to_arg32(4, "new file gid", gid));
901 }

903 /*ARGSUSED*/
904 static void
905 aus_chmod(struct t_audit_data *tad)
906 {
907     klpw_t *clwp = ttolwp(curthread);
908     uint32_t fmode;

910     struct a {
911         long     fname;          /* char * */
912         long     fmode;
913     } *uap = (struct a *)clwp->lwp_ap;

915     fmode = (uint32_t)uap->fmode;

917     au_uwrite(au_to_arg32(2, "new file mode", fmode&0777));
918 }

920 /*ARGSUSED*/
921 static void
922 aus_fchmod(struct t_audit_data *tad)
923 {
924     klpw_t *clwp = ttolwp(curthread);
925     uint32_t fmode, fd;
926     struct file *fp;
927     struct vnode *vp;
928     struct f_audit_data *fad;

930     struct a {
931         long     fd;

```

```

932     long    fmode;
933     } *uap = (struct a *)clwp->lwp_ap;

935     fd = (uint32_t)uap->fd;
936     fmode = (uint32_t)uap->fmode;

938     au_uwrite(au_to_arg32(2, "new file mode", fmode&07777));

940     /*
941     * convert file pointer to file descriptor
942     * Note: fd ref count incremented here.
943     */
944     if ((fp = getf(fd)) == NULL)
945         return;

947     /* get path from file struct here */
948     fad = F2A(fp);
949     if (fad->fad_aupath != NULL) {
950         au_uwrite(au_to_path(fad->fad_aupath));
951     } else {
952         au_uwrite(au_to_arg32(1, "no path: fd", fd));
953     }

955     vp = fp->f_vnode;
956     audit_attributes(vp);

958     /* decrement file descriptor reference count */
959     releasef(fd);
960 }

962 static au_event_t
963 aui_fchmodat(au_event_t e)
964 {
965     klpw_t *clwp = ttolwp(curthread);

967     struct a {
968         long    fd;
969         long    fname;          /* char * */
970         long    fmode;
971         long    flag;
972     } *uap = (struct a *)clwp->lwp_ap;

974     if (uap->fname == NULL)
975         e = AUE_FCHMOD;
976     else
977         e = AUE_CHMOD;

979     return (e);
980 }

982 /*ARGSUSED*/
983 static void
984 aus_fchmodat(struct t_audit_data *tad)
985 {
986     klpw_t *clwp = ttolwp(curthread);
987     uint32_t fmode;
988     uint32_t fd;
989     struct file *fp;
990     struct vnode *vp;
991     struct f_audit_data *fad;

993     struct a {
994         long    fd;
995         long    fname;          /* char * */
996         long    fmode;
997         long    flag;

```

```

998     } *uap = (struct a *)clwp->lwp_ap;

1000     fd = (uint32_t)uap->fd;
1001     fmode = (uint32_t)uap->fmode;

1003     au_uwrite(au_to_arg32(2, "new file mode", fmode&07777));

1005     if (fd == AT_FDCWD || uap->fname != NULL)      /* same as chmod() */
1006         return;

1008     /*
1009     * convert file pointer to file descriptor
1010     * Note: fd ref count incremented here.
1011     */
1012     if ((fp = getf(fd)) == NULL)
1013         return;

1015     /* get path from file struct here */
1016     fad = F2A(fp);
1017     if (fad->fad_aupath != NULL) {
1018         au_uwrite(au_to_path(fad->fad_aupath));
1019     } else {
1020         au_uwrite(au_to_arg32(1, "no path: fd", fd));
1021     }

1023     vp = fp->f_vnode;
1024     audit_attributes(vp);

1026     /* decrement file descriptor reference count */
1027     releasef(fd);
1028 }

1030 /*
1031 * convert open mode to appropriate open event
1032 */
1033 au_event_t
1034 open_event(uint_t fm)
1035 {
1036     au_event_t e;

1038     switch (fm & (O_ACCMODE | O_CREAT | O_TRUNC)) {
1039     case O_RDONLY:
1040         e = AUE_OPEN_R;
1041         break;
1042     case O_RDONLY | O_CREAT:
1043         e = AUE_OPEN_RC;
1044         break;
1045     case O_RDONLY | O_TRUNC:
1046         e = AUE_OPEN_RT;
1047         break;
1048     case O_RDONLY | O_TRUNC | O_CREAT:
1049         e = AUE_OPEN_RTC;
1050         break;
1051     case O_WRONLY:
1052         e = AUE_OPEN_W;
1053         break;
1054     case O_WRONLY | O_CREAT:
1055         e = AUE_OPEN_WC;
1056         break;
1057     case O_WRONLY | O_TRUNC:
1058         e = AUE_OPEN_WT;
1059         break;
1060     case O_WRONLY | O_TRUNC | O_CREAT:
1061         e = AUE_OPEN_WTC;
1062         break;
1063     case O_RDWR:

```

```

1064         e = AUE_OPEN_RW;
1065         break;
1066     case O_RDWR | O_CREAT:
1067         e = AUE_OPEN_RWC;
1068         break;
1069     case O_RDWR | O_TRUNC:
1070         e = AUE_OPEN_RWT;
1071         break;
1072     case O_RDWR | O_TRUNC | O_CREAT:
1073         e = AUE_OPEN_RWTC;
1074         break;
1075     case O_SEARCH:
1076         e = AUE_OPEN_S;
1077         break;
1078     case O_EXEC:
1079         e = AUE_OPEN_E;
1080         break;
1081     default:
1082         e = AUE_NULL;
1083         break;
1084     }
1086     return (e);
1087 }

1089 /* ARGSUSED */
1090 static au_event_t
1091 aui_open(au_event_t e)
1092 {
1093     klpw_t *clwp = ttolwp(curthread);
1094     uint_t fm;

1096     struct a {
1097         long    fnamep;        /* char * */
1098         long    fmode;
1099         long    cmode;
1100     } *uap = (struct a *)clwp->lwp_ap;

1102     fm = (uint_t)uap->fmode;

1104     return (open_event(fm));
1105 }

1107 static void
1108 aus_open(struct t_audit_data *tad)
1109 {
1110     klpw_t *clwp = ttolwp(curthread);
1111     uint_t fm;

1113     struct a {
1114         long    fnamep;        /* char * */
1115         long    fmode;
1116         long    cmode;
1117     } *uap = (struct a *)clwp->lwp_ap;

1119     fm = (uint_t)uap->fmode;

1121     /* If no write, create, or trunc modes, mark as a public op */
1122     if ((fm & (O_RDONLY|O_WRONLY|O_RDWR|O_CREAT|O_TRUNC)) == O_RDONLY)
1123         tad->tad_ctrl |= TAD_PUBLIC_EV;
1124 }

1126 /* ARGSUSED */
1127 static au_event_t
1128 aui_openat(au_event_t e)
1129 {

```

```

1130     t_audit_data_t *tad = T2A(curthread);
1131     klpw_t *clwp = ttolwp(curthread);
1132     uint_t fm;

1134     struct a {
1135         long    filedes;
1136         long    fnamep;        /* char * */
1137         long    fmode;
1138         long    cmode;
1139     } *uap = (struct a *)clwp->lwp_ap;

1141     fm = (uint_t)uap->fmode;

1143     /*
1144     * __openatrdirat() does an extra pathname lookup in order to
1145     * enter the extended system attribute namespace of the referenced
1146     * extended attribute filename.
1147     */
1148     if (fm & FXATTRDIROPEN)
1149         tad->tad_ctrl |= TAD_MLD;

1151     return (open_event(fm));
1152 }

1154 static void
1155 aus_openat(struct t_audit_data *tad)
1156 {
1157     klpw_t *clwp = ttolwp(curthread);
1158     uint_t fm;

1160     struct a {
1161         long    filedes;
1162         long    fnamep;        /* char * */
1163         long    fmode;
1164         long    cmode;
1165     } *uap = (struct a *)clwp->lwp_ap;

1167     fm = (uint_t)uap->fmode;

1169     /* If no write, create, or trunc modes, mark as a public op */
1170     if ((fm & (O_RDONLY|O_WRONLY|O_RDWR|O_CREAT|O_TRUNC)) == O_RDONLY)
1171         tad->tad_ctrl |= TAD_PUBLIC_EV;
1172 }

1174 static au_event_t
1175 aui_unlinkat(au_event_t e)
1176 {
1177     klpw_t *clwp = ttolwp(curthread);

1179     struct a {
1180         long    filedes;
1181         long    fnamep;        /* char * */
1182         long    flags;
1183     } *uap = (struct a *)clwp->lwp_ap;

1185     if (uap->flags & AT_REMOVEDIR)
1186         e = AUE_RMDIR;
1187     else
1188         e = AUE_UNLINK;

1190     return (e);
1191 }

1193 static au_event_t
1194 aui_fstatat(au_event_t e)
1195 {

```

```

1196     klwp_t *clwp = ttolwp(curthread);
1198     struct a {
1199         long   filedes;
1200         long   fnamep;      /* char ** */
1201         long   statb;
1202         long   flags;
1203     } *uap = (struct a *)clwp->lwp_ap;
1205     if (uap->fnamep == NULL)
1206         e = AUE_FSTAT;
1207     else if (uap->flags & AT_SYMLINK_NOFOLLOW)
1208         e = AUE_LSTAT;
1209     else
1210         e = AUE_STAT;
1212     return (e);
1213 }
1215 /* msgsys */
1216 static au_event_t
1217 aui_msgsys(au_event_t e)
1218 {
1219     klwp_t *clwp = ttolwp(curthread);
1220     uint_t fm;
1222     struct a {
1223         long   id;          /* function code id */
1224         long   ap;          /* arg pointer for recvmmsg */
1225     } *uap = (struct a *)clwp->lwp_ap;
1227     struct b {
1228         long   msgid;
1229         long   cmd;
1230         long   buf;        /* struct msqid_ds ** */
1231     } *uapl = (struct b *)&clwp->lwp_ap[1];
1233     fm = (uint_t)uap->id;
1235     switch (fm) {
1236     case 0:          /* msgget */
1237         e = AUE_MSGGET;
1238         break;
1239     case 1:          /* msgctl */
1240         switch ((uint_t)uapl->cmd) {
1241         case IPC_RMID:
1242             e = AUE_MSGCTL_RMID;
1243             break;
1244         case IPC_SET:
1245             e = AUE_MSGCTL_SET;
1246             break;
1247         case IPC_STAT:
1248             e = AUE_MSGCTL_STAT;
1249             break;
1250         default:
1251             e = AUE_MSGCTL;
1252             break;
1253         }
1254         break;
1255     case 2:          /* msgrcv */
1256         e = AUE_MSGRCV;
1257         break;
1258     case 3:          /* msgsnd */
1259         e = AUE_MSGSND;
1260         break;
1261     default:         /* illegal system call */

```

```

1262         e = AUE_NULL;
1263         break;
1264     }
1266     return (e);
1267 }
1270 /* shmsys */
1271 static au_event_t
1272 aui_shmsys(au_event_t e)
1273 {
1274     klwp_t *clwp = ttolwp(curthread);
1275     int fm;
1277     struct a {
1278         long   id;          /* function code id */
1279     } *uap = (struct a *)clwp->lwp_ap;
1281     struct b {
1282         long   shmid;
1283         long   cmd;
1284         long   arg;        /* struct shmid_ds ** */
1285     } *uapl = (struct b *)&clwp->lwp_ap[1];
1286     fm = (uint_t)uap->id;
1288     switch (fm) {
1289     case 0:          /* shmatt */
1290         e = AUE_SHMAT;
1291         break;
1292     case 1:          /* shmctl */
1293         switch ((uint_t)uapl->cmd) {
1294         case IPC_RMID:
1295             e = AUE_SHMCTL_RMID;
1296             break;
1297         case IPC_SET:
1298             e = AUE_SHMCTL_SET;
1299             break;
1300         case IPC_STAT:
1301             e = AUE_SHMCTL_STAT;
1302             break;
1303         default:
1304             e = AUE_SHMCTL;
1305             break;
1306         }
1307         break;
1308     case 2:          /* shmdt */
1309         e = AUE_SHMDT;
1310         break;
1311     case 3:          /* shmget */
1312         e = AUE_SHMGET;
1313         break;
1314     default:         /* illegal system call */
1315         e = AUE_NULL;
1316         break;
1317     }
1319     return (e);
1320 }
1323 /* semsys */
1324 static au_event_t
1325 aui_semsys(au_event_t e)
1326 {
1327     klwp_t *clwp = ttolwp(curthread);

```

```

1328     uint_t fm;
1330     struct a {           /* semsys */
1331         long    id;
1332     } *uap = (struct a *)clwp->lwp_ap;
1334     struct b {           /* ctrl */
1335         long    semid;
1336         long    semnum;
1337         long    cmd;
1338         long    arg;
1339     } *uapl = (struct b *)&clwp->lwp_ap[1];
1341     fm = (uint_t)uap->id;
1343     switch (fm) {
1344     case 0:               /* semctl */
1345         switch ((uint_t)uapl->cmd) {
1346         case IPC_RMID:
1347             e = AUE_SEMCTL_RMID;
1348             break;
1349         case IPC_SET:
1350             e = AUE_SEMCTL_SET;
1351             break;
1352         case IPC_STAT:
1353             e = AUE_SEMCTL_STAT;
1354             break;
1355         case GETNCNT:
1356             e = AUE_SEMCTL_GETNCNT;
1357             break;
1358         case GETPID:
1359             e = AUE_SEMCTL_GETPID;
1360             break;
1361         case GETVAL:
1362             e = AUE_SEMCTL_GETVAL;
1363             break;
1364         case GETALL:
1365             e = AUE_SEMCTL_GETALL;
1366             break;
1367         case GETZCNT:
1368             e = AUE_SEMCTL_GETZCNT;
1369             break;
1370         case SETVAL:
1371             e = AUE_SEMCTL_SETVAL;
1372             break;
1373         case SETALL:
1374             e = AUE_SEMCTL_SETALL;
1375             break;
1376         default:
1377             e = AUE_SEMCTL;
1378             break;
1379         }
1380     case 1:               /* semget */
1381         e = AUE_SEMGET;
1382         break;
1383     case 2:               /* semop */
1384         e = AUE_SEMOP;
1385         break;
1386     default:              /* illegal system call */
1387         e = AUE_NULL;
1388         break;
1389     }
1390
1392     return (e);
1393 }

```

```

1395 /* utssys - uname(2), ustat(2), fusers(2) */
1396 static au_event_t
1397 aui_utssys(au_event_t e)
1398 {
1399     klwp_t *clwp = ttolwp(curthread);
1400     uint_t type;
1402     struct a {
1403         union {
1404             long    cbuf;           /* char * */
1405             long    ubuf;           /* struct stat * */
1406         } ub;
1407         union {
1408             long    mv;           /* for USTAT */
1409             long    flags;        /* for FUSERS */
1410         } un;
1411         long    type;
1412         long    outbp;           /* char * for FUSERS */
1413     } *uap = (struct a *)clwp->lwp_ap;
1415     type = (uint_t)uap->type;
1417     if (type == UTS_FUSERS)
1418         return (e);
1419     else
1420         return ((au_event_t)AUE_NULL);
1421 }
1423 static au_event_t
1424 aui_fcntl(au_event_t e)
1425 {
1426     klwp_t *clwp = ttolwp(curthread);
1427     uint_t cmd;
1429     struct a {
1430         long    fdes;
1431         long    cmd;
1432         long    arg;
1433     } *uap = (struct a *)clwp->lwp_ap;
1435     cmd = (uint_t)uap->cmd;
1437     switch (cmd) {
1438     case F_GETLK:
1439     case F_SETLK:
1440     case F_SETLKW:
1441         break;
1442     case F_SETFL:
1443     case F_GETFL:
1444     case F_GETFD:
1445         break;
1446     default:
1447         e = (au_event_t)AUE_NULL;
1448         break;
1449     }
1450     return ((au_event_t)e);
1451 }
1453 /* null function for now */
1454 static au_event_t
1455 aui_execve(au_event_t e)
1456 {
1457     return (e);
1458 }

```

```

1460 /*ARGSUSED*/
1461 static void
1462 aus_fcntl(struct t_audit_data *tad)
1463 {
1464     klpw_t *clwp = ttolwp(curthread);
1465     uint32_t cmd, fd, flags;
1466     struct file *fp;
1467     struct vnode *vp;
1468     struct f_audit_data *fad;
1469
1470     struct a {
1471         long    fd;
1472         long    cmd;
1473         long    arg;
1474     } *uap = (struct a *)clwp->lwp_ap;
1475
1476     cmd = (uint32_t)uap->cmd;
1477     fd = (uint32_t)uap->fd;
1478     flags = (uint32_t)uap->arg;
1479
1480     au_uwrite(au_to_arg32(2, "cmd", cmd));
1481
1482     if (cmd == F_SETFL)
1483         au_uwrite(au_to_arg32(3, "flags", flags));
1484
1485     /*
1486      * convert file pointer to file descriptor
1487      * Note: fd ref count incremented here.
1488      */
1489     if ((fp = getf(fd)) == NULL)
1490         return;
1491
1492     /* get path from file struct here */
1493     fad = F2A(fp);
1494     if (fad->fad_aupath != NULL) {
1495         au_uwrite(au_to_path(fad->fad_aupath));
1496     } else {
1497         au_uwrite(au_to_arg32(1, "no path: fd", fd));
1498     }
1499
1500     vp = fp->f_vnode;
1501     audit_attributes(vp);
1502
1503     /* decrement file descriptor reference count */
1504     releasef(fd);
1505 }
1506
1507 /*ARGSUSED*/
1508 static void
1509 aus_kill(struct t_audit_data *tad)
1510 {
1511     klpw_t *clwp = ttolwp(curthread);
1512     struct proc *p;
1513     uint32_t signo;
1514     uid_t uid, ruid;
1515     gid_t gid, rgid;
1516     pid_t pid;
1517     const auditinfo_addr_t *ainfo;
1518     cred_t *cr;
1519
1520     struct a {
1521         long    pid;
1522         long    signo;
1523     } *uap = (struct a *)clwp->lwp_ap;
1524
1525     pid = (pid_t)uap->pid;

```

```

1526     signo = (uint32_t)uap->signo;
1527
1528     au_uwrite(au_to_arg32(2, "signal", signo));
1529     if (pid > 0) {
1530         mutex_enter(&pidlock);
1531         if ((p = prfind(pid)) == (struct proc *)0) ||
1532             (p->p_stat == SIDL) {
1533             mutex_exit(&pidlock);
1534             au_uwrite(au_to_arg32(1, "process", (uint32_t)pid));
1535             return;
1536         }
1537         mutex_enter(&p->p_lock); /* so process doesn't go away */
1538         mutex_exit(&pidlock);
1539
1540         mutex_enter(&p->p_crlock);
1541         crhold(cr = p->p_cred);
1542         mutex_exit(&p->p_crlock);
1543         mutex_exit(&p->p_lock);
1544
1545         ainfo = crgetauinfo(cr);
1546         if (ainfo == NULL) {
1547             crfree(cr);
1548             au_uwrite(au_to_arg32(1, "process", (uint32_t)pid));
1549             return;
1550         }
1551
1552         uid = crgetuid(cr);
1553         gid = crgetgid(cr);
1554         ruid = crgetruid(cr);
1555         rgid = crgetrgid(cr);
1556         au_uwrite(au_to_process(uid, gid, ruid, rgid, pid,
1557             ainfo->ai_auid, ainfo->ai_asid, &ainfo->ai_termid));
1558
1559         if (is_system_labeled())
1560             au_uwrite(au_to_label(CR_SL(cr)));
1561
1562         crfree(cr);
1563     }
1564     else
1565         au_uwrite(au_to_arg32(1, "process", (uint32_t)pid));
1566 }
1567
1568 /*ARGSUSED*/
1569 static void
1570 aus_mkdir(struct t_audit_data *tad)
1571 {
1572     klpw_t *clwp = ttolwp(curthread);
1573     uint32_t dmode;
1574
1575     struct a {
1576         long    dirnamep;          /* char * */
1577         long    dmode;
1578     } *uap = (struct a *)clwp->lwp_ap;
1579
1580     dmode = (uint32_t)uap->dmode;
1581
1582     au_uwrite(au_to_arg32(2, "mode", dmode));
1583 }
1584
1585 /*ARGSUSED*/
1586 static void
1587 aus_mkdirat(struct t_audit_data *tad)
1588 {
1589     klpw_t *clwp = ttolwp(curthread);
1590     uint32_t dmode;

```

```

1592     struct a {
1593         long    fd;
1594         long    dirnamep;          /* char * */
1595         long    dmode;
1596     } *uap = (struct a *)clwp->lwp_ap;
1598     dmode = (uint32_t)uap->dmode;
1600     au_uwrite(au_to_arg32(2, "mode", dmode));
1601 }
1603 /*ARGSUSED*/
1604 static void
1605 aus_mknod(struct t_audit_data *tad)
1606 {
1607     klpw_t *clwp = ttolwp(curthread);
1608     uint32_t fmode;
1609     dev_t dev;
1611     struct a {
1612         long    pnamep;          /* char * */
1613         long    fmode;
1614         long    dev;
1615     } *uap = (struct a *)clwp->lwp_ap;
1617     fmode = (uint32_t)uap->fmode;
1618     dev = (dev_t)uap->dev;
1620     au_uwrite(au_to_arg32(2, "mode", fmode));
1621 #ifdef _LP64
1622     au_uwrite(au_to_arg64(3, "dev", dev));
1623 #else
1624     au_uwrite(au_to_arg32(3, "dev", dev));
1625 #endif
1626 }
1628 /*ARGSUSED*/
1629 static void
1630 auf_mknodat(struct t_audit_data *tad, int error, rval_t *rval)
1631 {
1632     klpw_t *clwp = ttolwp(curthread);
1633     vnode_t *dvp;
1634     caddr_t pnamep;
1636     struct a {
1637         long    pnamep;          /* char * */
1638         long    fmode;
1639         long    dev;
1640     } *uap = (struct a *)clwp->lwp_ap;
1642     /* no error, then already path token in audit record */
1643     if (error != EPERM && error != EINVAL)
1644         return;
1646     /* do the lookup to force generation of path token */
1647     pnamep = (caddr_t)uap->pnamep;
1648     tad->tad_ctrl |= TAD_NOATTRB;
1649     error = lookupname(pnamep, UIO_USERSPACE, NO_FOLLOW, &dvp, NULLVPP);
1650     if (error == 0)
1651         VN_RELE(dvp);
1652 }
1654 /*ARGSUSED*/
1655 static void
1656 aus_mknodat(struct t_audit_data *tad)
1657 {

```

```

1658     klpw_t *clwp = ttolwp(curthread);
1659     uint32_t fmode;
1660     dev_t dev;
1662     struct a {
1663         long    fd;
1664         long    pnamep;          /* char * */
1665         long    fmode;
1666         long    dev;
1667     } *uap = (struct a *)clwp->lwp_ap;
1669     fmode = (uint32_t)uap->fmode;
1670     dev = (dev_t)uap->dev;
1672     au_uwrite(au_to_arg32(2, "mode", fmode));
1673 #ifdef _LP64
1674     au_uwrite(au_to_arg64(3, "dev", dev));
1675 #else
1676     au_uwrite(au_to_arg32(3, "dev", dev));
1677 #endif
1678 }
1680 /*ARGSUSED*/
1681 static void
1682 auf_mknodat(struct t_audit_data *tad, int error, rval_t *rval)
1683 {
1684     klpw_t *clwp = ttolwp(curthread);
1685     vnode_t *startvp;
1686     vnode_t *dvp;
1687     caddr_t pnamep;
1688     int fd;
1690     struct a {
1691         long    fd;
1692         long    pnamep;          /* char * */
1693         long    fmode;
1694         long    dev;
1695     } *uap = (struct a *)clwp->lwp_ap;
1697     /* no error, then already path token in audit record */
1698     if (error != EPERM && error != EINVAL)
1699         return;
1701     /* do the lookup to force generation of path token */
1702     fd = (int)uap->fd;
1703     pnamep = (caddr_t)uap->pnamep;
1704     if (pnamep == NULL ||
1705         fgetstartvp(fd, pnamep, &startvp) != 0)
1706         return;
1707     tad->tad_ctrl |= TAD_NOATTRB;
1708     error = lookupnameat(pnamep, UIO_USERSPACE, NO_FOLLOW, &dvp, NULLVPP,
1709         startvp);
1710     if (error == 0)
1711         VN_RELE(dvp);
1712     if (startvp != NULL)
1713         VN_RELE(startvp);
1714 }
1716 /*ARGSUSED*/
1717 static void
1718 aus_mount(struct t_audit_data *tad)
1719 {
1720     /* AUS_START */
1721     klpw_t *clwp = ttolwp(curthread);
1722     uint32_t flags;
1723     uintptr_t u_fstype, dataptr;
1724     STRUCT_DECL(nfs_args, nfsargs);

```



```

1724     size_t len;
1725     char *fstype, *hostname;

1727     struct a {
1728         long    spec;          /* char   **/
1729         long    dir;           /* char   **/
1730         long    flags;
1731         long    fstype;        /* char   **/
1732         long    dataptr;       /* char   **/
1733         long    datalen;
1734     } *uap = (struct a *)clwp->lwp_ap;

1736     u_fstype = (uintptr_t)uap->fstype;
1737     flags    = (uint32_t)uap->flags;
1738     dataptr  = (uintptr_t)uap->dataptr;

1740     fstype = kmem_alloc(MAXNAMELEN, KM_SLEEP);
1741     if (copyinstr((caddr_t)u_fstype, (caddr_t)fstype, MAXNAMELEN, &len))
1742         goto mount_free_fstype;

1744     au_uwrite(au_to_arg32(3, "flags", flags));
1745     au_uwrite(au_to_text(fstype));

1747     if (strncmp(fstype, "nfs", 3) == 0) {

1749         STRUCT_INIT(nfsargs, get_udatamodel());
1750         bzero(STRUCT_BUF(nfsargs), STRUCT_SIZE(nfsargs));

1752         if (copyin((caddr_t)dataptr,
1753                   STRUCT_BUF(nfsargs),
1754                   MIN(uap->datalen, STRUCT_SIZE(nfsargs)))) {
1755             /* DEBUG debug_enter((char *)NULL); */
1756             goto mount_free_fstype;
1757         }
1758         hostname = kmem_alloc(MAXNAMELEN, KM_SLEEP);
1759         if (copyinstr(STRUCT_FGETP(nfsargs, hostname),
1760                     (caddr_t)hostname,
1761                     MAXNAMELEN, &len)) {
1762             goto mount_free_hostname;
1763         }
1764         au_uwrite(au_to_text(hostname));
1765         au_uwrite(au_to_arg32(3, "internal flags",
1766                             (uint_t)STRUCT_FGET(nfsargs, flags)));

1768 mount_free_hostname:
1769         kmem_free(hostname, MAXNAMELEN);
1770     }

1772 mount_free_fstype:
1773     kmem_free(fstype, MAXNAMELEN);
1774 } /* AUS_MOUNT */

1776 static void
1777 aus_umount_path(caddr_t umount_dir)
1778 {
1779     char          *dir_path;
1780     struct audit_path *path;
1781     size_t        path_len, dir_len;

1783     /* length alloc'd for two string pointers */
1784     path_len = sizeof (struct audit_path) + sizeof (char *);
1785     path = kmem_alloc(path_len, KM_SLEEP);
1786     dir_path = kmem_alloc(MAXPATHLEN, KM_SLEEP);

1788     if (copyinstr(umount_dir, (caddr_t)dir_path,
1789                 MAXPATHLEN, &dir_len))

```

```

1790         goto umount2_free_dir;

1792     /*
1793     * the audit_path struct assumes that the buffer pointed to
1794     * by audp_sect[n] contains string 0 immediately followed
1795     * by string 1.
1796     */
1797     path->audp_sect[0] = dir_path;
1798     path->audp_sect[1] = dir_path + strlen(dir_path) + 1;
1799     path->audp_size = path_len;
1800     path->audp_ref = 1;          /* not used */
1801     path->audp_cnt = 1;        /* one path string */

1803     au_uwrite(au_to_path(path));

1805 umount2_free_dir:
1806     kmem_free(dir_path, MAXPATHLEN);
1807     kmem_free(path, path_len);
1808 }

1810 /*ARGSUSED*/
1811 static void
1812 aus_umount2(struct t_audit_data *tad)
1813 {
1814     klpw_t          *clwp = ttolwp(curthread);
1815     struct a {
1816         long    dir;          /* char   **/
1817         long    flags;
1818     } *uap = (struct a *)clwp->lwp_ap;

1820     aus_umount_path((caddr_t)uap->dir);

1822     au_uwrite(au_to_arg32(2, "flags", (uint32_t)uap->flags));
1823 }

1825 static void
1826 aus_msgsys(struct t_audit_data *tad)
1827 {
1828     klpw_t *clwp = ttolwp(curthread);
1829     uint32_t msgid;

1831     struct b {
1832         long    msgid;
1833         long    cmd;
1834         long    buf;          /* struct msqid_ds **/
1835     } *uap1 = (struct b *)&clwp->lwp_ap[1];

1837     msgid = (uint32_t)uap1->msgid;

1840     switch (tad->tad_event) {
1841     case AUE_MSGGET:          /* msgget */
1842         au_uwrite(au_to_arg32(1, "msg key", msgid));
1843         break;
1844     case AUE_MSGCTL:         /* msgctl */
1845     case AUE_MSGCTL_RMID:    /* msgctl */
1846     case AUE_MSGCTL_SET:    /* msgctl */
1847     case AUE_MSGCTL_STAT:   /* msgctl */
1848     case AUE_MSGRCV:        /* msgrcv */
1849     case AUE_MSGSND:        /* msgsnd */
1850         au_uwrite(au_to_arg32(1, "msg ID", msgid));
1851         break;
1852     }
1853 }

1855 /*ARGSUSED*/

```

```

1856 static void
1857 auf_msgsys(struct t_audit_data *tad, int error, rval_t *rval)
1858 {
1859     int id;
1860
1861     if (error != 0)
1862         return;
1863     if (tad->tad_event == AUE_MSGGET) {
1864         uint32_t scid;
1865         uint32_t sy_flags;
1866
1867         /* need to determine type of executing binary */
1868         scid = tad->tad_scid;
1869 #ifdef _SYSCALL32_IMPL
1870         if (lwp_getdatamodel(ttolwp(curthread)) == DATAMODEL_NATIVE)
1871             sy_flags = sysent[scid].sy_flags & SE_RVAL_MASK;
1872         else
1873             sy_flags = sysent32[scid].sy_flags & SE_RVAL_MASK;
1874 #else
1875         sy_flags = sysent[scid].sy_flags & SE_RVAL_MASK;
1876 #endif
1877         if (sy_flags == SE_32RVAL1)
1878             id = rval->r_val1;
1879         if (sy_flags == (SE_32RVAL2|SE_32RVAL1))
1880             id = rval->r_val1;
1881         if (sy_flags == SE_64RVAL)
1882             id = (int)rval->r_vals;
1883
1884         au_uwrite(au_to_ipc(AT_IPC_MSG, id));
1885     }
1886 }
1887
1888 static void
1889 aus_semsys(struct t_audit_data *tad)
1890 {
1891     klwp_t *clwp = ttolwp(curthread);
1892     uint32_t semid;
1893
1894     struct b {
1895         long    semid; /* ctrl */
1896         long    semnum;
1897         long    cmd;
1898         long    arg;
1899     } *uap1 = (struct b *)&clwp->lwp_ap[1];
1900
1901     semid = (uint32_t)uap1->semid;
1902
1903     switch (tad->tad_event) {
1904     case AUE_SEMCTL_RMID:
1905     case AUE_SEMCTL_STAT:
1906     case AUE_SEMCTL_GETNCNT:
1907     case AUE_SEMCTL_GETPID:
1908     case AUE_SEMCTL_GETVAL:
1909     case AUE_SEMCTL_GETALL:
1910     case AUE_SEMCTL_GETZCNT:
1911     case AUE_SEMCTL_SET:
1912     case AUE_SEMCTL_SETVAL:
1913     case AUE_SEMCTL_SETALL:
1914     case AUE_SEMCTL:
1915     case AUE_SEMOP:
1916         au_uwrite(au_to_arg32(1, "sem ID", semid));
1917         break;
1918     case AUE_SEMGET:
1919         au_uwrite(au_to_arg32(1, "sem key", semid));
1920         break;
1921     }

```

```

1922 }
1923
1924 /*ARGSUSED*/
1925 static void
1926 auf_semsys(struct t_audit_data *tad, int error, rval_t *rval)
1927 {
1928     int id;
1929
1930     if (error != 0)
1931         return;
1932     if (tad->tad_event == AUE_SEMGET) {
1933         uint32_t scid;
1934         uint32_t sy_flags;
1935
1936         /* need to determine type of executing binary */
1937         scid = tad->tad_scid;
1938 #ifdef _SYSCALL32_IMPL
1939         if (lwp_getdatamodel(ttolwp(curthread)) == DATAMODEL_NATIVE)
1940             sy_flags = sysent[scid].sy_flags & SE_RVAL_MASK;
1941         else
1942             sy_flags = sysent32[scid].sy_flags & SE_RVAL_MASK;
1943 #else
1944         sy_flags = sysent[scid].sy_flags & SE_RVAL_MASK;
1945 #endif
1946         if (sy_flags == SE_32RVAL1)
1947             id = rval->r_val1;
1948         if (sy_flags == (SE_32RVAL2|SE_32RVAL1))
1949             id = rval->r_val1;
1950         if (sy_flags == SE_64RVAL)
1951             id = (int)rval->r_vals;
1952
1953         au_uwrite(au_to_ipc(AT_IPC_SEM, id));
1954     }
1955 }
1956
1957 /*ARGSUSED*/
1958 static void
1959 aus_close(struct t_audit_data *tad)
1960 {
1961     klwp_t *clwp = ttolwp(curthread);
1962     uint32_t fd;
1963     struct file *fp;
1964     struct f_audit_data *fad;
1965     struct vnode *vp;
1966     struct vattr attr;
1967     au_kcontext_t *kctx = GET_KCTX_PZ;
1968
1969     struct a {
1970         long    i;
1971     } *uap = (struct a *)&clwp->lwp_ap;
1972
1973     fd = (uint32_t)uap->i;
1974
1975     attr.va_mask = 0;
1976     au_uwrite(au_to_arg32(1, "fd", fd));
1977
1978     /*
1979     * convert file pointer to file descriptor
1980     * Note: fd ref count incremented here.
1981     */
1982     if ((fp = getf(fd)) == NULL)
1983         return;
1984
1985     fad = F2A(fp);
1986     tad->tad_evmod = (au_emod_t)fad->fad_flags;
1987     if (fad->fad_aupath != NULL) {

```

```

1988     au_uwrite(au_to_path(fad->fad_aupath));
1989     if ((vp = fp->f_vnode) != NULL) {
1990         attr.va_mask = AT_ALL;
1991         if (VOP_GETATTR(vp, &attr, 0, CRED(), NULL) == 0) {
1992             /*
1993              * When write was not used and the file can be
1994              * considered public, skip the audit.
1995              */
1996             if (((fp->f_flag & FWRITE) == 0) &&
1997                 object_is_public(&attr)) {
1998                 tad->tad_flag = 0;
1999                 tad->tad_evmod = 0;
2000                 /* free any residual audit data */
2001                 au_close(kctx, &(u_ad), 0, 0, 0, NULL);
2002                 releasef(fd);
2003                 return;
2004             }
2005             au_uwrite(au_to_attr(&attr));
2006             audit_sec_attributes(&(u_ad), vp);
2007         }
2008     }
2009 }

2011 /* decrement file descriptor reference count */
2012 releasef(fd);
2013 }

2015 /*ARGSUSED*/
2016 static void
2017 aus_fstatfs(struct t_audit_data *tad)
2018 {
2019     klpw_t *clwp = ttolwp(curthread);
2020     uint32_t fd;
2021     struct file *fp;
2022     struct vnode *vp;
2023     struct f_audit_data *fad;

2025     struct a {
2026         long    fd;
2027         long    buf;          /* struct statfs * */
2028     } *uap = (struct a *)clwp->lwp_ap;

2030     fd = (uint_t)uap->fd;

2032     /*
2033      * convert file pointer to file descriptor
2034      * Note: fd ref count incremented here.
2035      */
2036     if ((fp = getf(fd)) == NULL)
2037         return;

2039     /* get path from file struct here */
2040     fad = F2A(fp);
2041     if (fad->fad_aupath != NULL) {
2042         au_uwrite(au_to_path(fad->fad_aupath));
2043     } else {
2044         au_uwrite(au_to_arg32(1, "no path: fd", fd));
2045     }

2047     vp = fp->f_vnode;
2048     audit_attributes(vp);

2050     /* decrement file descriptor reference count */
2051     releasef(fd);
2052 }

```

```

2054 static au_event_t
2055 aui_setpgrp(au_event_t e)
2056 {
2057     klpw_t *clwp = ttolwp(curthread);
2058     int flag;

2060     struct a {
2061         long    flag;
2062         long    pid;
2063         long    pgid;
2064     } *uap = (struct a *)clwp->lwp_ap;

2066     flag = (int)uap->flag;

2069     switch (flag) {

2071     case 1: /* setpgrp() */
2072         e = AUE_SETPGRP;
2073         break;

2075     case 3: /* setsid() */
2076         e = AUE_SETSID;
2077         break;

2079     case 5: /* setpgid() */
2080         e = AUE_SETPGID;
2081         break;

2083     case 0: /* getpgrp() - not security relevant */
2084     case 2: /* getsid() - not security relevant */
2085     case 4: /* getpgid() - not security relevant */
2086         e = AUE_NULL;
2087         break;

2089     default:
2090         e = AUE_NULL;
2091         break;
2092     }

2094     return (e);
2095 }

2097 /*ARGSUSED*/
2098 static void
2099 aus_setpgrp(struct t_audit_data *tad)
2100 {
2101     klpw_t        *clwp = ttolwp(curthread);
2102     pid_t         pgid;
2103     struct proc   *p;
2104     uid_t         uid, ruid;
2105     gid_t         gid, rgid;
2106     pid_t         pid;
2107     cred_t        *cr;
2108     int           flag;
2109     const auditinfo_addr_t *ainfo;

2111     struct a {
2112         long    flag;
2113         long    pid;
2114         long    pgid;
2115     } *uap = (struct a *)clwp->lwp_ap;

2117     flag = (int)uap->flag;
2118     pid = (pid_t)uap->pid;
2119     pgid = (pid_t)uap->pgid;

```

```

2122     switch (flag) {
2124     case 0: /* getpgrp() */
2125     case 1: /* setpgrp() */
2126     case 2: /* getsid() */
2127     case 3: /* setsid() */
2128     case 4: /* getpgid() */
2129         break;
2131     case 5: /* setpgid() */
2133         /* current process? */
2134         if (pid == 0) {
2135             return;
2136         }
2138         mutex_enter(&pidlock);
2139         p = prfind(pid);
2140         if (p == NULL || p->p_as == &kas ||
2141             p->p_stat == SIDL || p->p_stat == SZOMB) {
2142             mutex_exit(&pidlock);
2143             return;
2144         }
2145         mutex_enter(&p->p_lock);          /* so process doesn't go away */
2146         mutex_exit(&pidlock);
2148         mutex_enter(&p->p_crlock);
2149         crhold(cr = p->p_cred);
2150         mutex_exit(&p->p_crlock);
2151         mutex_exit(&p->p_lock);
2153         ainfo = crgetauido(cr);
2154         if (ainfo == NULL) {
2155             crfree(cr);
2156             return;
2157         }
2159         uid = crgetuid(cr);
2160         gid = crgetgid(cr);
2161         ruid = crgetruid(cr);
2162         rgid = crgetrgid(cr);
2163         au_uwrite(au_to_process(uid, gid, ruid, rgid, pid,
2164             ainfo->ai_auid, ainfo->ai_asid, &ainfo->ai_termid));
2165         crfree(cr);
2166         au_uwrite(au_to_arg32(2, "pgid", pgid));
2167         break;
2169     default:
2170         break;
2171     }
2172 }
2175 /*ARGSUSED*/
2176 static void
2177 aus_setregid(struct t_audit_data *tad)
2178 {
2179     klpw_t *clwp = ttolwp(curthread);
2180     uint32_t rgid, egid;
2182     struct a {
2183         long    rgid;
2184         long    egid;
2185     } *uap = (struct a *)clwp->lwp_ap;

```

```

2187         rgid = (uint32_t)uap->rgid;
2188         egid = (uint32_t)uap->egid;
2190         au_uwrite(au_to_arg32(1, "rgid", rgid));
2191         au_uwrite(au_to_arg32(2, "egid", egid));
2192     }
2194 /*ARGSUSED*/
2195 static void
2196 aus_setgid(struct t_audit_data *tad)
2197 {
2198     klpw_t *clwp = ttolwp(curthread);
2199     uint32_t gid;
2201     struct a {
2202         long    gid;
2203     } *uap = (struct a *)clwp->lwp_ap;
2205     gid = (uint32_t)uap->gid;
2207     au_uwrite(au_to_arg32(1, "gid", gid));
2208 }
2211 /*ARGSUSED*/
2212 static void
2213 aus_setreuid(struct t_audit_data *tad)
2214 {
2215     klpw_t *clwp = ttolwp(curthread);
2216     uint32_t ruid, euid;
2218     struct a {
2219         long    ruid;
2220         long    euid;
2221     } *uap = (struct a *)clwp->lwp_ap;
2223     ruid = (uint32_t)uap->ruid;
2224     euid = (uint32_t)uap->euid;
2226     au_uwrite(au_to_arg32(1, "ruid", ruid));
2227     au_uwrite(au_to_arg32(2, "euid", euid));
2228 }
2231 /*ARGSUSED*/
2232 static void
2233 aus_setuid(struct t_audit_data *tad)
2234 {
2235     klpw_t *clwp = ttolwp(curthread);
2236     uint32_t uid;
2238     struct a {
2239         long    uid;
2240     } *uap = (struct a *)clwp->lwp_ap;
2242     uid = (uint32_t)uap->uid;
2244     au_uwrite(au_to_arg32(1, "uid", uid));
2245 }
2247 /*ARGSUSED*/
2248 static void
2249 aus_shmsys(struct t_audit_data *tad)
2250 {
2251     klpw_t *clwp = ttolwp(curthread);

```

```

2252     uint32_t id, cmd;

2254     struct b {
2255         long   id;
2256         long   cmd;
2257         long   buf;
2258     } *uapl = (struct b *)&clwp->lwp_ap[1];

2260     id = (uint32_t)uapl->id;
2261     cmd = (uint32_t)uapl->cmd;

2263     switch (tad->tad_event) {
2264     case AUE_SHMGET: /* shmget */
2265         au_uwrite(au_to_arg32(1, "shm key", id));
2266         break;
2267     case AUE_SHMCTL: /* shmctl */
2268     case AUE_SHMCTL_RMID: /* shmctl */
2269     case AUE_SHMCTL_STAT: /* shmctl */
2270     case AUE_SHMCTL_SET: /* shmctl */
2271         au_uwrite(au_to_arg32(1, "shm ID", id));
2272         break;
2273     case AUE_SHMDT: /* shmdt */
2274         au_uwrite(au_to_arg32(1, "shm adr", id));
2275         break;
2276     case AUE_SHMAT: /* shmat */
2277         au_uwrite(au_to_arg32(1, "shm ID", id));
2278         au_uwrite(au_to_arg32(2, "shm adr", cmd));
2279         break;
2280     }
2281 }

2283 /*ARGSUSED*/
2284 static void
2285 auf_shmsys(struct t_audit_data *tad, int error, rval_t *rval)
2286 {
2287     int id;

2289     if (error != 0)
2290         return;
2291     if (tad->tad_event == AUE_SHMGET) {
2292         uint32_t scid;
2293         uint32_t sy_flags;

2295         /* need to determine type of executing binary */
2296         scid = tad->tad_scid;
2297 #ifdef _SYSCALL32_IMPL
2298         if (lwp_getdatamodel(ttolwp(curthread)) == DATAMODEL_NATIVE)
2299             sy_flags = sysent[scid].sy_flags & SE_RVAL_MASK;
2300         else
2301             sy_flags = sysent32[scid].sy_flags & SE_RVAL_MASK;
2302 #else
2303         sy_flags = sysent[scid].sy_flags & SE_RVAL_MASK;
2304 #endif
2305         if (sy_flags == SE_32RVAL1)
2306             id = rval->r_val1;
2307         if (sy_flags == (SE_32RVAL2|SE_32RVAL1))
2308             id = rval->r_val1;
2309         if (sy_flags == SE_64RVAL)
2310             id = (int)rval->r_vals;
2311         au_uwrite(au_to_ipc(AT_IPC_SHM, id));
2312     }
2313 }

2316 /*ARGSUSED*/
2317 static void

```

```

2318 aus_ioctl(struct t_audit_data *tad)
2319 {
2320     klpw_t *clwp = ttolwp(curthread);
2321     struct file *fp;
2322     struct vnode *vp;
2323     struct f_audit_data *fad;
2324     uint32_t fd, cmd;
2325     uintptr_t cmarg;

2327     /* XX64 */
2328     struct a {
2329         long   fd;
2330         long   cmd;
2331         long   cmarg; /* caddr_t */
2332     } *uap = (struct a *)&clwp->lwp_ap;

2334     fd = (uint32_t)uap->fd;
2335     cmd = (uint32_t)uap->cmd;
2336     cmarg = (uintptr_t)uap->cmarg;

2338     /*
2339     * convert file pointer to file descriptor
2340     * Note: fd ref count incremented here.
2341     */
2342     if ((fp = getf(fd)) == NULL) {
2343         au_uwrite(au_to_arg32(1, "fd", fd));
2344         au_uwrite(au_to_arg32(2, "cmd", cmd));
2345 #ifndef _LP64
2346         au_uwrite(au_to_arg32(3, "arg", (uint32_t)cmarg));
2347 #else
2348         au_uwrite(au_to_arg64(3, "arg", (uint64_t)cmarg));
2349 #endif
2350     }
2351     return;

2353     /* get path from file struct here */
2354     fad = F2A(fp);
2355     if (fad->fad_aupath != NULL) {
2356         au_uwrite(au_to_path(fad->fad_aupath));
2357     } else {
2358         au_uwrite(au_to_arg32(1, "no path: fd", fd));
2359     }

2361     vp = fp->f_vnode;
2362     audit_attributes(vp);

2364     /* decrement file descriptor reference count */
2365     releasef(fd);

2367     au_uwrite(au_to_arg32(2, "cmd", cmd));
2368 #ifndef _LP64
2369     au_uwrite(au_to_arg32(3, "arg", (uint32_t)cmarg));
2370 #else
2371     au_uwrite(au_to_arg64(3, "arg", (uint64_t)cmarg));
2372 #endif
2373 }

2375 /*
2376 * null function for memcntl for now. We might want to limit memcntl()
2377 * auditing to commands: MC_LOCKAS, MC_LOCK, MC_UNLOCKAS, MC_UNLOCK which
2378 * require privileges.
2379 */
2380 static au_event_t
2381 aui_memcntl(au_event_t e)
2382 {
2383     return (e);

```

```

2384 }
2386 /*ARGSUSED*/
2387 static au_event_t
2388 aui_privsys(au_event_t e)
2389 {
2390     klpw_t *clwp = ttolwp(curthread);
2392     struct a {
2393         long    opcode;
2394     } *uap = (struct a *)clwp->lwp_ap;
2396     switch (uap->opcode) {
2397     case PRIVSYS_SETPPRIV:
2398         return (AUE_SETPPRIV);
2399     default:
2400         return (AUE_NULL);
2401     }
2402 }
2404 /*ARGSUSED*/
2405 static void
2406 aus_memcntl(struct t_audit_data *tad)
2407 {
2408     klpw_t *clwp = ttolwp(curthread);
2410     struct a {
2411         long    addr;
2412         long    len;
2413         long    cmd;
2414         long    arg;
2415         long    attr;
2416         long    mask;
2417     } *uap = (struct a *)clwp->lwp_ap;
2419 #ifdef _LP64
2420     au_uwrite(au_to_arg64(1, "base", (uint64_t)uap->addr));
2421     au_uwrite(au_to_arg64(2, "len", (uint64_t)uap->len));
2422 #else
2423     au_uwrite(au_to_arg32(1, "base", (uint32_t)uap->addr));
2424     au_uwrite(au_to_arg32(2, "len", (uint32_t)uap->len));
2425 #endif
2426     au_uwrite(au_to_arg32(3, "cmd", (uint_t)uap->cmd));
2427 #ifdef _LP64
2428     au_uwrite(au_to_arg64(4, "arg", (uint64_t)uap->arg));
2429 #else
2430     au_uwrite(au_to_arg32(4, "arg", (uint32_t)uap->arg));
2431 #endif
2432     au_uwrite(au_to_arg32(5, "attr", (uint_t)uap->attr));
2433     au_uwrite(au_to_arg32(6, "mask", (uint_t)uap->mask));
2434 }
2436 /*ARGSUSED*/
2437 static void
2438 aus_mmap(struct t_audit_data *tad)
2439 {
2440     klpw_t *clwp = ttolwp(curthread);
2441     struct file *fp;
2442     struct f_audit_data *fad;
2443     struct vnode *vp;
2444     uint32_t fd;
2446     struct a {
2447         long    addr;
2448         long    len;
2449         long    prot;

```

```

2450         long    flags;
2451         long    fd;
2452         long    pos;
2453     } *uap = (struct a *)clwp->lwp_ap;
2455     fd = (uint32_t)uap->fd;
2457 #ifdef _LP64
2458     au_uwrite(au_to_arg64(1, "addr", (uint64_t)uap->addr));
2459     au_uwrite(au_to_arg64(2, "len", (uint64_t)uap->len));
2460 #else
2461     au_uwrite(au_to_arg32(1, "addr", (uint32_t)uap->addr));
2462     au_uwrite(au_to_arg32(2, "len", (uint32_t)uap->len));
2463 #endif
2465     if ((fp = getf(fd)) == NULL) {
2466         au_uwrite(au_to_arg32(5, "fd", (uint32_t)uap->fd));
2467         return;
2468     }
2470     /*
2471     * Mark in the tad if write access is NOT requested... if
2472     * this is later detected (in audit_attributes) to be a
2473     * public object, the mmap event may be discarded.
2474     */
2475     if (((uap->prot) & PROT_WRITE) == 0) {
2476         tad->tad_ctrl |= TAD_PUBLIC_EV;
2477     }
2479     fad = F2A(fp);
2480     if (fad->fad_aupath != NULL) {
2481         au_uwrite(au_to_path(fad->fad_aupath));
2482     } else {
2483         au_uwrite(au_to_arg32(1, "no path: fd", fd));
2484     }
2486     vp = (struct vnode *)fp->f_vnode;
2487     audit_attributes(vp);
2489     /* mark READ/WRITE since we can't predict access */
2490     if (uap->prot & PROT_READ)
2491         fad->fad_flags |= FAD_READ;
2492     if (uap->prot & PROT_WRITE)
2493         fad->fad_flags |= FAD_WRITE;
2495     /* decrement file descriptor reference count */
2496     releasef(fd);
2498 } /* AUS_MMAP */
2503 /*ARGSUSED*/
2504 static void
2505 aus_munmap(struct t_audit_data *tad)
2506 {
2507     klpw_t *clwp = ttolwp(curthread);
2509     struct a {
2510         long    addr;
2511         long    len;
2512     } *uap = (struct a *)clwp->lwp_ap;
2514 #ifdef _LP64
2515     au_uwrite(au_to_arg64(1, "addr", (uint64_t)uap->addr));

```

```

2516     au_uwrite(au_to_arg64(2, "len", (uint64_t)uap->len));
2517 #else
2518     au_uwrite(au_to_arg32(1, "addr", (uint32_t)uap->addr));
2519     au_uwrite(au_to_arg32(2, "len", (uint32_t)uap->len));
2520 #endif
2522 }      /* AUS_MUNMAP */

2530 /*ARGSUSED*/
2531 static void
2532 aus_priocntlsys(struct t_audit_data *tad)
2533 {
2534     klwp_t *clwp = ttolwp(curthread);

2536     struct a {
2537         long    pc_version;
2538         long    psp;          /* procset_t */
2539         long    cmd;
2540         long    arg;
2541     } *uap = (struct a *)clwp->lwp_ap;

2543     au_uwrite(au_to_arg32(1, "pc_version", (uint32_t)uap->pc_version));
2544     au_uwrite(au_to_arg32(3, "cmd", (uint32_t)uap->cmd));

2546 }      /* AUS_PRIOCNLSYS */

2549 /*ARGSUSED*/
2550 static void
2551 aus_setegid(struct t_audit_data *tad)
2552 {
2553     klwp_t *clwp = ttolwp(curthread);
2554     uint32_t gid;

2556     struct a {
2557         long    gid;
2558     } *uap = (struct a *)clwp->lwp_ap;

2560     gid = (uint32_t)uap->gid;

2562     au_uwrite(au_to_arg32(1, "gid", gid));
2563 }      /* AUS_SETEGID */

2568 /*ARGSUSED*/
2569 static void
2570 aus_setgroups(struct t_audit_data *tad)
2571 {
2572     klwp_t *clwp = ttolwp(curthread);
2573     int i;
2574     int gidsetsize;
2575     uintptr_t gidset;
2576     gid_t *gidlist;

2578     struct a {
2579         long    gidsetsize;
2580         long    gidset;
2581     } *uap = (struct a *)clwp->lwp_ap;

```

```

2583     gidsetsize = (uint_t)uap->gidsetsize;
2584     gidset = (uintptr_t)uap->gidset;

2586     if ((gidsetsize > NGROUPS_MAX_DEFAULT) || (gidsetsize < 0))
2587         return;
2588     if (gidsetsize != 0) {
2589         gidlist = kmem_alloc(gidsetsize * sizeof (gid_t),
2590             KM_SLEEP);
2591         if (copyin((caddr_t)gidset, gidlist,
2592             gidsetsize * sizeof (gid_t)) == 0)
2593             for (i = 0; i < gidsetsize; i++)
2594                 au_uwrite(au_to_arg32(1, "setgroups",
2595                     (uint32_t)gidlist[i]));
2596         kmem_free(gidlist, gidsetsize * sizeof (gid_t));
2597     } else
2598         au_uwrite(au_to_arg32(1, "setgroups", (uint32_t)0));

2600 }      /* AUS_SETGROUPS */

2606 /*ARGSUSED*/
2607 static void
2608 aus_seteuid(struct t_audit_data *tad)
2609 {
2610     klwp_t *clwp = ttolwp(curthread);
2611     uint32_t uid;

2613     struct a {
2614         long    uid;
2615     } *uap = (struct a *)clwp->lwp_ap;

2617     uid = (uint32_t)uap->uid;

2619     au_uwrite(au_to_arg32(1, "euid", uid));

2621 }      /* AUS_SETEUID */

2623 /*ARGSUSED*/
2624 static void
2625 aus_putmsg(struct t_audit_data *tad)
2626 {
2627     klwp_t *clwp = ttolwp(curthread);
2628     uint32_t fd, pri;
2629     struct file *fp;
2630     struct f_audit_data *fad;

2632     struct a {
2633         long    fdes;
2634         long    ctl;          /* struct strbuf */
2635         long    data;         /* struct strbuf */
2636         long    pri;
2637     } *uap = (struct a *)clwp->lwp_ap;

2639     fd = (uint32_t)uap->fdes;
2640     pri = (uint32_t)uap->pri;

2642     au_uwrite(au_to_arg32(1, "fd", fd));

2644     if ((fp = getf(fd)) != NULL) {
2645         fad = F2A(fp);

2647         fad->fad_flags |= FAD_WRITE;

```

```

2649         /* add path name to audit record */
2650         if (fad->fad_aupath != NULL) {
2651             au_uwrite(au_to_path(fad->fad_aupath));
2652         }
2653         audit_attributes(fp->f_vnode);
2655         releasef(fd);
2656     }
2658     au_uwrite(au_to_arg32(4, "pri", pri));
2659 }

2661 /*ARGSUSED*/
2662 static void
2663 aus_putpmsg(struct t_audit_data *tad)
2664 {
2665     klpw_t *clwp = ttolwp(curthread);
2666     uint32_t fd, pri, flags;
2667     struct file *fp;
2668     struct f_audit_data *fad;

2670     struct a {
2671         long     fdes;
2672         long     ctl;           /* struct strbuf * */
2673         long     data;         /* struct strbuf * */
2674         long     pri;
2675         long     flags;
2676     } *uap = (struct a *)clwp->lwp_ap;

2678     fd = (uint32_t)uap->fdes;
2679     pri = (uint32_t)uap->pri;
2680     flags = (uint32_t)uap->flags;

2682     au_uwrite(au_to_arg32(1, "fd", fd));

2684     if ((fp = getf(fd)) != NULL) {
2685         fad = F2A(fp);

2687         fad->fad_flags |= FAD_WRITE;

2689         /* add path name to audit record */
2690         if (fad->fad_aupath != NULL) {
2691             au_uwrite(au_to_path(fad->fad_aupath));
2692         }
2693         audit_attributes(fp->f_vnode);

2695         releasef(fd);
2696     }

2699     au_uwrite(au_to_arg32(4, "pri", pri));
2700     au_uwrite(au_to_arg32(5, "flags", flags));
2701 }

2703 /*ARGSUSED*/
2704 static void
2705 aus_getmsg(struct t_audit_data *tad)
2706 {
2707     klpw_t *clwp = ttolwp(curthread);
2708     uint32_t fd, pri;
2709     struct file *fp;
2710     struct f_audit_data *fad;

2712     struct a {
2713         long     fdes;

```

```

2714         long     ctl;           /* struct strbuf * */
2715         long     data;         /* struct strbuf * */
2716         long     pri;
2717     } *uap = (struct a *)clwp->lwp_ap;

2719     fd = (uint32_t)uap->fdes;
2720     pri = (uint32_t)uap->pri;

2722     au_uwrite(au_to_arg32(1, "fd", fd));

2724     if ((fp = getf(fd)) != NULL) {
2725         fad = F2A(fp);

2727         /*
2728          * read operation on this object
2729          */
2730         fad->fad_flags |= FAD_READ;

2732         /* add path name to audit record */
2733         if (fad->fad_aupath != NULL) {
2734             au_uwrite(au_to_path(fad->fad_aupath));
2735         }
2736         audit_attributes(fp->f_vnode);

2738         releasef(fd);
2739     }

2741     au_uwrite(au_to_arg32(4, "pri", pri));
2742 }

2744 /*ARGSUSED*/
2745 static void
2746 aus_getpmsg(struct t_audit_data *tad)
2747 {
2748     klpw_t *clwp = ttolwp(curthread);
2749     uint32_t fd;
2750     struct file *fp;
2751     struct f_audit_data *fad;

2753     struct a {
2754         long     fdes;
2755         long     ctl;           /* struct strbuf * */
2756         long     data;         /* struct strbuf * */
2757         long     pri;
2758         long     flags;
2759     } *uap = (struct a *)clwp->lwp_ap;

2761     fd = (uint32_t)uap->fdes;

2763     au_uwrite(au_to_arg32(1, "fd", fd));

2765     if ((fp = getf(fd)) != NULL) {
2766         fad = F2A(fp);

2768         /*
2769          * read operation on this object
2770          */
2771         fad->fad_flags |= FAD_READ;

2773         /* add path name to audit record */
2774         if (fad->fad_aupath != NULL) {
2775             au_uwrite(au_to_path(fad->fad_aupath));
2776         }
2777         audit_attributes(fp->f_vnode);

2779         releasef(fd);

```



```

2780     }
2781 }

2783 static au_event_t
2784 aus_labelsys(au_event_t e)
2785 {
2786     klwp_t *clwp = ttolwp(curthread);
2787     uint32_t code;
2788     uint32_t cmd;

2790     struct a {
2791         long    code;
2792         long    cmd;
2793     } *uap = (struct a *)clwp->lwp_ap;

2795     code = (uint32_t)uap->code;
2796     cmd = (uint32_t)uap->cmd;

2798     /* not security relevant if not changing kernel cache */
2799     if (cmd == TNDB_GET)
2800         return (AUE_NULL);

2802     switch (code) {
2803     case TSOL_TNRH:
2804         e = AUE_LABELSYS_TNRH;
2805         break;
2806     case TSOL_TNRHTP:
2807         e = AUE_LABELSYS_TNRHTP;
2808         break;
2809     case TSOL_TNMPL:
2810         e = AUE_LABELSYS_TNMPL;
2811         break;
2812     default:
2813         e = AUE_NULL;
2814         break;
2815     }

2817     return (e);

2819 }

2821 static void
2822 aus_labelsys(struct t_audit_data *tad)
2823 {
2824     klwp_t *clwp = ttolwp(curthread);
2825     uint32_t cmd;
2826     uintptr_t a2;

2828     struct a {
2829         long    code;
2830         long    cmd;
2831         long    a2;
2832     } *uap = (struct a *)clwp->lwp_ap;

2834     cmd = (uint32_t)uap->cmd;
2835     a2 = (uintptr_t)uap->a2;

2837     switch (tad->tad_event) {
2838     case AUE_LABELSYS_TNRH:
2839     {
2840         tsol_rhent_t    *rhent;
2841         tnaddr_t        *rh_addr;

2843         au_uwrite(au_to_arg32(1, "cmd", cmd));

2845         /* Remaining args don't apply for FLUSH, so skip */

```

```

2846         if (cmd == TNDB_FLUSH)
2847             break;

2849         rhent = kmem_alloc(sizeof (tsol_rhent_t), KM_SLEEP);
2850         if (copyin((caddr_t)a2, rhent, sizeof (tsol_rhent_t))) {
2851             kmem_free(rhent, sizeof (tsol_rhent_t));
2852             return;
2853         }

2855         rh_addr = &rhent->rh_address;
2856         if (rh_addr->ta_family == AF_INET) {
2857             struct in_addr    *ipaddr;

2859             ipaddr = &(rh_addr->ta_addr_v4);
2860             au_uwrite(au_to_in_addr(ipaddr));
2861         } else if (rh_addr->ta_family == AF_INET6) {
2862             int32_t    *ipaddr;

2864             ipaddr = (int32_t *)&(rh_addr->ta_addr_v6);
2865             au_uwrite(au_to_in_addr_ex(ipaddr));
2866         }
2867         au_uwrite(au_to_arg32(2, "prefix len", rhent->rh_prefix));

2869         kmem_free(rhent, sizeof (tsol_rhent_t));

2871         break;
2872     }
2873     case AUE_LABELSYS_TNRHTP:
2874     {
2875         tsol_tpent_t    *tpent;

2877         au_uwrite(au_to_arg32(1, "cmd", cmd));

2879         /* Remaining args don't apply for FLUSH, so skip */
2880         if (cmd == TNDB_FLUSH)
2881             break;

2883         tpent = kmem_alloc(sizeof (tsol_tpent_t), KM_SLEEP);
2884         if (copyin((caddr_t)a2, tpent, sizeof (tsol_tpent_t))) {
2885             kmem_free(tpent, sizeof (tsol_tpent_t));
2886             return;
2887         }

2889         /* Make sure that the template name is null-terminated. */
2890         *(tpent->name + TNINAMSIZ - 1) = '\0';

2892         au_uwrite(au_to_text(tpent->name));
2893         kmem_free(tpent, sizeof (tsol_tpent_t));

2895         break;
2896     }
2897     case AUE_LABELSYS_TNMPL:
2898     {
2899         tsol_mlpent_t    *mlpent;

2901         au_uwrite(au_to_arg32(1, "cmd", cmd));

2903         mlpent = kmem_alloc(sizeof (tsol_mlpent_t), KM_SLEEP);
2904         if (copyin((caddr_t)a2, mlpent, sizeof (tsol_mlpent_t))) {
2905             kmem_free(mlpent, sizeof (tsol_mlpent_t));
2906             return;
2907         }

2909         if (mlpent->tsme_flags & TSOL_MEF_SHARED) {
2910             au_uwrite(au_to_text("shared"));
2911         } else {

```

```

2912         zone_t *zone;
2913
2914         zone = zone_find_by_id(mlpent->tsme_zoneid);
2915         if (zone != NULL) {
2916             au_uwrite(au_to_text(zone->zone_name));
2917             zone_rele(zone);
2918         }
2919     }
2920
2921     /* Remaining args don't apply for FLUSH, so skip */
2922     if (cmd == TNDE_FLUSH) {
2923         kmem_free(mlpent, sizeof (tsol_mlpent_t));
2924         break;
2925     }
2926
2927     au_uwrite(au_to_arg32(2, "proto num",
2928         (uint32_t)mlpent->tsme_mlp.mlp_ipp));
2929     au_uwrite(au_to_arg32(2, "mlp_port",
2930         (uint32_t)mlpent->tsme_mlp.mlp_port));
2931
2932     if (mlpent->tsme_mlp.mlp_port_upper != 0)
2933         au_uwrite(au_to_arg32(2, "mlp_port_upper",
2934             (uint32_t)mlpent->tsme_mlp.mlp_port_upper));
2935
2936     kmem_free(mlpent, sizeof (tsol_mlpent_t));
2937
2938     break;
2939 }
2940 default:
2941     break;
2942 }
2943 }
2944
2945
2946 static au_event_t
2947 aui_auditsys(au_event_t e)
2948 {
2949     klpw_t *clwp = ttolwp(curthread);
2950     uint32_t code;
2951
2952     struct a {
2953         long    code;
2954         long    a1;
2955         long    a2;
2956         long    a3;
2957         long    a4;
2958         long    a5;
2959         long    a6;
2960         long    a7;
2961     } *uap = (struct a *)clwp->lwp_ap;
2962
2963     code = (uint32_t)uap->code;
2964
2965     switch (code) {
2966     case BSM_GETAUDIT:
2967         e = AUE_GETAUDIT;
2968         break;
2969     case BSM_SETAUDIT:
2970         e = AUE_SETAUDIT;
2971         break;
2972     case BSM_GETAUDIT_ADDR:
2973         e = AUE_GETAUDIT_ADDR;
2974         break;
2975     case BSM_SETAUDIT_ADDR:
2976         e = AUE_SETAUDIT_ADDR;
2977         break;

```

```

2978         break;
2979     case BSM_SETAUDIT:
2980         e = AUE_SETAUDIT;
2981         break;
2982     case BSM_SETAUDIT_ADDR:
2983         e = AUE_SETAUDIT_ADDR;
2984         break;
2985     case BSM_AUDIT:
2986         e = AUE_AUDIT;
2987         break;
2988     case BSM_AUDITCTL:
2989         switch ((uint_t)uap->a1) {
2990
2991         case A_GETPOLICY:
2992             e = AUE_AUDITON_GPOLICY;
2993             break;
2994         case A_SETPOLICY:
2995             e = AUE_AUDITON_SPOLICY;
2996             break;
2997         case A_GETAMASK:
2998             e = AUE_AUDITON_GETAMASK;
2999             break;
3000         case A_SETAMASK:
3001             e = AUE_AUDITON_SETAMASK;
3002             break;
3003         case A_GETKMASK:
3004             e = AUE_AUDITON_GETKMASK;
3005             break;
3006         case A_SETKMASK:
3007             e = AUE_AUDITON_SETKMASK;
3008             break;
3009         case A_GETQCTRL:
3010             e = AUE_AUDITON_GQCTRL;
3011             break;
3012         case A_SETQCTRL:
3013             e = AUE_AUDITON_SQCTRL;
3014             break;
3015         case A_GETCWD:
3016             e = AUE_AUDITON_GETCWD;
3017             break;
3018         case A_GETCAR:
3019             e = AUE_AUDITON_GETCAR;
3020             break;
3021         case A_GETSTAT:
3022             e = AUE_AUDITON_GETSTAT;
3023             break;
3024         case A_SETSTAT:
3025             e = AUE_AUDITON_SETSTAT;
3026             break;
3027         case A_SETUMASK:
3028             e = AUE_AUDITON_SETUMASK;
3029             break;
3030         case A_SETSMASK:
3031             e = AUE_AUDITON_SETSMASK;
3032             break;
3033         case A_GETCOND:
3034             e = AUE_AUDITON_GETCOND;
3035             break;
3036         case A_SETCOND:
3037             e = AUE_AUDITON_SETCOND;
3038             break;
3039         case A_GETCLASS:
3040             e = AUE_AUDITON_GETCLASS;
3041             break;
3042         case A_SETCLASS:
3043             e = AUE_AUDITON_SETCLASS;

```

```

3044         break;
3045     default:
3046         e = AUE_NULL;
3047         break;
3048     }
3049     break;
3050 default:
3051     e = AUE_NULL;
3052     break;
3053 }
3055 return (e);
3057 } /* AUI_AUDITSYS */

3060 static void
3061 aus_auditsys(struct t_audit_data *tad)
3062 {
3063     klpw_t *clwp = ttolwp(curthread);
3064     uintptr_t a1, a2;
3065     STRUCT_DECL(auditinfo, ainfo);
3066     STRUCT_DECL(auditinfo_addr, ainfo_addr);
3067     au_evclass_map_t event;
3068     au_mask_t mask;
3069     int auditstate, policy;
3070     au_id_t auid;

3073     struct a {
3074         long    code;
3075         long    a1;
3076         long    a2;
3077         long    a3;
3078         long    a4;
3079         long    a5;
3080         long    a6;
3081         long    a7;
3082     } *uap = (struct a *)clwp->lwp_ap;

3084     a1 = (uintptr_t)uap->a1;
3085     a2 = (uintptr_t)uap->a2;

3087     switch (tad->tad_event) {
3088     case AUE_SETAUDIT:
3089         if (copyin((caddr_t)a1, &auid, sizeof (au_id_t)))
3090             return;
3091         au_uwrite(au_to_arg32(2, "setauid", auid));
3092         break;
3093     case AUE_SETAUDIT:
3094         STRUCT_INIT(ainfo, get_udatamodel());
3095         if (copyin((caddr_t)a1, STRUCT_BUF(ainfo),
3096                 STRUCT_SIZE(ainfo))) {
3097             return;
3098         }
3099         au_uwrite(au_to_arg32((char)1, "setaudit:auid",
3100                             (uint32_t)STRUCT_FGET(ainfo, ai_auid)));
3101 #ifdef_LP64
3102         au_uwrite(au_to_arg64((char)1, "setaudit:port",
3103                             (uint64_t)STRUCT_FGET(ainfo, ai_termid.port)));
3104 #else
3105         au_uwrite(au_to_arg32((char)1, "setaudit:port",
3106                             (uint32_t)STRUCT_FGET(ainfo, ai_termid.port)));
3107 #endif
3108         au_uwrite(au_to_arg32((char)1, "setaudit:machine",
3109                             (uint32_t)STRUCT_FGET(ainfo, ai_termid.machine)));

```

```

3110         au_uwrite(au_to_arg32((char)1, "setaudit:as_success",
3111                             (uint32_t)STRUCT_FGET(ainfo, ai_mask.as_success)));
3112         au_uwrite(au_to_arg32((char)1, "setaudit:as_failure",
3113                             (uint32_t)STRUCT_FGET(ainfo, ai_mask.as_failure)));
3114         au_uwrite(au_to_arg32((char)1, "setaudit:asid",
3115                             (uint32_t)STRUCT_FGET(ainfo, ai_asid)));
3116         break;
3117     case AUE_SETAUDIT_ADDR:
3118         STRUCT_INIT(ainfo_addr, get_udatamodel());
3119         if (copyin((caddr_t)a1, STRUCT_BUF(ainfo_addr),
3120                 STRUCT_SIZE(ainfo_addr))) {
3121             return;
3122         }
3123         au_uwrite(au_to_arg32((char)1, "auid",
3124                             (uint32_t)STRUCT_FGET(ainfo_addr, ai_auid)));
3125 #ifdef_LP64
3126         au_uwrite(au_to_arg64((char)1, "port",
3127                             (uint64_t)STRUCT_FGET(ainfo_addr, ai_termid.at_port)));
3128 #else
3129         au_uwrite(au_to_arg32((char)1, "port",
3130                             (uint32_t)STRUCT_FGET(ainfo_addr, ai_termid.at_port)));
3131 #endif
3132         au_uwrite(au_to_arg32((char)1, "type",
3133                             (uint32_t)STRUCT_FGET(ainfo_addr, ai_termid.at_type)));
3134         if ((uint32_t)STRUCT_FGET(ainfo_addr, ai_termid.at_type) ==
3135             AU_IPv4) {
3136             au_uwrite(au_to_in_addr(
3137                 (struct in_addr *)STRUCT_FGETP(ainfo_addr,
3138                 ai_termid.at_addr)));
3139         } else {
3140             au_uwrite(au_to_in_addr_ex(
3141                 (int32_t *)STRUCT_FGETP(ainfo_addr,
3142                 ai_termid.at_addr)));
3143         }
3144         au_uwrite(au_to_arg32((char)1, "as_success",
3145                             (uint32_t)STRUCT_FGET(ainfo_addr, ai_mask.as_success)));
3146         au_uwrite(au_to_arg32((char)1, "as_failure",
3147                             (uint32_t)STRUCT_FGET(ainfo_addr, ai_mask.as_failure)));
3148         au_uwrite(au_to_arg32((char)1, "asid",
3149                             (uint32_t)STRUCT_FGET(ainfo_addr, ai_asid)));
3150         break;
3151     case AUE_AUDITON_SETAMASK:
3152         if (copyin((caddr_t)a2, &mask, sizeof (au_mask_t)))
3153             return;
3154         au_uwrite(au_to_arg32(
3155             2, "setamask:as_success", (uint32_t)mask.as_success));
3156         au_uwrite(au_to_arg32(
3157             2, "setamask:as_failure", (uint32_t)mask.as_failure));
3158         break;
3159     case AUE_AUDITON_SETKMASK:
3160         if (copyin((caddr_t)a2, &mask, sizeof (au_mask_t)))
3161             return;
3162         au_uwrite(au_to_arg32(
3163             2, "setkmask:as_success", (uint32_t)mask.as_success));
3164         au_uwrite(au_to_arg32(
3165             2, "setkmask:as_failure", (uint32_t)mask.as_failure));
3166         break;
3167     case AUE_AUDITON_SPOLICY:
3168         if (copyin((caddr_t)a2, &policy, sizeof (int)))
3169             return;
3170         au_uwrite(au_to_arg32(3, "setpolicy", (uint32_t)policy));
3171         break;
3172     case AUE_AUDITON_SQCTRL: {
3173         STRUCT_DECL(au_qctrl, qctrl);
3174         model_t model;

```

```

3176     model = get_udatamodel();
3177     STRUCT_INIT(qctrl, model);
3178     if (copyin((caddr_t)a2, STRUCT_BUF(qctrl), STRUCT_SIZE(qctrl)))
3179         return;
3180     if (model == DATAMODEL_ILP32) {
3181         au_uwrite(au_to_arg32(
3182             3, "setqctrl:aq_hiwater",
3183             (uint32_t)STRUCT_FGET(qctrl, aq_hiwater)));
3184         au_uwrite(au_to_arg32(
3185             3, "setqctrl:aq_lowater",
3186             (uint32_t)STRUCT_FGET(qctrl, aq_lowater)));
3187         au_uwrite(au_to_arg32(
3188             3, "setqctrl:aq_bufsz",
3189             (uint32_t)STRUCT_FGET(qctrl, aq_bufsz)));
3190         au_uwrite(au_to_arg32(
3191             3, "setqctrl:aq_delay",
3192             (uint32_t)STRUCT_FGET(qctrl, aq_delay)));
3193     } else {
3194         au_uwrite(au_to_arg64(
3195             3, "setqctrl:aq_hiwater",
3196             (uint64_t)STRUCT_FGET(qctrl, aq_hiwater)));
3197         au_uwrite(au_to_arg64(
3198             3, "setqctrl:aq_lowater",
3199             (uint64_t)STRUCT_FGET(qctrl, aq_lowater)));
3200         au_uwrite(au_to_arg64(
3201             3, "setqctrl:aq_bufsz",
3202             (uint64_t)STRUCT_FGET(qctrl, aq_bufsz)));
3203         au_uwrite(au_to_arg64(
3204             3, "setqctrl:aq_delay",
3205             (uint64_t)STRUCT_FGET(qctrl, aq_delay)));
3206     }
3207     break;
3208 }
3209 case AUE_AUDITON_SETUMASK:
3210     STRUCT_INIT(ainfo, get_udatamodel());
3211     if (copyin((caddr_t)uap->a2, STRUCT_BUF(ainfo),
3212         STRUCT_SIZE(ainfo))) {
3213         return;
3214     }
3215     au_uwrite(au_to_arg32(3, "setumask:as_success",
3216         (uint32_t)STRUCT_FGET(ainfo, ai_mask.as_success)));
3217     au_uwrite(au_to_arg32(3, "setumask:as_failure",
3218         (uint32_t)STRUCT_FGET(ainfo, ai_mask.as_failure)));
3219     break;
3220 case AUE_AUDITON_SETSMASK:
3221     STRUCT_INIT(ainfo, get_udatamodel());
3222     if (copyin((caddr_t)uap->a2, STRUCT_BUF(ainfo),
3223         STRUCT_SIZE(ainfo))) {
3224         return;
3225     }
3226     au_uwrite(au_to_arg32(3, "setsmask:as_success",
3227         (uint32_t)STRUCT_FGET(ainfo, ai_mask.as_success)));
3228     au_uwrite(au_to_arg32(3, "setsmask:as_failure",
3229         (uint32_t)STRUCT_FGET(ainfo, ai_mask.as_failure)));
3230     break;
3231 case AUE_AUDITON_SETCOND:
3232     if (copyin((caddr_t)a2, &auditstate, sizeof (int)))
3233         return;
3234     au_uwrite(au_to_arg32(3, "setcond", (uint32_t)auditstate));
3235     break;
3236 case AUE_AUDITON_SETCLASS:
3237     if (copyin((caddr_t)a2, &event, sizeof (au_evclass_map_t)))
3238         return;
3239     au_uwrite(au_to_arg32(
3240         2, "setclass:ec_event", (uint32_t)event.ec_number));
3241     au_uwrite(au_to_arg32(

```

```

3242         3, "setclass:ec_class", (uint32_t)event.ec_class));
3243     break;
3244 case AUE_GETAUDIT:
3245 case AUE_GETAUDIT:
3246 case AUE_GETAUDIT_ADDR:
3247 case AUE_AUDIT:
3248 case AUE_AUDITON_GPOLICY:
3249 case AUE_AUDITON_GQCTRL:
3250 case AUE_AUDITON_GETAMASK:
3251 case AUE_AUDITON_GETKMASK:
3252 case AUE_AUDITON_GETCWD:
3253 case AUE_AUDITON_GETCAR:
3254 case AUE_AUDITON_GETSTAT:
3255 case AUE_AUDITON_SETSTAT:
3256 case AUE_AUDITON_GETCOND:
3257 case AUE_AUDITON_GETCLASS:
3258     break;
3259 default:
3260     break;
3261 }
3262 } /* AUS_AUDITSYS */
3263
3266 /* only audit privileged operations for systeminfo(2) system call */
3267 static au_event_t
3268 aui_sysinfo(au_event_t e)
3269 {
3270     klpw_t *clwp = ttolwp(curthread);
3271     uint32_t command;
3272
3273     struct a {
3274         long    command;
3275         long    buf; /* char * */
3276         long    count;
3277     } *uap = (struct a *)clwp->lwp_ap;
3278
3279     command = (uint32_t)uap->command;
3280
3281     switch (command) {
3282     case SI_SET_HOSTNAME:
3283     case SI_SET_SRPC_DOMAIN:
3284         e = (au_event_t)AUE_SYSINFO;
3285         break;
3286     default:
3287         e = (au_event_t)AUE_NULL;
3288         break;
3289     }
3290     return (e);
3291 }
3292
3293 /*ARGSUSED*/
3294 static void
3295 aus_sysinfo(struct t_audit_data *tad)
3296 {
3297     klpw_t *clwp = ttolwp(curthread);
3298     uint32_t command;
3299     size_t len, maxlen;
3300     char *name;
3301     uintptr_t buf;
3302
3303     struct a {
3304         long    command;
3305         long    buf; /* char * */
3306         long    count;
3307     } *uap = (struct a *)clwp->lwp_ap;

```

```

3309     command = (uint32_t)uap->command;
3310     buf = (uintptr_t)uap->buf;

3312     au_uwrite(au_to_arg32(1, "cmd", command));

3314     switch (command) {
3315     case SI_SET_HOSTNAME:
3316     {
3317         if (secpolicy_sys_config(CRED(), B_TRUE) != 0)
3318             return;

3320         maxlen = SYS_NMLN;
3321         name = kmem_alloc(maxlen, KM_SLEEP);
3322         if (copyinstr((caddr_t)buf, name, SYS_NMLN, &len))
3323             break;

3325         /*
3326          * Must be non-NULL string and string
3327          * must be less than SYS_NMLN chars.
3328          */
3329         if (len < 2 || (len == SYS_NMLN && name[SYS_NMLN - 1] != '\0'))
3330             break;

3332         au_uwrite(au_to_text(name));
3333         break;
3334     }

3336     case SI_SET_SRPC_DOMAIN:
3337     {
3338         if (secpolicy_sys_config(CRED(), B_TRUE) != 0)
3339             return;

3341         maxlen = SYS_NMLN;
3342         name = kmem_alloc(maxlen, KM_SLEEP);
3343         if (copyinstr((caddr_t)buf, name, SYS_NMLN, &len))
3344             break;

3346         /*
3347          * If string passed in is longer than length
3348          * allowed for domain name, fail.
3349          */
3350         if (len == SYS_NMLN && name[SYS_NMLN - 1] != '\0')
3351             break;

3353         au_uwrite(au_to_text(name));
3354         break;
3355     }

3357     default:
3358         return;
3359     }

3361     kmem_free(name, maxlen);
3362 }

3364 static au_event_t
3365 aui_modctl(au_event_t e)
3366 {
3367     klwp_t *clwp = ttolwp(curthread);
3368     uint_t cmd;

3370     struct a {
3371         long    cmd;
3372     } *uap = (struct a *)clwp->lwp_ap;

```

```

3374     cmd = (uint_t)uap->cmd;

3376     switch (cmd) {
3377     case MODLOAD:
3378         e = AUE_MODLOAD;
3379         break;
3380     case MODUNLOAD:
3381         e = AUE_MODUNLOAD;
3382         break;
3383     case MODADDMAJBIND:
3384         e = AUE_MODADDMAJ;
3385         break;
3386     case MODSETDEVPOLICY:
3387         e = AUE_MODDEVPLCY;
3388         break;
3389     case MODALLOCPRIV:
3390         e = AUE_MODADDPRIV;
3391         break;
3392     default:
3393         e = AUE_NULL;
3394         break;
3395     }
3396     return (e);
3397 }

3400 /*ARGSUSED*/
3401 static void
3402 aus_modctl(struct t_audit_data *tad)
3403 {
3404     klwp_t *clwp = ttolwp(curthread);
3405     void *a = clwp->lwp_ap;
3406     uint_t use_path;

3408     switch (tad->tad_event) {
3409     case AUE_MODLOAD: {
3410         typedef struct {
3411             long    cmd;
3412             long    use_path;
3413             long    filename;
3414         } modloada_t;

3416         char *filenamep;
3417         uintptr_t fname;
3418         extern char *default_path;

3420         fname = (uintptr_t)((modloada_t *)a)->filename;
3421         use_path = (uint_t)((modloada_t *)a)->use_path;

3423         /* space to hold path */
3424         filenamep = kmem_alloc(MOD_MAXPATH, KM_SLEEP);
3425         /* get string */
3426         if (copyinstr((caddr_t)fname, filenamep, MOD_MAXPATH, 0)) {
3427             /* free allocated path */
3428             kmem_free(filenamep, MOD_MAXPATH);
3429             return;
3430         }
3431         /* ensure it's null terminated */
3432         filenamep[MOD_MAXPATH - 1] = 0;

3434         if (use_path)
3435             au_uwrite(au_to_text(default_path));
3436         au_uwrite(au_to_text(filenamep));

3438         /* release temporary memory */
3439         kmem_free(filenamep, MOD_MAXPATH);

```

```

3440         break;
3441     }
3442     case AUE_MODUNLOAD: {
3443         typedef struct {
3444             long    cmd;
3445             long    id;
3446         } modunloada_t;
3448         uint32_t id = (uint32_t)((modunloada_t *)a)->id;
3450         au_uwrite(au_to_arg32(1, "id", id));
3451         break;
3452     }
3453     case AUE_MODADDMAJ: {
3454         STRUCT_DECL(modconfig, mc);
3455         typedef struct {
3456             long    cmd;
3457             long    subcmd;
3458             long    data;          /* int * */
3459         } modconfiga_t;
3461         STRUCT_DECL(aliases, alias);
3462         caddr_t ap;
3463         int i, num_aliases;
3464         char *drvname, *mc_drvname;
3465         char *name;
3466         extern char *ddi_major_to_name(major_t);
3467         model_t model;
3469         uintptr_t data = (uintptr_t)((modconfiga_t *)a)->data;
3471         model = get_udatamodel();
3472         STRUCT_INIT(mc, model);
3473         /* sanitize buffer */
3474         bzero((caddr_t)STRUCT_BUF(mc), STRUCT_SIZE(mc));
3475         /* get user arguments */
3476         if (copyin((caddr_t)data, (caddr_t)STRUCT_BUF(mc),
3477             STRUCT_SIZE(mc)) != 0)
3478             return;
3480         mc_drvname = STRUCT_FGET(mc, drvname);
3481         if ((drvname = ddi_major_to_name(
3482             (major_t)STRUCT_FGET(mc, major))) != NULL &&
3483             strcmp(drvname, mc_drvname, MAXMODCONFNAME) != 0) {
3484             /* safety */
3485             if (mc_drvname[0] != '\0') {
3486                 mc_drvname[MAXMODCONFNAME-1] = '\0';
3487                 au_uwrite(au_to_text(mc_drvname));
3488             }
3489             /* drvname != NULL from test above */
3490             au_uwrite(au_to_text(drvname));
3491             return;
3492         }
3494         if (mc_drvname[0] != '\0') {
3495             /* safety */
3496             mc_drvname[MAXMODCONFNAME-1] = '\0';
3497             au_uwrite(au_to_text(mc_drvname));
3498         } else
3499             au_uwrite(au_to_text("no drvname"));
3501         num_aliases = STRUCT_FGET(mc, num_aliases);
3502         au_uwrite(au_to_arg32(5, "", (uint32_t)num_aliases));
3503         ap = (caddr_t)STRUCT_FGETP(mc, ap);
3504         name = kmem_alloc(MAXMODCONFNAME, KM_SLEEP);
3505         STRUCT_INIT(alias, model);

```

```

3506         for (i = 0; i < num_aliases; i++) {
3507             bzero((caddr_t)STRUCT_BUF(alias),
3508                 STRUCT_SIZE(alias));
3509             if (copyin((caddr_t)ap, (caddr_t)STRUCT_BUF(alias),
3510                 STRUCT_SIZE(alias)) != 0)
3511                 break;
3512             if (copyinstr(STRUCT_FGETP(alias, a_name), name,
3513                 MAXMODCONFNAME, NULL) != 0) {
3514                 break;
3515             }
3517             au_uwrite(au_to_text(name));
3518             ap = (caddr_t)STRUCT_FGETP(alias, a_next);
3519         }
3520         kmem_free(name, MAXMODCONFNAME);
3521         break;
3522     }
3523     default:
3524         break;
3525     }
3526 }
3529 /*ARGSUSED*/
3530 static void
3531 auf_accept(
3532     struct t_audit_data *tad,
3533     int error,
3534     rval_t *rval)
3535 {
3536     uint32_t scid;
3537     uint32_t sy_flags;
3538     int fd;
3539     struct sonode *so;
3540     char so_laddr[sizeof (struct sockaddr_in6)];
3541     char so_faddr[sizeof (struct sockaddr_in6)];
3542     int err;
3543     short so_family, so_type;
3544     int add_sock_token = 0;
3546     /* need to determine type of executing binary */
3547     scid = tad->tad_scid;
3548     #ifndef _SYSCALL32_IMPL
3549     if (lwp_getdatamodel(ttolwp(curthread)) == DATAMODEL_NATIVE)
3550         sy_flags = sysent[scid].sy_flags & SE_RVAL_MASK;
3551     else
3552         sy_flags = sysent32[scid].sy_flags & SE_RVAL_MASK;
3553     #else
3554     sy_flags = sysent[scid].sy_flags & SE_RVAL_MASK;
3555     #endif
3556     switch (sy_flags) {
3557     case SE_32RVAL1:
3558         /* FALLTHRU */
3559     case SE_32RVAL2|SE_32RVAL1:
3560         fd = rval->r_val1;
3561         break;
3562     case SE_64RVAL:
3563         fd = (int)rval->r_vals;
3564         break;
3565     default:
3566         /*
3567          * should never happen, seems to be an internal error
3568          * in sysent => no fd, nothing to audit here, returning
3569          */
3570         return;
3571     }

```

```

3573     if (error) {
3574         /* can't trust socket contents. Just return */
3575         au_uwrite(au_to_arg32(1, "so", (uint32_t)fd));
3576         return;
3577     }
3579     if ((so = getsonode(fd, &err, NULL)) == NULL) {
3580         /*
3581          * not security relevant if doing a accept from non socket
3582          * so no extra tokens. Should probably turn off audit record
3583          * generation here.
3584          */
3585         return;
3586     }
3588     so_family = so->so_family;
3589     so_type = so->so_type;
3591     switch (so_family) {
3592     case AF_INET:
3593     case AF_INET6:
3594         /*
3595          * XXX - what about other socket types for AF_INET (e.g. DGRAM)
3596          */
3597         if (so->so_type == SOCK_STREAM) {
3598             socklen_t len;
3600             bzero((void *)so_laddr, sizeof (so_laddr));
3601             bzero((void *)so_faddr, sizeof (so_faddr));
3603             len = sizeof (so_laddr);
3604             (void) socket_getsockname(so,
3605                                     (struct sockaddr *)so_laddr, &len, CRED());
3606             len = sizeof (so_faddr);
3607             (void) socket_getpeername(so,
3608                                     (struct sockaddr *)so_faddr, &len, B_FALSE, CRED());
3610             add_sock_token = 1;
3611         }
3612         break;
3614     default:
3615         /* AF_UNIX, AF_ROUTE, AF_KEY do not support accept */
3616         break;
3617     }
3619     releasef(fd);
3621     au_uwrite(au_to_arg32(1, "so", (uint32_t)fd));
3623     if (add_sock_token == 0) {
3624         au_uwrite(au_to_arg32(0, "family", (uint32_t)(so_family)));
3625         au_uwrite(au_to_arg32(0, "type", (uint32_t)(so_type)));
3626         return;
3627     }
3629     au_uwrite(au_to_socket_ex(so_family, so_type, so_laddr, so_faddr));
3631 }
3633 /*ARGSUSED*/
3634 static void
3635 auf_bind(struct t_audit_data *tad, int error, rval_t *rvp)
3636 {
3637     struct a {

```

```

3638         long    fd;
3639         long    addr;
3640         long    len;
3641     } *uap = (struct a *)ttolwp(curthread)->lwp_ap;
3643     struct sonode *so;
3644     char so_laddr[sizeof (struct sockaddr_in6)];
3645     char so_faddr[sizeof (struct sockaddr_in6)];
3646     int err, fd;
3647     socklen_t len;
3648     short so_family, so_type;
3649     int add_sock_token = 0;
3651     fd = (int)uap->fd;
3653     /*
3654      * bind failed, then nothing extra to add to audit record.
3655      */
3656     if (error) {
3657         au_uwrite(au_to_arg32(1, "so", (uint32_t)fd));
3658         /* XXX may want to add failed address some day */
3659         return;
3660     }
3662     if ((so = getsonode(fd, &err, NULL)) == NULL) {
3663         /*
3664          * not security relevant if doing a bind from non socket
3665          * so no extra tokens. Should probably turn off audit record
3666          * generation here.
3667          */
3668         return;
3669     }
3671     so_family = so->so_family;
3672     so_type = so->so_type;
3674     switch (so_family) {
3675     case AF_INET:
3676     case AF_INET6:
3678         bzero(so_faddr, sizeof (so_faddr));
3679         len = sizeof (so_faddr);
3681         (void) socket_getpeername(so,
3682                                 (struct sockaddr *)so_faddr, &len, B_FALSE, CRED());
3683         add_sock_token = 1;
3685         break;
3687     case AF_UNIX:
3688         /* token added by lookup */
3689         break;
3690     default:
3691         /* AF_ROUTE, AF_KEY do not support accept */
3692         break;
3693     }
3695     releasef(fd);
3697     au_uwrite(au_to_arg32(1, "so", (uint32_t)fd));
3699     if (add_sock_token == 0) {
3700         au_uwrite(au_to_arg32(1, "family", (uint32_t)(so_family)));
3701         au_uwrite(au_to_arg32(1, "type", (uint32_t)(so_type)));
3702         return;
3703     }

```

```

3705     au_uwrite(au_to_socket_ex(so_family, so_type, so_laddr, so_faddr));
3707 }

3709 /*ARGSUSED*/
3710 static void
3711 auf_connect(struct t_audit_data *tad, int error, rval_t *rval)
3712 {
3713     struct a {
3714         long    fd;
3715         long    addr;
3716         long    len;
3717     } *uap = (struct a *)ttolwp(curthread)->lwp_ap;

3719     struct sonode *so;
3720     char so_laddr[sizeof (struct sockaddr_in6)];
3721     char so_faddr[sizeof (struct sockaddr_in6)];
3722     int err, fd;
3723     socklen_t len;
3724     short so_family, so_type;
3725     int add_sock_token = 0;

3727     fd = (int)uap->fd;

3730     if ((so = getsonode(fd, &err, NULL)) == NULL) {
3731         /*
3732          * not security relevant if doing a connect from non socket
3733          * so no extra tokens. Should probably turn off audit record
3734          * generation here.
3735          */
3736         return;
3737     }

3739     so_family = so->so_family;
3740     so_type = so->so_type;

3742     switch (so_family) {
3743     case AF_INET:
3744     case AF_INET6:

3746         bzero(so_laddr, sizeof (so_laddr));
3747         bzero(so_faddr, sizeof (so_faddr));

3749         len = sizeof (so_laddr);
3750         (void) socket_getsockname(so, (struct sockaddr *)so_laddr,
3751             &len, CRED());
3752         if (error) {
3753             if (uap->addr == NULL)
3754                 break;
3755             if (uap->len <= 0)
3756                 break;
3757             len = min(uap->len, sizeof (so_faddr));
3758             if (copyin((caddr_t)(uap->addr), so_faddr, len) != 0)
3759                 break;
3760 #ifdef NOTYET
3761             au_uwrite(au_to_data(AUP_HEX, AUR_CHAR, len, so_faddr));
3762 #endif
3763         } else {
3764             /* sanity check on length */
3765             len = sizeof (so_faddr);
3766             (void) socket_getpeername(so,
3767                 (struct sockaddr *)so_faddr, &len, B_FALSE, CRED());
3768         }

```

```

3770         add_sock_token = 1;
3772         break;

3774     case AF_UNIX:
3775         /* does a lookup on name */
3776         break;

3778     default:
3779         /* AF_ROUTE, AF_KEY do not support accept */
3780         break;
3781     }

3783     releasef(fd);

3785     au_uwrite(au_to_arg32(1, "so", (uint32_t)fd));

3787     if (add_sock_token == 0) {
3788         au_uwrite(au_to_arg32(1, "family", (uint32_t)(so_family)));
3789         au_uwrite(au_to_arg32(1, "type", (uint32_t)(so_type)));
3790         return;
3791     }

3793     au_uwrite(au_to_socket_ex(so_family, so_type, so_laddr, so_faddr));
3795 }

3797 /*ARGSUSED*/
3798 static void
3799 aus_shutdown(struct t_audit_data *tad)
3800 {
3801     struct a {
3802         long    fd;
3803         long    how;
3804     } *uap = (struct a *)ttolwp(curthread)->lwp_ap;

3806     struct sonode *so;
3807     char so_laddr[sizeof (struct sockaddr_in6)];
3808     char so_faddr[sizeof (struct sockaddr_in6)];
3809     int err, fd;
3810     socklen_t len;
3811     short so_family, so_type;
3812     int add_sock_token = 0;
3813     file_t *fp;
3814     struct f_audit_data *fad;
3815     /* unix domain sockets */
3816     /* unix domain sockets */

3818     fd = (int)uap->fd;

3818     if ((so = getsonode(fd, &err, &fp)) == NULL) {
3819         /*
3820          * not security relevant if doing a shutdown using non socket
3821          * so no extra tokens. Should probably turn off audit record
3822          * generation here.
3823          */
3824         return;
3825     }

3827     so_family = so->so_family;
3828     so_type = so->so_type;

3830     switch (so_family) {
3831     case AF_INET:
3832     case AF_INET6:

3834         bzero(so_laddr, sizeof (so_laddr));
3835         bzero(so_faddr, sizeof (so_faddr));

```



```

3837         len = sizeof (so_laddr);
3838         (void) socket_getsockname(so,
3839             (struct sockaddr *)so_laddr, &len, CRED());
3840         len = sizeof (so_faddr);
3841         (void) socket_getpeername(so,
3842             (struct sockaddr *)so_faddr, &len, B_FALSE, CRED());

3844         add_sock_token = 1;

3846         break;

3848     case AF_UNIX:

3850         /* get path from file struct here */
3851         fad = F2A(fp);
3852         ASSERT(fad);

3854         if (fad->fad_aupath != NULL) {
3855             au_uwrite(au_to_path(fad->fad_aupath));
3856         } else {
3857             au_uwrite(au_to_arg32(1, "no path: fd", fd));
3858         }

3860         audit_attributes(fp->f_vnode);

3862         break;

3864     default:
3865         /*
3866          * AF_KEY and AF_ROUTE support shutdown. No socket token
3867          * added.
3868          */
3869         break;
3870     }

3872     releasef(fd);

3874     au_uwrite(au_to_arg32(1, "so", (uint32_t)fd));

3876     if (add_sock_token == 0) {
3877         au_uwrite(au_to_arg32(1, "family", (uint32_t)(so_family)));
3878         au_uwrite(au_to_arg32(1, "type", (uint32_t)(so_type)));
3879         au_uwrite(au_to_arg32(2, "how", (uint32_t)(uap->how)));
3880         return;
3881     }

3883     au_uwrite(au_to_arg32(2, "how", (uint32_t)(uap->how)));

3885     au_uwrite(au_to_socket_ex(so_family, so_type, so_laddr, so_faddr));

3887 }

3889 /*ARGSUSED*/
3890 static void
3891 auf_setsockopt(struct t_audit_data *tad, int error, rval_t *rval)
3892 {
3893     struct a {
3894         long    fd;
3895         long    level;
3896         long    optname;
3897         long    *optval;
3898         long    optlen;
3899     } *uap = (struct a *)ttolwp(curthread)->lwp_ap;

3901     struct sonode *so;

```

```

3902     char so_laddr[sizeof (struct sockaddr_in6)];
3903     char so_faddr[sizeof (struct sockaddr_in6)];
3904     char    val[AU_BUFSIZE];
3905     int     err, fd;
3906     socklen_t    len;
3907     short so_family, so_type;
3908     int     add_sock_token = 0;
3909     file_t *fp;
3910     struct f_audit_data *fad;
3911
3912     fd = (int)uap->fd;

3914     if (error) {
3915         au_uwrite(au_to_arg32(1, "so", (uint32_t)fd));
3916         au_uwrite(au_to_arg32(2, "level", (uint32_t)uap->level));
3917         /* XXX may want to include other arguments */
3918         return;
3919     }

3921     if ((so = getsonode(fd, &err, &fp)) == NULL) {
3922         /*
3923          * not security relevant if doing a setsockopt from non socket
3924          * so no extra tokens. Should probably turn off audit record
3925          * generation here.
3926          */
3927         return;
3928     }

3930     so_family = so->so_family;
3931     so_type = so->so_type;

3933     switch (so_family) {
3934     case AF_INET:
3935     case AF_INET6:
3936         bzero((void *)so_laddr, sizeof (so_laddr));
3937         bzero((void *)so_faddr, sizeof (so_faddr));

3939         /* get local and foreign addresses */
3940         len = sizeof (so_laddr);
3941         (void) socket_getsockname(so, (struct sockaddr *)so_laddr,
3942             &len, CRED());
3943         len = sizeof (so_faddr);
3944         (void) socket_getpeername(so, (struct sockaddr *)so_faddr,
3945             &len, B_FALSE, CRED());

3947         add_sock_token = 1;

3949         break;

3951     case AF_UNIX:

3953         /* get path from file struct here */
3954         fad = F2A(fp);
3955         ASSERT(fad);

3957         if (fad->fad_aupath != NULL) {
3958             au_uwrite(au_to_path(fad->fad_aupath));
3959         } else {
3960             au_uwrite(au_to_arg32(1, "no path: fd", fd));
3961         }

3963         audit_attributes(fp->f_vnode);

3965         break;

3967     default:

```

```

3968     /*
3969     * AF_KEY and AF_ROUTE support setsockopt. No socket token
3970     * added.
3971     */
3972     break;
3973 }
3975 releasef(fd);
3977 au_uwrite(au_to_arg32(1, "so", (uint32_t)fd));
3979 if (add_sock_token == 0) {
3980     au_uwrite(au_to_arg32(1, "family", (uint32_t)(so_family)));
3981     au_uwrite(au_to_arg32(1, "type", (uint32_t)(so_type)));
3982 }
3983 au_uwrite(au_to_arg32(2, "level", (uint32_t)(uap->level)));
3984 au_uwrite(au_to_arg32(3, "optname", (uint32_t)(uap->optname)));
3986 bzero(val, sizeof (val));
3987 len = min(uap->optlen, sizeof (val));
3988 if ((len > 0) &&
3989     (copyin((caddr_t)(uap->optval), (caddr_t)val, len) == 0)) {
3990     au_uwrite(au_to_arg32(5, "optlen", (uint32_t)(uap->optlen)));
3991     au_uwrite(au_to_data(AUP_HEX, AUR_BYTE, len, val));
3992 }
3994 if (add_sock_token == 0)
3995     return;
3997 au_uwrite(au_to_socket_ex(so_family, so_type, so_laddr, so_faddr));
3999 }
4001 /*ARGSUSED*/
4002 static void
4003 aus_sockconfig(tad)
4004     struct t_audit_data *tad;
4005 {
4006     struct a {
4007         long    cmd;
4008         long    arg1;
4009         long    arg2;
4010         long    arg3;
4011         long    arg4;
4012     } *uap = (struct a *)ttolwp(curthread)->lwp_ap;
4014     char    *buf;
4015     int     buflen;
4016     size_t  size;
4018     au_uwrite(au_to_arg32(1, "cmd", (uint_t)uap->cmd));
4019     switch (uap->cmd) {
4020     case SOCKCONFIG_ADD_SOCK:
4021     case SOCKCONFIG_REMOVE_SOCK:
4022         au_uwrite(au_to_arg32(2, "domain", (uint32_t)uap->arg1));
4023         au_uwrite(au_to_arg32(3, "type", (uint32_t)uap->arg2));
4024         au_uwrite(au_to_arg32(4, "protocol", (uint32_t)uap->arg3));
4026         if (uap->arg4 == 0) {
4027             au_uwrite(au_to_arg32(5, "devpath", (uint32_t)0));
4028         } else {
4029             buflen = MAXPATHLEN + 1;
4030             buf = kmem_alloc(buflen, KM_SLEEP);
4031             if (copyinstr((caddr_t)uap->arg4, buf, buflen,
4032                 &size)) {
4033                 kmem_free(buf, buflen);

```

```

4034         return;
4035     }
4037     if (size > MAXPATHLEN) {
4038         kmem_free(buf, buflen);
4039         return;
4040     }
4042     au_uwrite(au_to_text(buf));
4043     kmem_free(buf, buflen);
4044 }
4045 break;
4046 case SOCKCONFIG_ADD_FILTER:
4047 case SOCKCONFIG_REMOVE_FILTER:
4048     buflen = FILNAME_MAX;
4049     buf = kmem_alloc(buflen, KM_SLEEP);
4051     if (copyinstr((caddr_t)uap->arg1, buf, buflen, &size)) {
4052         kmem_free(buf, buflen);
4053         return;
4054     }
4056     au_uwrite(au_to_text(buf));
4057     kmem_free(buf, buflen);
4058     break;
4059 default:
4060     break;
4061 }
4062 }
4064 /*
4065 * only audit recvmmsg when the system call represents the creation of a new
4066 * circuit. This effectively occurs for all UDP packets and may occur for
4067 * special TCP situations where the local host has not set a local address
4068 * in the socket structure.
4069 */
4070 /*ARGSUSED*/
4071 static void
4072 auf_recvmmsg(
4073     struct t_audit_data *tad,
4074     int error,
4075     rval_t *rvp)
4076 {
4077     struct a {
4078         long    fd;
4079         long    msg; /* struct msghdr */
4080         long    flags;
4081     } *uap = (struct a *)ttolwp(curthread)->lwp_ap;
4083     struct sonode *so;
4084     STRUCT_DECL(msghdr, msg);
4085     caddr_t msg_name;
4086     socklen_t msg_namelen;
4087     int fd;
4088     int err;
4089     char so_laddr[sizeof (struct sockaddr_in6)];
4090     char so_faddr[sizeof (struct sockaddr_in6)];
4091     socklen_t len;
4092     file_t *fp; /* unix domain sockets */
4093     struct f_audit_data *fad; /* unix domain sockets */
4094     short so_family, so_type;
4095     int add_sock_token = 0;
4096     au_kcontext_t *kctx = GET_KCTX_PZ;
4098     fd = (int)uap->fd;

```

```

4100  /* bail if an error */
4101  if (error) {
4102      au_uwrite(au_to_arg32(1, "so", (uint32_t)fd));
4103      au_uwrite(au_to_arg32(3, "flags", (uint32_t)(uap->flags));
4104      return;
4105  }

4107  if ((so = getsonode(fd, &err, &fp)) == NULL) {
4108      /*
4109       * not security relevant if doing a recvmmsg from non socket
4110       * so no extra tokens. Should probably turn off audit record
4111       * generation here.
4112       */
4113      return;
4114  }

4116  so_family = so->so_family;
4117  so_type = so->so_type;

4119  /*
4120   * only putout SOCKET_EX token if INET/INET6 family.
4121   * XXX - what do we do about other families?
4122   */

4124  switch (so_family) {
4125  case AF_INET:
4126

4128      /*
4129       * if datagram type socket, then just use what is in
4130       * socket structure for local address.
4131       * XXX - what do we do for other types?
4132       */
4133      if ((so->so_type == SOCK_DGRAM) ||
4134          (so->so_type == SOCK_RAW)) {
4135          add_sock_token = 1;

4137          bzero((void *)so_laddr, sizeof (so_laddr));
4138          bzero((void *)so_faddr, sizeof (so_faddr));

4140          /* get local address */
4141          len = sizeof (so_laddr);
4142          (void) socket_getsockname(so,
4143              (struct sockaddr *)so_laddr, &len, CRED());

4145          /* get peer address */
4146          STRUCT_INIT(msg, get_udatamodel());

4148          if (copyin((caddr_t)(uap->msg),
4149              (caddr_t)STRUCT_BUF(msg), STRUCT_SIZE(msg)) != 0) {
4150              break;
4151          }
4152          msg_name = (caddr_t)STRUCT_FGETP(msg, msg_name);
4153          if (msg_name == NULL) {
4154              break;
4155          }

4157          /* length is value from recvmmsg - sanity check */
4158          msg_namelen = (socklen_t)STRUCT_FGET(msg, msg_namelen);
4159          if (msg_namelen == 0) {
4160              break;
4161          }
4162          if (copyin(msg_name, so_faddr,
4163              sizeof (so_faddr)) != 0) {
4164              break;
4165          }

```

```

4167  } else if (so->so_type == SOCK_STREAM) {

4169      /* get path from file struct here */
4170      fad = F2A(fp);
4171      ASSERT(fad);

4173      /*
4174       * already processed this file for read attempt
4175       */
4176      if (fad->fad_flags & FAD_READ) {
4177          /* don't want to audit every recvmmsg attempt */
4178          tad->tad_flag = 0;
4179          /* free any residual audit data */
4180          au_close(kctx, &(u_ad), 0, 0, 0, NULL);
4181          releasef(fd);
4182          return;
4183      }
4184      /*
4185       * mark things so we know what happened and don't
4186       * repeat things
4187       */
4188      fad->fad_flags |= FAD_READ;

4190      bzero((void *)so_laddr, sizeof (so_laddr));
4191      bzero((void *)so_faddr, sizeof (so_faddr));

4193      /* get local and foreign addresses */
4194      len = sizeof (so_laddr);
4195      (void) socket_getsockname(so,
4196          (struct sockaddr *)so_laddr, &len, CRED());
4197      len = sizeof (so_faddr);
4198      (void) socket_getpeername(so,
4199          (struct sockaddr *)so_faddr, &len, B_FALSE, CRED());

4201      add_sock_token = 1;
4202  }

4204      /* XXX - what about SOCK_RDM/SOCK_SEQPACKET ??? */

4206      break;

4208  case AF_UNIX:
4209      /*
4210       * first check if this is first time through. Too much
4211       * duplicate code to put this in an aui_ routine.
4212       */

4214      /* get path from file struct here */
4215      fad = F2A(fp);
4216      ASSERT(fad);

4218      /*
4219       * already processed this file for read attempt
4220       */
4221      if (fad->fad_flags & FAD_READ) {
4222          releasef(fd);
4223          /* don't want to audit every recvmmsg attempt */
4224          tad->tad_flag = 0;
4225          /* free any residual audit data */
4226          au_close(kctx, &(u_ad), 0, 0, 0, NULL);
4227          return;
4228      }
4229      /*
4230       * mark things so we know what happened and don't
4231       * repeat things

```

```

4232     */
4233     fad->fad_flags |= FAD_READ;

4235     if (fad->fad_aupath != NULL) {
4236         au_uwrite(au_to_path(fad->fad_aupath));
4237     } else {
4238         au_uwrite(au_to_arg32(1, "no path: fd", fd));
4239     }

4241     audit_attributes(fp->f_vnode);

4243     releasef(fd);

4245     return;

4247 default:
4248     break;

4250 }

4252 releasef(fd);

4254 au_uwrite(au_to_arg32(1, "so", (uint32_t)fd));

4256 if (add_sock_token == 0) {
4257     au_uwrite(au_to_arg32(1, "family", (uint32_t)so_family));
4258     au_uwrite(au_to_arg32(1, "type", (uint32_t)so_type));
4259     au_uwrite(au_to_arg32(3, "flags", (uint32_t)(uap->flags)));
4260     return;
4261 }

4263 au_uwrite(au_to_arg32(3, "flags", (uint32_t)(uap->flags)));

4265 au_uwrite(au_to_socket_ex(so_family, so_type, so_laddr, so_faddr));

4267 }

4269 /*ARGSUSED*/
4270 static void
4271 auf_recvfrom(
4272     struct t_audit_data *tad,
4273     int error,
4274     rval_t *rvp)
4275 {
4277     struct a {
4278         long    fd;
4279         long    msg;    /* char */
4280         long    len;
4281         long    flags;
4282         long    from;   /* struct sockaddr */
4283         long    fromlen;
4284     } *uap = (struct a *)ttolwp(curthread)->lwp_ap;

4286     socklen_t    fromlen;
4287     struct sonode *so;
4288     char so_laddr[sizeof (struct sockaddr_in6)];
4289     char so_faddr[sizeof (struct sockaddr_in6)];
4290     int    fd;
4291     short so_family, so_type;
4292     int add_sock_token = 0;
4293     socklen_t len;
4294     int err;
4295     struct file *fp;
4296     struct f_audit_data *fad;    /* unix domain sockets */
4297     au_kcontext_t *kctx = GET_KCTX_PZ;

```

```

4299     fd = (int)uap->fd;

4301     /* bail if an error */
4302     if (error) {
4303         au_uwrite(au_to_arg32(1, "so", (uint32_t)fd));
4304         au_uwrite(au_to_arg32(3, "flags", (uint32_t)(uap->flags)));
4305         return;
4306     }

4308     if ((so = getsonode(fd, &err, &fp)) == NULL) {
4309         /*
4310          * not security relevant if doing a recvmmsg from non socket
4311          * so no extra tokens. Should probably turn off audit record
4312          * generation here.
4313          */
4314         return;
4315     }

4317     so_family = so->so_family;
4318     so_type = so->so_type;

4320     /*
4321     * only putout SOCKET_EX token if INET/INET6 family.
4322     * XXX - what do we do about other families?
4323     */

4325     switch (so_family) {
4326     case AF_INET:
4327     case AF_INET6:

4329         /*
4330          * if datagram type socket, then just use what is in
4331          * socket structure for local address.
4332          * XXX - what do we do for other types?
4333          */
4334         if ((so->so_type == SOCK_DGRAM) ||
4335             (so->so_type == SOCK_RAW)) {
4336             add_sock_token = 1;

4338             /* get local address */
4339             len = sizeof (so_laddr);
4340             (void) socket_getsockname(so,
4341                 (struct sockaddr *)so_laddr, &len, CRED());

4343             /* get peer address */
4344             bzero((void *)so_faddr, sizeof (so_faddr));

4346             /* sanity check */
4347             if (uap->from == NULL)
4348                 break;

4350             /* sanity checks */
4351             if (uap->fromlen == 0)
4352                 break;

4354             if (copyin((caddr_t)(uap->fromlen), (caddr_t)&fromlen,
4355                 sizeof (fromlen)) != 0)
4356                 break;

4358             if (fromlen == 0)
4359                 break;

4361             /* enforce maximum size */
4362             if (fromlen > sizeof (so_faddr))
4363                 fromlen = sizeof (so_faddr);

```

```

4365         if (copyin((caddr_t)(uap->from), so_faddr,
4366                   fromlen) != 0)
4367             break;
4369     } else if (so->so_type == SOCK_STREAM) {
4371         /* get path from file struct here */
4372         fad = F2A(fp);
4373         ASSERT(fad);
4375         /*
4376          * already processed this file for read attempt
4377          */
4378         if (fad->fad_flags & FAD_READ) {
4379             /* don't want to audit every recvfrom attempt */
4380             tad->tad_flag = 0;
4381             /* free any residual audit data */
4382             au_close(kctx, &(u_ad), 0, 0, NULL);
4383             releasef(fd);
4384             return;
4385         }
4386         /*
4387          * mark things so we know what happened and don't
4388          * repeat things
4389          */
4390         fad->fad_flags |= FAD_READ;
4392         bzero((void *)so_laddr, sizeof (so_laddr));
4393         bzero((void *)so_faddr, sizeof (so_faddr));
4395         /* get local and foreign addresses */
4396         len = sizeof (so_laddr);
4397         (void) socket_getsockname(so,
4398                                  (struct sockaddr *)so_laddr, &len, CRED());
4399         len = sizeof (so_faddr);
4400         (void) socket_getpeername(so,
4401                                   (struct sockaddr *)so_faddr, &len, B_FALSE, CRED());
4403         add_sock_token = 1;
4404     }
4406     /* XXX - what about SOCK_RDM/SOCK_SEQPACKET ??? */
4408     break;
4410 case AF_UNIX:
4411     /*
4412      * first check if this is first time through. Too much
4413      * duplicate code to put this in an aui_ routine.
4414      */
4416     /* get path from file struct here */
4417     fad = F2A(fp);
4418     ASSERT(fad);
4420     /*
4421      * already processed this file for read attempt
4422      */
4423     if (fad->fad_flags & FAD_READ) {
4424         /* don't want to audit every recvfrom attempt */
4425         tad->tad_flag = 0;
4426         /* free any residual audit data */
4427         au_close(kctx, &(u_ad), 0, 0, 0, NULL);
4428         releasef(fd);
4429         return;

```

```

4430     }
4431     /*
4432     * mark things so we know what happened and don't
4433     * repeat things
4434     */
4435     fad->fad_flags |= FAD_READ;
4437     if (fad->fad_aupath != NULL) {
4438         au_uwrite(au_to_path(fad->fad_aupath));
4439     } else {
4440         au_uwrite(au_to_arg32(1, "no path: fd", fd));
4441     }
4443     audit_attributes(fp->f_vnode);
4445     releasef(fd);
4447     return;
4449 default:
4450     break;
4452 }
4454 releasef(fd);
4456 au_uwrite(au_to_arg32(1, "so", (uint32_t)fd));
4458 if (add_sock_token == 0) {
4459     au_uwrite(au_to_arg32(1, "family", (uint32_t)so_family));
4460     au_uwrite(au_to_arg32(1, "type", (uint32_t)so_type));
4461     au_uwrite(au_to_arg32(3, "flags", (uint32_t)(uap->flags)));
4462     return;
4463 }
4465 au_uwrite(au_to_arg32(3, "flags", (uint32_t)(uap->flags)));
4467 au_uwrite(au_to_socket_ex(so_family, so_type, so_laddr, so_faddr));
4468 }
4470 /*ARGSUSED*/
4471 static void
4472 auf_sendmsg(struct t_audit_data *tad, int error, rval_t *rval)
4473 {
4474     struct a {
4475         long    fd;
4476         long    msg; /* struct msghdr */
4477         long    flags;
4478     } *uap = (struct a *)ttolwp(curthread)->lwp_ap;
4480     struct sonode *so;
4481     char so_laddr[sizeof (struct sockaddr_in6)];
4482     char so_faddr[sizeof (struct sockaddr_in6)];
4483     int    err;
4484     int    fd;
4485     short so_family, so_type;
4486     int    add_sock_token = 0;
4487     socklen_t len;
4488     struct file *fp;
4489     struct f_audit_data *fad;
4490     caddr_t msg_name;
4491     socklen_t msg_namelen;
4492     STRUCT_DECL(msghdr, msg);
4493     au_kcontext_t *kctx = GET_KCTX_PZ;
4495     fd = (int)uap->fd;

```

```

4497     /* bail if an error */
4498     if (error) {
4499         /* XXX include destination address from system call arguments */
4500         au_uwrite(au_to_arg32(1, "so", (uint32_t)fd));
4501         au_uwrite(au_to_arg32(3, "flags", (uint32_t)(uap->flags)));
4502         return;
4503     }

4505     if ((so = getsonode(fd, &err, &fp)) == NULL) {
4506         /*
4507          * not security relevant if doing a sendmsg from non socket
4508          * so no extra tokens. Should probably turn off audit record
4509          * generation here.
4510          */
4511         return;
4512     }

4514     so_family = so->so_family;
4515     so_type = so->so_type;

4517     switch (so_family) {
4518     case AF_INET:
4519     case AF_INET6:
4520         /*
4521          * if datagram type socket, then just use what is in
4522          * socket structure for local address.
4523          * XXX - what do we do for other types?
4524          */
4525         if ((so->so_type == SOCK_DGRAM) ||
4526             (so->so_type == SOCK_RAW)) {

4528             bzero((void *)so_laddr, sizeof (so_laddr));
4529             bzero((void *)so_faddr, sizeof (so_faddr));

4531             /* get local address */
4532             len = sizeof (so_laddr);
4533             (void) socket_getsockname(so,
4534                 (struct sockaddr *)so_laddr, &len, CRED());

4536             /* get peer address */
4537             STRUCT_INIT(msg, get_udatamodel());

4539             if (copyin((caddr_t)(uap->msg),
4540                 (caddr_t)STRUCT_BUF(msg), STRUCT_SIZE(msg)) != 0) {
4541                 break;
4542             }
4543             msg_name = (caddr_t)STRUCT_FGETP(msg, msg_name);
4544             if (msg_name == NULL)
4545                 break;

4547             msg_namelen = (socklen_t)STRUCT_FGET(msg, msg_namelen);
4548             /* length is value from recvmmsg - sanity check */
4549             if (msg_namelen == 0)
4550                 break;

4552             if (copyin(msg_name, so_faddr,
4553                 sizeof (so_faddr)) != 0)
4554                 break;

4556             add_sock_token = 1;

4558         } else if (so->so_type == SOCK_STREAM) {

4560             /* get path from file struct here */
4561             fad = F2A(fp);

```

```

4562             ASSERT(fad);

4564             /*
4565              * already processed this file for write attempt
4566              */
4567             if (fad->fad_flags & FAD_WRITE) {
4568                 releasef(fd);
4569                 /* don't want to audit every sendmsg attempt */
4570                 tad->tad_flag = 0;
4571                 /* free any residual audit data */
4572                 au_close(kctx, &(u_ad), 0, 0, 0, NULL);
4573                 return;
4574             }

4576             /*
4577              * mark things so we know what happened and don't
4578              * repeat things
4579              */
4580             fad->fad_flags |= FAD_WRITE;

4582             bzero((void *)so_laddr, sizeof (so_laddr));
4583             bzero((void *)so_faddr, sizeof (so_faddr));

4585             /* get local and foreign addresses */
4586             len = sizeof (so_laddr);
4587             (void) socket_getsockname(so,
4588                 (struct sockaddr *)so_laddr, &len, CRED());
4589             len = sizeof (so_faddr);
4590             (void) socket_getpeername(so,
4591                 (struct sockaddr *)so_faddr, &len, B_FALSE, CRED());

4593             add_sock_token = 1;
4594         }

4596         /* XXX - what about SOCK_RAW/SOCK_RDM/SOCK_SEQPACKET ??? */

4598         break;

4600     case AF_UNIX:
4601         /*
4602          * first check if this is first time through. Too much
4603          * duplicate code to put this in an aui_ routine.
4604          */

4606         /* get path from file struct here */
4607         fad = F2A(fp);
4608         ASSERT(fad);

4610         /*
4611          * already processed this file for write attempt
4612          */
4613         if (fad->fad_flags & FAD_WRITE) {
4614             releasef(fd);
4615             /* don't want to audit every sendmsg attempt */
4616             tad->tad_flag = 0;
4617             /* free any residual audit data */
4618             au_close(kctx, &(u_ad), 0, 0, 0, NULL);
4619             return;
4620         }
4621         /*
4622          * mark things so we know what happened and don't
4623          * repeat things
4624          */
4625         fad->fad_flags |= FAD_WRITE;

4627         if (fad->fad_aupath != NULL) {

```

```

4628         au_uwrite(au_to_path(fad->fad_aupath));
4629     } else {
4630         au_uwrite(au_to_arg32(1, "no path: fd", fd));
4631     }
4633     audit_attributes(fp->f_vnode);
4635     releasef(fd);
4637     return;
4639 default:
4640     break;
4641 }
4643 releasef(fd);
4645 au_uwrite(au_to_arg32(1, "so", (uint32_t)fd));
4647 if (add_sock_token == 0) {
4648     au_uwrite(au_to_arg32(1, "family", (uint32_t)so_family));
4649     au_uwrite(au_to_arg32(1, "type", (uint32_t)so_type));
4650     au_uwrite(au_to_arg32(3, "flags", (uint32_t)(uap->flags)));
4651     return;
4652 }
4654 au_uwrite(au_to_arg32(3, "flags", (uint32_t)(uap->flags)));
4656 au_uwrite(au_to_socket_ex(so_family, so_type, so_laddr, so_faddr));
4657 }
4659 /*ARGSUSED*/
4660 static void
4661 auf_sendto(struct t_audit_data *tad, int error, rval_t *rval)
4662 {
4663     struct a {
4664         long    fd;
4665         long    msg;    /* char */
4666         long    len;
4667         long    flags;
4668         long    to;    /* struct sockaddr */
4669         long    tolen;
4670     } *uap = (struct a *)ttolwp(curthread)->lwp_ap;
4672     struct sonode *so;
4673     char so_laddr[sizeof (struct sockaddr_in6)];
4674     char so_faddr[sizeof (struct sockaddr_in6)];
4675     socklen_t    tolen;
4676     int          err;
4677     int          fd;
4678     socklen_t    len;
4679     short so_family, so_type;
4680     int          add_sock_token = 0;
4681     struct file  *fp;
4682     struct f_audit_data *fad;
4683     au_kcontext_t *kctx = GET_KCTX_PZ;
4685     fd = (int)uap->fd;
4687     /* bail if an error */
4688     if (error) {
4689         au_uwrite(au_to_arg32(1, "so", (uint32_t)fd));
4690         au_uwrite(au_to_arg32(3, "flags", (uint32_t)(uap->flags)));
4691         /* XXX include destination address from system call arguments */
4692         return;
4693     }

```

```

4695     if ((so = getsonode(fd, &err, &fp)) == NULL) {
4696         /*
4697          * not security relevant if doing a sendto using non socket
4698          * so no extra tokens. Should probably turn off audit record
4699          * generation here.
4700          */
4701         return;
4702     }
4704     so_family = so->so_family;
4705     so_type = so->so_type;
4707     /*
4708      * only putout SOCKET_EX token if INET/INET6 family.
4709      * XXX - what do we do about other families?
4710      */
4712     switch (so_family) {
4713     case AF_INET:
4714     case AF_INET6:
4716         /*
4717          * if datagram type socket, then just use what is in
4718          * socket structure for local address.
4719          * XXX - what do we do for other types?
4720          */
4721         if ((so->so_type == SOCK_DGRAM) ||
4722             (so->so_type == SOCK_RAW)) {
4724             bzero((void *)so_laddr, sizeof (so_laddr));
4725             bzero((void *)so_faddr, sizeof (so_faddr));
4727             /* get local address */
4728             len = sizeof (so_laddr);
4729             (void) socket_getsockname(so,
4730                 (struct sockaddr *)so_laddr, &len, CRED());
4732             /* get peer address */
4734             /* sanity check */
4735             if (uap->to == NULL)
4736                 break;
4738             /* sanity checks */
4739             if (uap->tolen == 0)
4740                 break;
4742             tolen = (socklen_t)uap->tolen;
4744             /* enforce maximum size */
4745             if (tolen > sizeof (so_faddr))
4746                 tolen = sizeof (so_faddr);
4748             if (copyin((caddr_t)(uap->to), so_faddr, tolen) != 0)
4749                 break;
4751             add_sock_token = 1;
4752         } else {
4753             /*
4754              * check if this is first time through.
4755              */
4757             /* get path from file struct here */
4758             fad = F2A(fp);
4759             ASSERT(fad);

```

```

4761         /*
4762          * already processed this file for write attempt
4763          */
4764         if (fad->fad_flags & FAD_WRITE) {
4765             /* don't want to audit every sendto attempt */
4766             tad->tad_flag = 0;
4767             /* free any residual audit data */
4768             au_close(kctx, &(u_ad), 0, 0, 0, NULL);
4769             releasef(fd);
4770             return;
4771         }
4772         /*
4773          * mark things so we know what happened and don't
4774          * repeat things
4775          */
4776         fad->fad_flags |= FAD_WRITE;
4777
4778         bzero((void *)so_laddr, sizeof (so_laddr));
4779         bzero((void *)so_faddr, sizeof (so_faddr));
4780
4781         /* get local and foreign addresses */
4782         len = sizeof (so_laddr);
4783         (void) socket_getsockname(so,
4784             (struct sockaddr *)so_laddr, &len, CRED());
4785         len = sizeof (so_faddr);
4786         (void) socket_getpeername(so,
4787             (struct sockaddr *)so_faddr, &len, B_FALSE, CRED());
4788
4789         add_sock_token = 1;
4790     }
4791
4792     /* XXX - what about SOCK_RDM/SOCK_SEQPACKET ??? */
4793
4794     break;
4795
4796     case AF_UNIX:
4797         /*
4798          * first check if this is first time through. Too much
4799          * duplicate code to put this in an aui_ routine.
4800          */
4801
4802         /* get path from file struct here */
4803         fad = F2A(fp);
4804         ASSERT(fad);
4805
4806         /*
4807          * already processed this file for write attempt
4808          */
4809         if (fad->fad_flags & FAD_WRITE) {
4810             /* don't want to audit every sendto attempt */
4811             tad->tad_flag = 0;
4812             /* free any residual audit data */
4813             au_close(kctx, &(u_ad), 0, 0, 0, NULL);
4814             releasef(fd);
4815             return;
4816         }
4817         /*
4818          * mark things so we know what happened and don't
4819          * repeat things
4820          */
4821         fad->fad_flags |= FAD_WRITE;
4822
4823         if (fad->fad_aupath != NULL) {
4824             au_uwrite(au_to_path(fad->fad_aupath));
4825         } else {

```

```

4826             au_uwrite(au_to_arg32(1, "no path: fd", fd));
4827         }
4828
4829         audit_attributes(fp->f_vnode);
4830
4831         releasef(fd);
4832
4833         return;
4834
4835     default:
4836         break;
4837
4838     }
4839
4840     releasef(fd);
4841
4842     au_uwrite(au_to_arg32(1, "so", (uint32_t)fd));
4843
4844     if (add_sock_token == 0) {
4845         au_uwrite(au_to_arg32(1, "family", (uint32_t)so_family));
4846         au_uwrite(au_to_arg32(1, "type", (uint32_t)so_type));
4847         au_uwrite(au_to_arg32(3, "flags", (uint32_t)(uap->flags)));
4848         return;
4849     }
4850
4851     au_uwrite(au_to_arg32(3, "flags", (uint32_t)(uap->flags)));
4852
4853     au_uwrite(au_to_socket_ex(so_family, so_type, so_laddr, so_faddr));
4854
4855 }
4856
4857 /*
4858  * XXX socket(2) may be equivalent to open(2) on a unix domain
4859  * socket. This needs investigation.
4860  */
4861
4862 /*ARGSUSED*/
4863 static void
4864 aus_socket(struct t_audit_data *tad)
4865 {
4866     struct a {
4867         long    domain;
4868         long    type;
4869         long    protocol;
4870     } *uap = (struct a *)ttolwp(curthread)->lwp_ap;
4871
4872     au_uwrite(au_to_arg32(1, "domain", (uint32_t)uap->domain));
4873     au_uwrite(au_to_arg32(2, "type", (uint32_t)uap->type));
4874     au_uwrite(au_to_arg32(3, "protocol", (uint32_t)uap->protocol));
4875 }
4876
4877 /*ARGSUSED*/
4878 static void
4879 aus_sigqueue(struct t_audit_data *tad)
4880 {
4881     struct a {
4882         long    pid;
4883         long    signo;
4884         long    *val;
4885     } *uap = (struct a *)ttolwp(curthread)->lwp_ap;
4886     struct proc *p;
4887     uid_t uid, ruid;
4888     gid_t gid, rgid;
4889     pid_t pid;
4890     const auditinfo_addr_t *ainfo;
4891     cred_t *cr;

```



```

4893     pid = (pid_t)uap->pid;
4895     au_uwrite(au_to_arg32(2, "signal", (uint32_t)uap->signo));
4896     if (pid > 0) {
4897         mutex_enter(&pidlock);
4898         if ((p = prfind(pid)) == (struct proc *)0) {
4899             mutex_exit(&pidlock);
4900             return;
4901         }
4902         mutex_enter(&p->p_lock); /* so process doesn't go away */
4903         mutex_exit(&pidlock);
4905         mutex_enter(&p->p_crlock);
4906         crhold(cr = p->p_cred);
4907         mutex_exit(&p->p_crlock);
4908         mutex_exit(&p->p_lock);
4910         ainfo = crgetauid(cr);
4911         if (ainfo == NULL) {
4912             crfree(cr);
4913             return;
4914         }
4916         uid = crgetuid(cr);
4917         gid = crgetgid(cr);
4918         ruid = crgetruid(cr);
4919         rgid = crgetrgid(cr);
4920         au_uwrite(au_to_process(uid, gid, ruid, rgid, pid,
4921             ainfo->ai_auid, ainfo->ai_asid, &ainfo->ai_termid));
4922         crfree(cr);
4923     }
4924     else
4925         au_uwrite(au_to_arg32(1, "process ID", (uint32_t)pid));
4926 }
4928 /*ARGSUSED*/
4929 static void
4930 aus_inst_sync(struct t_audit_data *tad)
4931 {
4932     struct a {
4933         long    name;    /* char */
4934         long    flags;
4935     } *uap = (struct a *)ttolwp(curthread)->lwp_ap;
4937     au_uwrite(au_to_arg32(2, "flags", (uint32_t)uap->flags));
4938 }
4940 /*ARGSUSED*/
4941 static void
4942 aus_brandsys(struct t_audit_data *tad)
4943 {
4944     klpw_t *clwp = ttolwp(curthread);
4946     struct a {
4947         long    cmd;
4948         long    arg1;
4949         long    arg2;
4950         long    arg3;
4951         long    arg4;
4952         long    arg5;
4953         long    arg6;
4954     } *uap = (struct a *)clwp->lwp_ap;
4956     au_uwrite(au_to_arg32(1, "cmd", (uint_t)uap->cmd));
4957 #ifdef _LP64

```

```

4958     au_uwrite(au_to_arg64(2, "arg1", (uint64_t)uap->arg1));
4959     au_uwrite(au_to_arg64(3, "arg2", (uint64_t)uap->arg2));
4960     au_uwrite(au_to_arg64(4, "arg3", (uint64_t)uap->arg3));
4961     au_uwrite(au_to_arg64(5, "arg4", (uint64_t)uap->arg4));
4962     au_uwrite(au_to_arg64(6, "arg5", (uint64_t)uap->arg5));
4963     au_uwrite(au_to_arg64(7, "arg6", (uint64_t)uap->arg6));
4964 #else
4965     au_uwrite(au_to_arg32(2, "arg1", (uint32_t)uap->arg1));
4966     au_uwrite(au_to_arg32(3, "arg2", (uint32_t)uap->arg2));
4967     au_uwrite(au_to_arg32(4, "arg3", (uint32_t)uap->arg3));
4968     au_uwrite(au_to_arg32(5, "arg4", (uint32_t)uap->arg4));
4969     au_uwrite(au_to_arg32(6, "arg5", (uint32_t)uap->arg5));
4970     au_uwrite(au_to_arg32(7, "arg6", (uint32_t)uap->arg6));
4971 #endif
4972 }
4974 /*ARGSUSED*/
4975 static void
4976 aus_p_online(struct t_audit_data *tad)
4977 {
4978     struct a {
4979         long    processor_id;
4980         long    flag;
4981     } *uap = (struct a *)ttolwp(curthread)->lwp_ap;
4983     struct flags {
4984         int     flag;
4985         char    *cflag;
4986     } aflags[6] = {
4987         { P_ONLINE, "P_ONLINE"},
4988         { P_OFFLINE, "P_OFFLINE"},
4989         { P_NOINTR, "P_NOINTR"},
4990         { P_SPARE, "P_SPARE"},
4991         { P_FAULTED, "P_FAULTED"},
4992         { P_STATUS, "P_STATUS"}
4993     };
4994     int i;
4995     char *cflag;
4997     au_uwrite(au_to_arg32(1, "processor ID", (uint32_t)uap->processor_id));
4998     au_uwrite(au_to_arg32(2, "flag", (uint32_t)uap->flag));
5000     for (i = 0; i < 6; i++) {
5001         if (aflags[i].flag == uap->flag)
5002             break;
5003     }
5004     cflag = (i == 6) ? "bad flag":aflags[i].cflag;
5006     au_uwrite(au_to_text(cflag));
5007 }
5009 /*ARGSUSED*/
5010 static void
5011 aus_processor_bind(struct t_audit_data *tad)
5012 {
5013     struct a {
5014         long    id_type;
5015         long    id;
5016         long    processor_id;
5017         long    obind;
5018     } *uap = (struct a *)ttolwp(curthread)->lwp_ap;
5020     struct proc *p;
5021     int lwpcnt;
5022     uid_t uid, ruid;
5023     gid_t gid, rgid;

```

```

5024     pid_t pid;
5025     const auditinfo_addr_t *ainfo;
5026     cred_t *cr;

5028     au_uwrite(au_to_arg32(1, "ID type", (uint32_t)uap->id_type));
5029     au_uwrite(au_to_arg32(2, "ID", (uint32_t)uap->id));
5030     if (uap->processor_id == PBIND_NONE)
5031         au_uwrite(au_to_text("PBIND_NONE"));
5032     else
5033         au_uwrite(au_to_arg32(3, "processor_id",
5034             (uint32_t)uap->processor_id));

5036     switch (uap->id_type) {
5037     case P_MYID:
5038     case P_LWPID:
5039         mutex_enter(&pidlock);
5040         p = ttoproc(curthread);
5041         if (p == NULL || p->p_as == &kas) {
5042             mutex_exit(&pidlock);
5043             return;
5044         }
5045         mutex_enter(&p->p_lock);
5046         mutex_exit(&pidlock);
5047         lwpcnt = p->p_lwpcnt;
5048         pid = p->p_pid;

5050         mutex_enter(&p->p_crlock);
5051         crhold(cr = p->p_cred);
5052         mutex_exit(&p->p_crlock);
5053         mutex_exit(&p->p_lock);

5055         ainfo = crgetauinfo(cr);
5056         if (ainfo == NULL) {
5057             crfree(cr);
5058             return;
5059         }

5061         uid = crgetuid(cr);
5062         gid = crgetgid(cr);
5063         ruid = crgetruid(cr);
5064         rgid = crgetrgid(cr);
5065         au_uwrite(au_to_process(uid, gid, ruid, rgid, pid,
5066             ainfo->ai_auid, ainfo->ai_asid, &ainfo->ai_termid));
5067         crfree(cr);
5068         break;
5069     case P_PID:
5070         mutex_enter(&pidlock);
5071         p = prfind(uap->id);
5072         if (p == NULL || p->p_as == &kas) {
5073             mutex_exit(&pidlock);
5074             return;
5075         }
5076         mutex_enter(&p->p_lock);
5077         mutex_exit(&pidlock);
5078         lwpcnt = p->p_lwpcnt;
5079         pid = p->p_pid;

5081         mutex_enter(&p->p_crlock);
5082         crhold(cr = p->p_cred);
5083         mutex_exit(&p->p_crlock);
5084         mutex_exit(&p->p_lock);

5086         ainfo = crgetauinfo(cr);
5087         if (ainfo == NULL) {
5088             crfree(cr);
5089             return;

```

```

5090     }

5092     uid = crgetuid(cr);
5093     gid = crgetgid(cr);
5094     ruid = crgetruid(cr);
5095     rgid = crgetrgid(cr);
5096     au_uwrite(au_to_process(uid, gid, ruid, rgid, pid,
5097         ainfo->ai_auid, ainfo->ai_asid, &ainfo->ai_termid));
5098     crfree(cr);

5100     break;
5101     default:
5102         return;
5103     }

5105     if (uap->processor_id == PBIND_NONE &&
5106         (!(uap->id_type == P_LWPID && lwpcnt > 1)))
5107         au_uwrite(au_to_text("PBIND_NONE for process"));
5108     else
5109         au_uwrite(au_to_arg32(3, "processor_id",
5110             (uint32_t)uap->processor_id));
5111 }

5113 /*ARGSUSED*/
5114 static au_event_t
5115 aui_doorfs(au_event_t e)
5116 {
5117     uint32_t code;

5119     struct a {
5120         long    a1;
5121         long    a2;
5122         long    a3;
5123         long    a4;
5124         long    a5;
5125         long    code;
5126     } *uap = (struct a *)ttolwp(curthread)->lwp_ap;

5128     /*
5129     *     audit formats for several of the
5130     *     door calls have not yet been determined
5131     */
5132     code = (uint32_t)uap->code;
5133     switch (code) {
5134     case DOOR_CALL:
5135         e = AUE_DOORFS_DOOR_CALL;
5136         break;
5137     case DOOR_RETURN:
5138         e = AUE_NULL;
5139         break;
5140     case DOOR_CREATE:
5141         e = AUE_DOORFS_DOOR_CREATE;
5142         break;
5143     case DOOR_REVOKE:
5144         e = AUE_DOORFS_DOOR_REVOKE;
5145         break;
5146     case DOOR_INFO:
5147         e = AUE_NULL;
5148         break;
5149     case DOOR_UCRED:
5150         e = AUE_NULL;
5151         break;
5152     case DOOR_BIND:
5153         e = AUE_NULL;
5154         break;
5155     case DOOR_UNBIND:

```

```

5156         e = AUE_NULL;
5157         break;
5158     case DOOR_GETPARAM:
5159         e = AUE_NULL;
5160         break;
5161     case DOOR_SETPARAM:
5162         e = AUE_NULL;
5163         break;
5164     default: /* illegal system call */
5165         e = AUE_NULL;
5166         break;
5167     }
5169     return (e);
5170 }

5172 static door_node_t *
5173 au_door_lookup(int did)
5174 {
5175     vnode_t *vp;
5176     file_t *fp;

5178     if ((fp = getf(did)) == NULL)
5179         return (NULL);
5180     /*
5181      * Use the underlying vnode (we may be namefs mounted)
5182      */
5183     if (VOP_REALVP(fp->f_vnode, &vp, NULL))
5184         vp = fp->f_vnode;

5186     if (vp == NULL || vp->v_type != VDOOR) {
5187         releasef(did);
5188         return (NULL);
5189     }

5191     return (VTOD(vp));
5192 }

5194 /*ARGSUSED*/
5195 static void
5196 aus_doorfs(struct t_audit_data *tad)
5197 {
5199     struct a { /* doorfs */
5200         long    a1;
5201         long    a2;
5202         long    a3;
5203         long    a4;
5204         long    a5;
5205         long    code;
5206     } *uap = (struct a *)ttolwp(curthread)->lwp_ap;

5208     door_node_t *dp;
5209     struct proc *p;
5210     uint32_t did;
5211     uid_t uid, ruid;
5212     gid_t gid, rgid;
5213     pid_t pid;
5214     const auditinfo_addr_t *ainfo;
5215     cred_t *cr;

5217     did = (uint32_t)uap->a1;

5219     switch (tad->tad_event) {
5220     case AUE_DOORFS_DOOR_CALL:
5221         au_uwrite(au_to_arg32(1, "door ID", (uint32_t)did));

```

```

5222         if ((dp = au_door_lookup(did)) == NULL)
5223             break;

5225         if (DOOR_INVALID(dp)) {
5226             releasef(did);
5227             break;
5228         }

5230         if ((p = dp->door_target) == NULL) {
5231             releasef(did);
5232             break;
5233         }
5234         mutex_enter(&p->p_lock);
5235         releasef(did);

5237         pid = p->p_pid;

5239         mutex_enter(&p->p_crlock);
5240         crhold(cr = p->p_cred);
5241         mutex_exit(&p->p_crlock);
5242         mutex_exit(&p->p_lock);

5244         ainfo = crgetainfo(cr);
5245         if (ainfo == NULL) {
5246             crfree(cr);
5247             return;
5248         }
5249         uid = crgetuid(cr);
5250         gid = crgetgid(cr);
5251         ruid = crgetruid(cr);
5252         rgid = crgetrgid(cr);
5253         au_uwrite(au_to_process(uid, gid, ruid, rgid, pid,
5254             ainfo->ai_auid, ainfo->ai_asid, &ainfo->ai_termid));
5255         crfree(cr);
5256         break;
5257     case AUE_DOORFS_DOOR_RETURN:
5258         /*
5259          * We may want to write information about
5260          * all doors (if any) which will be copied
5261          * by this call to the user space
5262          */
5263         break;
5264     case AUE_DOORFS_DOOR_CREATE:
5265         au_uwrite(au_to_arg32(3, "door attr", (uint32_t)uap->a3));
5266         break;
5267     case AUE_DOORFS_DOOR_REVOKE:
5268         au_uwrite(au_to_arg32(1, "door ID", (uint32_t)did));
5269         break;
5270     case AUE_DOORFS_DOOR_INFO:
5271         break;
5272     case AUE_DOORFS_DOOR_CRED:
5273         break;
5274     case AUE_DOORFS_DOOR_BIND:
5275         break;
5276     case AUE_DOORFS_DOOR_UNBIND: {
5277         break;
5278     }
5279     default: /* illegal system call */
5280         break;
5281     }
5282 }

5284 /*ARGSUSED*/
5285 static au_event_t
5286 aui_acl(au_event_t e)
5287 {

```

```

5288 struct a {
5289     union {
5290         long    name; /* char */
5291         long    fd;
5292     }
5293     long    cmd;
5294     long    nentries;
5295     long    arg; /* aclent_t */
5296 } *uap = (struct a *)ttolwp(curthread)->lwp_ap;
5297
5299 switch (uap->cmd) {
5300 case SETACL:
5301 case ACE_SETACL:
5302     /*
5303      * acl(SETACL/ACE_SETACL, ...) and facl(SETACL/ACE_SETACL, ...)
5304      * are expected.
5305      */
5306     break;
5307 case GETACL:
5308 case GETACLCNT:
5309 case ACE_GETACL:
5310 case ACE_GETACLCNT:
5311     /* do nothing for these four values. */
5312     e = AUE_NULL;
5313     break;
5314 default:
5315     /* illegal system call */
5316     break;
5317 }
5318
5319 return (e);
5320 }
5321
5322 static void
5323 au_acl(int cmd, int nentries, caddr_t bufp)
5324 {
5325     size_t    a_size;
5326     aclent_t  *aclbufp;
5327     ace_t     *acebufp;
5328     int       i;
5329
5330     switch (cmd) {
5331     case GETACL:
5332     case GETACLCNT:
5333         break;
5334     case SETACL:
5335         if (nentries < 3)
5336             break;
5337
5338         a_size = nentries * sizeof (aclent_t);
5339
5340         if ((aclbufp = kmem_alloc(a_size, KM_SLEEP)) == NULL)
5341             break;
5342         if (copyin(bufp, aclbufp, a_size)) {
5343             kmem_free(aclbufp, a_size);
5344             break;
5345         }
5346         for (i = 0; i < nentries; i++) {
5347             au_uwrite(au_to_acl(aclbufp + i));
5348         }
5349         kmem_free(aclbufp, a_size);
5350         break;
5351
5352     case ACE_SETACL:
5353         if (nentries < 1 || nentries > MAX_ACL_ENTRIES)

```

```

5354         break;
5355
5356         a_size = nentries * sizeof (ace_t);
5357         if ((acebufp = kmem_alloc(a_size, KM_SLEEP)) == NULL)
5358             break;
5359         if (copyin(bufp, acebufp, a_size)) {
5360             kmem_free(acebufp, a_size);
5361             break;
5362         }
5363         for (i = 0; i < nentries; i++) {
5364             au_uwrite(au_to_ace(acebufp + i));
5365         }
5366         kmem_free(acebufp, a_size);
5367         break;
5368     default:
5369         break;
5370     }
5371 }
5372
5373 /*ARGSUSED*/
5374 static void
5375 aus_acl(struct t_audit_data *tad)
5376 {
5377     struct a {
5378         long    fname;
5379         long    cmd;
5380         long    nentries;
5381         long    aclbufp;
5382     } *uap = (struct a *)ttolwp(curthread)->lwp_ap;
5383
5384     au_uwrite(au_to_arg32(2, "cmd", (uint32_t)uap->cmd));
5385     au_uwrite(au_to_arg32(3, "nentries", (uint32_t)uap->nentries));
5386
5387     au_acl(uap->cmd, uap->nentries, (caddr_t)uap->aclbufp);
5388 }
5389
5390 /*ARGSUSED*/
5391 static void
5392 aus_facl(struct t_audit_data *tad)
5393 {
5394     struct a {
5395         long    fd;
5396         long    cmd;
5397         long    nentries;
5398         long    aclbufp;
5399     } *uap = (struct a *)ttolwp(curthread)->lwp_ap;
5400     struct file *fp;
5401     struct vnode *vp;
5402     struct f_audit_data *fad;
5403     int fd;
5404
5405     au_uwrite(au_to_arg32(2, "cmd", (uint32_t)uap->cmd));
5406     au_uwrite(au_to_arg32(3, "nentries", (uint32_t)uap->nentries));
5407
5408     fd = (int)uap->fd;
5409
5410     if ((fp = getf(fd)) == NULL)
5411         return;
5412
5413     /* get path from file struct here */
5414     fad = F2A(fp);
5415     if (fad->fad_aupath != NULL) {
5416         au_uwrite(au_to_path(fad->fad_aupath));
5417     } else {
5418         au_uwrite(au_to_arg32(1, "no path: fd", (uint32_t)fd));
5419     }

```

```

5421     vp = fp->f_vnode;
5422     audit_attributes(vp);

5424     /* decrement file descriptor reference count */
5425     releasef(fd);

5427     au_acl(uap->cmd, uap->nentries, (caddr_t)uap->aclbufp);
5428 }

5430 /*ARGSUSED*/
5431 static void
5432 auf_read(tad, error, rval)
5433     struct t_audit_data *tad;
5434     int error;
5435     rval_t *rval;
5436 {
5437     struct file *fp;
5438     struct f_audit_data *fad;
5439     int fd;
5440     register struct a {
5441         long fd;
5442     } *uap = (struct a *)ttolwp(curthread)->lwp_ap;
5443     au_kcontext_t *kctx = GET_KCTX_PZ;

5445     fd = (int)uap->fd;

5447     /*
5448     * convert file pointer to file descriptor
5449     * Note: fd ref count incremented here.
5450     */
5451     if ((fp = getf(fd)) == NULL)
5452         return;

5454     /* get path from file struct here */
5455     fad = F2A(fp);
5456     ASSERT(fad);

5458     /*
5459     * already processed this file for read attempt
5460     *
5461     * XXX might be better to turn off auditing in a aui_read() routine.
5462     */
5463     if (fad->fad_flags & FAD_READ) {
5464         /* don't really want to audit every read attempt */
5465         tad->tad_flag = 0;
5466         /* free any residual audit data */
5467         au_close(kctx, &(u_ad), 0, 0, 0, NULL);
5468         releasef(fd);
5469         return;
5470     }
5471     /* mark things so we know what happened and don't repeat things */
5472     fad->fad_flags |= FAD_READ;

5474     if (fad->fad_aupath != NULL) {
5475         au_uwrite(au_to_path(fad->fad_aupath));
5476     } else {
5477         au_uwrite(au_to_arg32(1, "no path: fd", (uint32_t)fd));
5478     }

5480     /* include attributes */
5481     audit_attributes(fp->f_vnode);

5483     /* decrement file descriptor reference count */
5484     releasef(fd);
5485 }

```

```

5487 /*ARGSUSED*/
5488 static void
5489 auf_write(tad, error, rval)
5490     struct t_audit_data *tad;
5491     int error;
5492     rval_t *rval;
5493 {
5494     struct file *fp;
5495     struct f_audit_data *fad;
5496     int fd;
5497     register struct a {
5498         long fd;
5499     } *uap = (struct a *)ttolwp(curthread)->lwp_ap;
5500     au_kcontext_t *kctx = GET_KCTX_PZ;

5502     fd = (int)uap->fd;

5504     /*
5505     * convert file pointer to file descriptor
5506     * Note: fd ref count incremented here.
5507     */
5508     if ((fp = getf(fd)) == NULL)
5509         return;

5511     /* get path from file struct here */
5512     fad = F2A(fp);
5513     ASSERT(fad);

5515     /*
5516     * already processed this file for write attempt
5517     *
5518     * XXX might be better to turn off auditing in a aus_write() routine.
5519     */
5520     if (fad->fad_flags & FAD_WRITE) {
5521         /* don't really want to audit every write attempt */
5522         tad->tad_flag = 0;
5523         /* free any residual audit data */
5524         au_close(kctx, &(u_ad), 0, 0, 0, NULL);
5525         releasef(fd);
5526         return;
5527     }
5528     /* mark things so we know what happened and don't repeat things */
5529     fad->fad_flags |= FAD_WRITE;

5531     if (fad->fad_aupath != NULL) {
5532         au_uwrite(au_to_path(fad->fad_aupath));
5533     } else {
5534         au_uwrite(au_to_arg32(1, "no path: fd", (uint32_t)fd));
5535     }

5537     /* include attributes */
5538     audit_attributes(fp->f_vnode);

5540     /* decrement file descriptor reference count */
5541     releasef(fd);
5542 }

5544 /*ARGSUSED*/
5545 static void
5546 auf_recv(tad, error, rval)
5547     struct t_audit_data *tad;
5548     int error;
5549     rval_t *rval;
5550 {
5551     struct sonode *so;

```

```

5552 char so_laddr[sizeof (struct sockaddr_in6)];
5553 char so_faddr[sizeof (struct sockaddr_in6)];
5554 struct file *fp;
5555 struct f_audit_data *fad;
5556 int fd;
5557 int err;
5558 socklen_t len;
5559 short so_family, so_type;
5560 register struct a {
5561     long fd;
5562 } *uap = (struct a *)ttolwp(curthread)->lwp_ap;
5563 au_kcontext_t *kctx = GET_KCTX_PZ;

5565 /*
5566  * If there was an error, then nothing to do. Only generate
5567  * audit record on first successful recv.
5568  */
5569 if (error) {
5570     /* Turn off audit record generation here. */
5571     tad->tad_flag = 0;
5572     /* free any residual audit data */
5573     au_close(kctx, &(u_ad), 0, 0, 0, NULL);
5574     return;
5575 }

5577 fd = (int)uap->fd;

5579 if ((so = getsonode(fd, &err, &fp)) == NULL) {
5580     /* Turn off audit record generation here. */
5581     tad->tad_flag = 0;
5582     /* free any residual audit data */
5583     au_close(kctx, &(u_ad), 0, 0, 0, NULL);
5584     return;
5585 }

5587 /* get path from file struct here */
5588 fad = F2A(fp);
5589 ASSERT(fad);

5591 /*
5592  * already processed this file for read attempt
5593  */
5594 if (fad->fad_flags & FAD_READ) {
5595     releasef(fd);
5596     /* don't really want to audit every recv call */
5597     tad->tad_flag = 0;
5598     /* free any residual audit data */
5599     au_close(kctx, &(u_ad), 0, 0, 0, NULL);
5600     return;
5601 }

5603 /* mark things so we know what happened and don't repeat things */
5604 fad->fad_flags |= FAD_READ;

5606 so_family = so->so_family;
5607 so_type = so->so_type;

5609 switch (so_family) {
5610 case AF_INET:
5611 case AF_INET6:
5612     /*
5613      * Only for connections.
5614      * XXX - do we need to worry about SOCK_DGRAM or other types???
5615      */
5616     if (so->so_state & SS_ISBOUND) {

```

```

5618     bzero((void *)so_laddr, sizeof (so_laddr));
5619     bzero((void *)so_faddr, sizeof (so_faddr));

5621     /* get local and foreign addresses */
5622     len = sizeof (so_laddr);
5623     (void) socket_getsockname(so,
5624         (struct sockaddr *)so_laddr, &len, CRED());
5625     len = sizeof (so_faddr);
5626     (void) socket_getpeername(so,
5627         (struct sockaddr *)so_faddr, &len, B_FALSE, CRED());

5629     /*
5630      * only way to drop out of switch. Note that we
5631      * we release fd below.
5632      */

5634     break;
5635 }

5637 releasef(fd);

5639 /* don't really want to audit every recv call */
5640 tad->tad_flag = 0;
5641 /* free any residual audit data */
5642 au_close(kctx, &(u_ad), 0, 0, 0, NULL);

5644     return;

5646 case AF_UNIX:

5648     if (fad->fad_aupath != NULL) {
5649         au_uwrite(au_to_path(fad->fad_aupath));
5650     } else {
5651         au_uwrite(au_to_arg32(1, "no path: fd", fd));
5652     }

5654     audit_attributes(fp->f_vnode);

5656     releasef(fd);

5658     return;

5660 default:
5661     releasef(fd);

5663     au_uwrite(au_to_arg32(1, "so", (uint32_t)fd));
5664     au_uwrite(au_to_arg32(1, "family", (uint32_t)so_family));
5665     au_uwrite(au_to_arg32(1, "type", (uint32_t)so_type));

5667     return;
5668 }

5670 releasef(fd);

5672     au_uwrite(au_to_arg32(1, "so", (uint32_t)fd));

5674     au_uwrite(au_to_socket_ex(so_family, so_type, so_laddr, so_faddr));

5676 }

5678 /*ARGSUSED*/
5679 static void
5680 auf_send(tad, error, rval)
5681     struct t_audit_data *tad;
5682     int error;
5683     rval_t *rval;

```

```

5684 {
5685     struct sonode *so;
5686     char so_laddr[sizeof (struct sockaddr_in6)];
5687     char so_faddr[sizeof (struct sockaddr_in6)];
5688     struct file *fp;
5689     struct f_audit_data *fad;
5690     int fd;
5691     int err;
5692     socklen_t len;
5693     short so_family, so_type;
5694     register struct a {
5695         long fd;
5696     } *uap = (struct a *)ttolwp(curthread)->lwp_ap;
5697     au_kcontext_t *kctx = GET_KCTX_PZ;

5699     fd = (int)uap->fd;

5701     /*
5702     * If there was an error, then nothing to do. Only generate
5703     * audit record on first successful send.
5704     */
5705     if (error != 0) {
5706         /* Turn off audit record generation here. */
5707         tad->tad_flag = 0;
5708         /* free any residual audit data */
5709         au_close(kctx, &(u_ad), 0, 0, 0, NULL);
5710         return;
5711     }

5713     fd = (int)uap->fd;

5715     if ((so = getsonode(fd, &err, &fp)) == NULL) {
5716         /* Turn off audit record generation here. */
5717         tad->tad_flag = 0;
5718         /* free any residual audit data */
5719         au_close(kctx, &(u_ad), 0, 0, 0, NULL);
5720         return;
5721     }

5723     /* get path from file struct here */
5724     fad = F2A(fp);
5725     ASSERT(fad);

5727     /*
5728     * already processed this file for write attempt
5729     */
5730     if (fad->fad_flags & FAD_WRITE) {
5731         releasef(fd);
5732         /* don't really want to audit every send call */
5733         tad->tad_flag = 0;
5734         /* free any residual audit data */
5735         au_close(kctx, &(u_ad), 0, 0, 0, NULL);
5736         return;
5737     }

5739     /* mark things so we know what happened and don't repeat things */
5740     fad->fad_flags |= FAD_WRITE;

5742     so_family = so->so_family;
5743     so_type = so->so_type;

5745     switch (so_family) {
5746     case AF_INET:
5747     case AF_INET6:
5748         /*
5749         * Only for connections.

```

```

5750     * XXX - do we need to worry about SOCK_DGRAM or other types???
5751     */
5752     if (so->so_state & SS_ISBOUND) {

5754         bzero((void *)so_laddr, sizeof (so_laddr));
5755         bzero((void *)so_faddr, sizeof (so_faddr));

5757         /* get local and foreign addresses */
5758         len = sizeof (so_laddr);
5759         (void) socket_getsockname(so,
5760             (struct sockaddr *)so_laddr, &len, CRED());
5761         len = sizeof (so_faddr);
5762         (void) socket_getpeername(so,
5763             (struct sockaddr *)so_faddr, &len, B_FALSE, CRED());

5765         /*
5766         * only way to drop out of switch. Note that we
5767         * we release fd below.
5768         */

5770         break;
5771     }

5773     releasef(fd);
5774     /* don't really want to audit every send call */
5775     tad->tad_flag = 0;
5776     /* free any residual audit data */
5777     au_close(kctx, &(u_ad), 0, 0, 0, NULL);

5779     return;

5781     case AF_UNIX:

5783         if (fad->fad_aupath != NULL) {
5784             au_uwrite(au_to_path(fad->fad_aupath));
5785         } else {
5786             au_uwrite(au_to_arg32(1, "no path: fd", fd));
5787         }

5789         audit_attributes(fp->f_vnode);

5791         releasef(fd);

5793         return;

5795     default:
5796         releasef(fd);

5798         au_uwrite(au_to_arg32(1, "so", (uint32_t)fd));
5799         au_uwrite(au_to_arg32(1, "family", (uint32_t)so_family));
5800         au_uwrite(au_to_arg32(1, "type", (uint32_t)so_type));

5802         return;
5803     }

5805     releasef(fd);

5807     au_uwrite(au_to_arg32(1, "so", (uint32_t)fd));

5809     au_uwrite(au_to_socket_ex(so_family, so_type, so_laddr, so_faddr));
5810 }

5812 static au_event_t
5813 aui_forksys(au_event_t e)
5814 {
5815     struct a {

```

```
5816         long    subcode;
5817         long    flags;
5818     } *uap = (struct a *)ttolwp(curthread)->lwp_ap;

5820     switch ((uint_t)uap->subcode) {
5821     case 0:
5822         e = AUE_FORK1;
5823         break;
5824     case 1:
5825         e = AUE_FORKALL;
5826         break;
5827     case 2:
5828         e = AUE_VFORK;
5829         break;
5830     default:
5831         e = AUE_NULL;
5832         break;
5833     }

5835     return (e);
5836 }

5838 /*ARGSUSED*/
5839 static au_event_t
5840 aui_portfs(au_event_t e)
5841 {
5842     struct a {          /* portfs */
5843         long    a1;
5844         long    a2;
5845         long    a3;
5846     } *uap = (struct a *)ttolwp(curthread)->lwp_ap;

5848     /*
5849     * check opcode
5850     */
5851     switch (((uint_t)uap->a1) & PORT_CODE_MASK) {
5852     case PORT_ASSOCIATE:
5853         /* check source */
5854         if (((uint_t)uap->a3 == PORT_SOURCE_FILE) ||
5855             ((uint_t)uap->a3 == PORT_SOURCE_FD)) {
5856             e = AUE_PORTFS_ASSOCIATE;
5857         } else {
5858             e = AUE_NULL;
5859         }
5860         break;
5861     case PORT DISSOCIATE:
5862         /* check source */
5863         if (((uint_t)uap->a3 == PORT_SOURCE_FILE) ||
5864             ((uint_t)uap->a3 == PORT_SOURCE_FD)) {
5865             e = AUE_PORTFS DISSOCIATE;
5866         } else {
5867             e = AUE_NULL;
5868         }
5869         break;
5870     default:
5871         e = AUE_NULL;
5872     }
5873     return (e);
5874 }
```



```

*****
15523 Thu Jun 30 22:03:15 2016
new/usr/src/uts/common/c2/audit_kevents.h
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
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 (c) 1992, 2010, Oracle and/or its affiliates. All rights reserved.
23 */

25 #ifndef _BSM_AUDIT_KEVENTS_H
26 #define _BSM_AUDIT_KEVENTS_H

28 #ifdef __cplusplus
29 extern "C" {
30 #endif

32 /*
33  * Audit event numbers.
34  *
35  *      0          Reserved as an invalid event number.
36  *      1 - 511   Allocated for Solaris kernel
37  *      512 - 2047 (reserved but not allocated)
38  *      2048 - 32767 Reserved for the Solaris TCB application.
39  *      32768 - 65535 Available for third party applications.
40  *
41  *      NOTE: libbsm/audit_event.txt must be updated elsewhere when changes
42  *      are made to kernel events.
43  */

45 #define AUE_NULL          0          /* =no indir system call */
46 #define AUE_EXIT          1          /* =ps exit(2) */
47 #define AUE_FORKALL       2          /* =ps forkall(2) */
48 #define AUE_OPEN          3          /* =no open(2): place holder */
49 #define AUE_CREAT         4          /* =no obsolete */
50 #define AUE_LINK          5          /* =fc link(2) */
51 #define AUE_UNLINK        6          /* =fd unlink(2) */
52 #define AUE_EXEC          7          /* =no obsolete */
53 #define AUE_CHDIR         8          /* =pm chdir(2) */
54 #define AUE_MKNOD         9          /* =fc mknod(2) */
55 #define AUE_CHMOD         10         /* =fm chmod(2) */
56 #define AUE_CHOWN        11         /* =fm chown(2) */
57 #define AUE_UMOUNT        12         /* =as umount(2): old version */
58 #define AUE_JUNK          13         /* =no non existant event */

```

```

59 #define AUE_ACCESS        14         /* =fa access(2) */
60 #define AUE_KILL          15         /* =pm kill(2) */
61 #define AUE_STAT          16         /* =fa stat(2) */
62 #define AUE_LSTAT        17         /* =fa lstat(2) */
63 #define AUE_ACCT          18         /* =as acct(2) */
64 #define AUE_MCTL         19         /* =no mctl(2) */
65 #define AUE_REBOOT        20         /* =no reboot(2) */
66 #define AUE_SYMLINK       21         /* =fc symlink(2) */
67 #define AUE_READLINK     22         /* =fr readlink(2) */
68 #define AUE_EXECEV       23         /* =ps,ex execve(2) */
69 #define AUE_CHROOT        24         /* =pm chroot(2) */
70 #define AUE_VFORK         25         /* =ps vfork(2) */
71 #define AUE_SETGROUPS     26         /* =pm setgroups(2) */
72 #define AUE_SETPGRP       27         /* =pm setpgrp(2) */
73 #define AUE_SWAPON        28         /* =no swapon(2) */
74 #define AUE_SETHOSTNAME   29         /* =no sethostname(2) */
75 #define AUE_FCNTL         30         /* =fm fcntl(2) */
76 #define AUE_SETPRIORITY   31         /* =no setpriority(2) */
77 #define AUE_CONNECT       32         /* =nt connect(2) */
78 #define AUE_ACCEPT        33         /* =nt accept(2) */
79 #define AUE_BIND          34         /* =nt bind(2) */
80 #define AUE_SETSOCKOPT    35         /* =nt setsockopt(2) */
81 #define AUE_VTRACE        36         /* =no vtrace(2) */
82 #define AUE_SETTIMEOFDAY  37         /* =no settimeofday(2) */
83 #define AUE_FCHOWN        38         /* =fm fchown(2) */
84 #define AUE_FCHMOD        39         /* =fm fchmod(2) */
85 #define AUE_SETREUID      40         /* =pm setreuid(2) */
86 #define AUE_SETREGID      41         /* =pm setregid(2) */
87 #define AUE_RENAME        42         /* =fc,fd rename(2) */
88 #define AUE_TRUNCATE      43         /* =no truncate(2) */
89 #define AUE_FTRUNCATE     44         /* =no ftruncate(2) */
90 #define AUE_FLOCK         45         /* =no flock(2) */
91 #define AUE_SHUTDOWN      46         /* =nt shutdown(2) */
92 #define AUE_MKDIR         47         /* =fc mkdir(2) */
93 #define AUE_RMDIR         48         /* =fd rmdir(2) */
94 #define AUE_UTIMES        49         /* =fm futimes(2), utimensat(2) */
95 #define AUE_ADJTIME       50         /* =as adjtime(2) */
96 #define AUE_SETRLIMIT     51         /* =ua setrlimit(2) */
97 #define AUE_KILLPG        52         /* =no killpg(2) */
98 #define AUE_NFS_SVC       53         /* =no nfs_svc(2) */
99 #define AUE_STATFS        54         /* =fa statfs(2) */
100 #define AUE_FSTATFS       55         /* =fa fstatfs(2) */
101 #define AUE_UNMOUNT       56         /* =no umount(2) */
102 #define AUE_ASYNC_DAEMON  57         /* =no async_daemon(2) */
103 #define AUE_NFS_GETFH     58         /* =no nfs_getfh(2) */
104 #define AUE_SETDOMAINNAME 59         /* =no setdomainname(2) */
105 #define AUE_QUOTACTL      60         /* =no quotactl(2) */
106 #define AUE_EXPORTFS      61         /* =nt exportfs(2) */
107 #define AUE_MOUNT         62         /* =as mount(2) */
108 #define AUE_SEMSYS        63         /* =no semsys(2): place holder */
109 #define AUE_MSGSYS        64         /* =no msgsys(2): place holder */
110 #define AUE_SHMSYS        65         /* =no shmsys(2): place holder */
111 #define AUE_BSM SYS        66         /* =no bsm sys(2): place holder */
112 #define AUE_RFSSYS        67         /* =no rfssys(2): place holder */
113 #define AUE_FCHDIR        68         /* =pm fchdir(2) */
114 #define AUE_FCHROOT       69         /* =pm fchroot(2) */
115 #define AUE_VPIXSYS       70         /* =no obsolete */
116 #define AUE_PATHCONF      71         /* =fa pathconf(2) */
117 #define AUE_OPEN_R        72         /* =fr open(2): read */
118 #define AUE_OPEN_RC       73         /* =fc,fr open(2): read,creat */
119 #define AUE_OPEN_RT       74         /* =fd,fr open(2): read,trunc */
120 #define AUE_OPEN_RTC      75         /* =fc,fd,fr open(2): rd,cr,tr */
121 #define AUE_OPEN_W        76         /* =fw open(2): write */
122 #define AUE_OPEN_WC       77         /* =fc,fw open(2): write,creat */
123 #define AUE_OPEN_WT       78         /* =fd,fw open(2): write,trunc */
124 #define AUE_OPEN_WTC      79         /* =fc,fd,fw open(2): wr,cr,tr */

```

```

125 #define AUE_OPEN_RW      80    /* =fr,fw open(2): read,write */
126 #define AUE_OPEN_RWC    81    /* =fc,fw,fr open(2): rd,wr,cr */
127 #define AUE_OPEN_RWT    82    /* =fd,fr,fw open(2): rd,wr,tr */
128 #define AUE_OPEN_RWTC   83    /* =fc,fd,fw,fr open(2): rd,wr,cr,tr */
129 #define AUE_MSGCTL      84    /* =ip msgctl(2): illegal command */
130 #define AUE_MSGCTL_RMID  85    /* =ip msgctl(2): IPC_RMID command */
131 #define AUE_MSGCTL_SET   86    /* =ip msgctl(2): IPC_SET command */
132 #define AUE_MSGCTL_STAT  87    /* =ip msgctl(2): IPC_STAT command */
133 #define AUE_MSGGET      88    /* =ip msgget(2) */
134 #define AUE_MSGRCV      89    /* =ip msgrcv(2) */
135 #define AUE_MSGSND      90    /* =ip msgsnd(2) */
136 #define AUE_SHMCTL      91    /* =ip shmctl(2): Illegal command */
137 #define AUE_SHMCTL_RMID  92    /* =ip shmctl(2): IPC_RMID command */
138 #define AUE_SHMCTL_SET   93    /* =ip shmctl(2): IPC_SET command */
139 #define AUE_SHMCTL_STAT  94    /* =ip shmctl(2): IPC_STAT command */
140 #define AUE_SHMGET      95    /* =ip shmget(2) */
141 #define AUE_SHMAT       96    /* =ip shmat(2) */
142 #define AUE_SHMDT       97    /* =ip shmdt(2) */
143 #define AUE_SEMCTL      98    /* =ip semctl(2): illegal command */
144 #define AUE_SEMCTL_RMID  99    /* =ip semctl(2): IPC_RMID command */
145 #define AUE_SEMCTL_SET  100   /* =ip semctl(2): IPC_SET command */
146 #define AUE_SEMCTL_STAT 101   /* =ip semctl(2): IPC_STAT command */
147 #define AUE_SEMCTL_GETNCT 102  /* =ip semctl(2): GETNCT command */
148 #define AUE_SEMCTL_GETPID 103  /* =ip semctl(2): GETPID command */
149 #define AUE_SEMCTL_GETVAL 104  /* =ip semctl(2): GETVAL command */
150 #define AUE_SEMCTL_GETALL 105  /* =ip semctl(2): GETALL command */
151 #define AUE_SEMCTL_GETZCNT 106  /* =ip semctl(2): GETZCNT command */
152 #define AUE_SEMCTL_SETVAL 107  /* =ip semctl(2): SETVAL command */
153 #define AUE_SEMCTL_SETALL 108  /* =ip semctl(2): SETALL command */
154 #define AUE_SEMGET      109  /* =ip semget(2) */
155 #define AUE_SEMOP       110  /* =ip semop(2) */
156 #define AUE_CORE        111  /* =fc process dumped core */
157 #define AUE_CLOSE       112  /* =cl close(2) */
158 #define AUE_SYSTEMBOOT  113  /* =na system booted */
159 #define AUE_ASYNC_DAEMON_EXIT 114 /* =no async_daemon(2) exited */
160 #define AUE_NFSSVC_EXIT  115  /* =no nfssvc(2) exited */
161 #define AUE_PFEXEC      116  /* =ps,ex,ua,as execve(2) w/ pfexec */
162 #define AUE_OPEN_S      117  /* =fr open(2): search */
163 #define AUE_OPEN_E      118  /* =fr open(2): exec */
164 /*
165 * 119 - 129 are available for future growth (old SunOS_CMW events
166 * that had no libbsm or praudit support or references)
167 */
168 #define AUE_GETAUID      130  /* =aa getauid(2) */
169 #define AUE_SETAUID      131  /* =aa setauid(2) */
170 #define AUE_GETAUDIT     132  /* =aa getaudit(2) */
171 #define AUE_SETAUDIT     133  /* =aa setaudit(2) */
172 /*
173 */
174 #define AUE_AUDITSVCS   136  /* =no obsolete */
175 /*
176 */
177 #define AUE_AUDITON      138  /* =no auditon(2) */
178 #define AUE_AUDITON_GTERMID 139 /* =no auditctl(2): GETTERMID */
179 #define AUE_AUDITON_STERMID 140 /* =no auditctl(2): SETTERMID */
180 #define AUE_AUDITON_GPOLICY 141 /* =aa auditctl(2): GETPOLICY */
181 #define AUE_AUDITON_SPOLICY 142 /* =as auditctl(2): SETPOLICY */
182 #define AUE_AUDITON_GESTATE 143 /* =no auditctl(2): GETESTATE */
183 #define AUE_AUDITON_SESTATE 144 /* =no auditctl(2): SETESTATE */
184 #define AUE_AUDITON_GQCTRL 145 /* =as auditctl(2): GETQCTRL */
185 #define AUE_AUDITON_SQCTRL 146 /* =as auditctl(2): SETQCTRL */
186 /*
187 */
188 #define AUE_PUTMSG      150  /* =nt */
189 /*
190 */

```

```

191 #define AUE_ENTERPROM   153  /* =na enter prom */
192 #define AUE_EXITPROM   154  /* =na exit prom */
193 /*
194 */
195 /*
196 #define AUE_IOCTL      158  /* =io ioctl(2) */
197 /*
198 */
199 /*
200 */
201 /*
202 */
203 /*
204 */
205 /*
206 */
207 /*
208 /*
209 /*
210 /*
211 #define AUE_ONESIDE     173  /* =no one-sided session record */
212 #define AUE_MSGGETL    174  /* =no msggetl(2) */
213 #define AUE_MSGRCVCL   175  /* =no msgrcvl(2) */
214 #define AUE_MSGSNDL    176  /* =no msgsndl(2) */
215 #define AUE_SEMGETL    177  /* =no semgetl(2) */
216 #define AUE_SHMGETL    178  /* =no shmgetl(2) */
217 /*
218 */
219 /*
220 /*
221 #define AUE_SOCKET      183  /* =nt socket(2) */
222 #define AUE_SENTO      184  /* =nt sendto(2) */
223 #define AUE_PIPE       185  /* =no pipe(2) */
224 #define AUE_SOCKETPAIR 186  /* =no socketpair(2) */
225 #define AUE_SEND       187  /* =no send(2) */
226 #define AUE_SENDSMSG   188  /* =nt sendmsg(2) */
227 #define AUE_RECV       189  /* =no recv(2) */
228 #define AUE_RECVMSG    190  /* =nt recvmsg(2) */
229 #define AUE_RECVFROM   191  /* =nt recvfrom(2) */
230 #define AUE_READ       192  /* =no read(2) */
231 #define AUE_GETDENTS   193  /* =no getdents(2) */
232 #define AUE_LSEEK      194  /* =no lseek(2) */
233 #define AUE_WRITE      195  /* =no write(2) */
234 #define AUE_WRITEV     196  /* =no writev(2) */
235 #define AUE_NFS        197  /* =no NFS server */
236 #define AUE_READV      198  /* =no readv(2) */
237 #define AUE_OSTAT      199  /* =no obsolete */
238 #define AUE_SETUID     200  /* =pm old setuid(2) */
239 #define AUE_STIME      201  /* =as old stime(2) */
240 #define AUE_UTIME      202  /* =no obsolete */
241 #define AUE_NICE       203  /* =pm old nice(2) */
242 #define AUE_OSETPGRP   204  /* =no old setpggrp(2) */
243 #define AUE_SETGID     205  /* =pm old setgid(2) */
244 #define AUE_READL      206  /* =no readl(2) */
245 #define AUE_READVL     207  /* =no readvl(2) */
246 #define AUE_FSTAT      208  /* =no fstat(2) */
247 #define AUE_DUP2       209  /* =no obsolete */
248 #define AUE_MMAP       210  /* =no mmap(2) u-o-p */
249 #define AUE_AUDIT      211  /* =no audit(2) u-o-p */
250 #define AUE_PRIOCNLTLSYS 212 /* =pm priocntlsys */
251 #define AUE_MUNMAP     213  /* =cl munmap(2) u-o-p */
252 #define AUE_SETEGID    214  /* =pm setegid(2) */
253 #define AUE_SETEUID    215  /* =pm seteuid(2) */
254 #define AUE_PUTMSG     216  /* =nt */
255 #define AUE_GETMSG     217  /* =nt */
256 #define AUE_PUTPMMSG   218  /* =nt */

```

```

257 #define AUE_GETPMMSG 219 /* =nt */
258 #define AUE_AUDITSYS 220 /* =no place holder */
259 #define AUE_AUDITON_GETMASK 221 /* =aa */
260 #define AUE_AUDITON_SETMASK 222 /* =as */
261 #define AUE_AUDITON_GETCWD 223 /* =aa,as */
262 #define AUE_AUDITON_GETCAR 224 /* =aa,as */
263 #define AUE_AUDITON_GETSTAT 225 /* =as */
264 #define AUE_AUDITON_SETSTAT 226 /* =as */
265 #define AUE_AUDITON_SETUMASK 227 /* =as */
266 #define AUE_AUDITON_SETSMASK 228 /* =as */
267 #define AUE_AUDITON_GETCOND 229 /* =aa */
268 #define AUE_AUDITON_SETCOND 230 /* =as */
269 #define AUE_AUDITON_GETCLASS 231 /* =aa,as */
270 #define AUE_AUDITON_SETCLASS 232 /* =as */
271 #define AUE_FUSERS 233 /* =fa */
272 #define AUE_STATVFS 234 /* =fa */
273 #define AUE_XSTAT 235 /* =no obsolete */
274 #define AUE_LXSTAT 236 /* =no obsolete */
275 #define AUE_LCHOWN 237 /* =fm */
276 #define AUE_MEMCNTL 238 /* =ot */
277 #define AUE_SYSINFO 239 /* =as */
278 #define AUE_XMKNOD 240 /* =no obsolete */
279 #define AUE_FORK1 241 /* =ps */
280 #define AUE_MODCTL 242 /* =no */
281 #define AUE_MODLOAD 243 /* =as */
282 #define AUE_MODUNLOAD 244 /* =as */
283 #define AUE_MODCONFIG 245 /* =no obsolete */
284 #define AUE_MODADDMAJ 246 /* =as */
285 #define AUE_SOCKETACCEPT 247 /* =nt */
286 #define AUE_SOCKETCONNECT 248 /* =nt */
287 #define AUE_SOCKETSEND 249 /* =nt */
288 #define AUE_SOCKETRECEIVE 250 /* =nt */
289 #define AUE_ACLSET 251 /* =fm */
290 #define AUE_FACLSET 252 /* =fm */
291 #define AUE_DOORFS 253 /* =no */
292 #define AUE_DOORFS_DOOR_CALL 254 /* =ip */
293 #define AUE_DOORFS_DOOR_RETURN 255 /* =ip */
294 #define AUE_DOORFS_DOOR_CREATE 256 /* =ip */
295 #define AUE_DOORFS_DOOR_REVOKE 257 /* =ip */
296 #define AUE_DOORFS_DOOR_INFO 258 /* =ip */
297 #define AUE_DOORFS_DOOR_CRED 259 /* =ip */
298 #define AUE_DOORFS_DOOR_BIND 260 /* =ip */
299 #define AUE_DOORFS_DOOR_UNBIND 261 /* =ip */
300 #define AUE_P_ONLINE 262 /* =as */
301 #define AUE_PROCESSOR_BIND 263 /* =as */
302 #define AUE_INST_SYNC 264 /* =as */
303 #define AUE_SOCKETCONFIG 265 /* =nt */
304 #define AUE_SETAUDIT_ADDR 266 /* =aa setaudit_addr(2) */
305 #define AUE_GETAUDIT_ADDR 267 /* =aa getaudit_addr(2) */
306 #define AUE_UMOUNT2 268 /* =as umount2(2) */
307 #define AUE_FSAT 269 /* =no obsolete */
308 #define AUE_OPENAT_R 270 /* =no obsolete */
309 #define AUE_OPENAT_RC 271 /* =no obsolete */
310 #define AUE_OPENAT_RT 272 /* =no obsolete */
311 #define AUE_OPENAT_RTC 273 /* =no obsolete */
312 #define AUE_OPENAT_W 274 /* =no obsolete */
313 #define AUE_OPENAT_WC 275 /* =no obsolete */
314 #define AUE_OPENAT_WT 276 /* =no obsolete */
315 #define AUE_OPENAT_WTC 277 /* =no obsolete */
316 #define AUE_OPENAT_RW 278 /* =no obsolete */
317 #define AUE_OPENAT_RWC 279 /* =no obsolete */
318 #define AUE_OPENAT_RWT 280 /* =no obsolete */
319 #define AUE_OPENAT_RWTC 281 /* =no obsolete */
320 #define AUE_RENAMEAT 282 /* =no obsolete */
321 #define AUE_FSTATAT 283 /* =no obsolete */
322 #define AUE_FCHOWNAT 284 /* =no obsolete */

```

```

323 #define AUE_FUTIMESAT 285 /* =no obsolete */
324 #define AUE_UNLINKAT 286 /* =no obsolete */
325 #define AUE_CLOCK_SETTIME 287 /* =as clock_settime(3RT) */
326 #define AUE_NTP_ADJTIME 288 /* =as ntp_adjtime(2) */
327 #define AUE_SETPPRIV 289 /* =pm setppriv(2) */
328 #define AUE_MODDEVPLCY 290 /* =as modctl(2) */
329 #define AUE_MODADDPRI 291 /* =as modctl(2) */
330 #define AUE_CRYPTOADM 292 /* =as kernel cryptographic framework */
331 #define AUE_CONFIGKSSL 293 /* =as kernel SSL */
332 #define AUE_BRANDSYS 294 /* =ot */
333 #define AUE_PF_POLICY_ADDRULE 295 /* =as Add IPsec policy rule */
334 #define AUE_PF_POLICY_DELRULE 296 /* =as Delete IPsec policy rule */
335 #define AUE_PF_POLICY_CLONE 297 /* =as Clone IPsec policy */
336 #define AUE_PF_POLICY_FLIP 298 /* =as Flip IPsec policy */
337 #define AUE_PF_POLICY_FLUSH 299 /* =as Flush IPsec policy rules */
338 #define AUE_PF_POLICY_ALGS 300 /* =as Update IPsec algorithms */
339 #define AUE_PORTFS 301 /* =no portfs(2) - place holder */
340 #define AUE_LABELSYS_TNRH 302 /* =as tnhr(2) */
341 #define AUE_LABELSYS_TNRHTP 303 /* =as tnhrtp(2) */
342 #define AUE_LABELSYS_TNMLP 304 /* =as tnmlp(2) */
343 #define AUE_PORTFS_ASSOCIATE 305 /* =fa portfs(2) - port associate */
344 #define AUE_PORTFS DISSOCIATE 306 /* =fa portfs(2) - port disassociate */
345 #define AUE_SETSID 307 /* =pm setsid(2) */
346 #define AUE_SETPGID 308 /* =pm setpgid(2) */
347 #define AUE_FACCESSAT 309 /* =no obsolete */
348 #define AUE_AUDITON_GETAMASK 310 /* =aa */
349 #define AUE_AUDITON_SETAMASK 311 /* =as */
350 #define AUE_PSECFLAGS 312 /* =pm psecflags */

```

```
352 /* NOTE: update MAX_KEVENTS below if events are added. */
```

```
353 #define MAX_KEVENTS 312
```

```
355 #define MAX_KEVENTS 311
```

```
355 #ifdef __cplusplus
```

```
356 }
```

```
_____unchanged_portion_omitted_____
```

new/usr/src/uts/common/c2/audit_record.h

1

```
*****
14681 Thu Jun 30 22:03:16 2016
new/usr/src/uts/common/c2/audit_record.h
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
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 2010 Sun Microsystems, Inc. All rights reserved.
23 * Use is subject to license terms.
24 */

26 #ifndef _BSM_AUDIT_RECORD_H
27 #define _BSM_AUDIT_RECORD_H

30 #ifdef _KERNEL
31 #include <sys/priv.h>
32 #else
33 #include <priv.h>
34 #endif
35 #include <sys/socket.h>
36 #include <sys/acl.h>

38 #include <sys/tsol/label.h>

40 #ifdef __cplusplus
41 extern "C" {
42 #endif

44 /*
45  * Version of audit attributes
46  *
47  * OS Release      Version Number      Comments
48  * =====
49  * SunOS 5.1       2                Unbundled Package
50  * SunOS 5.3       2                Bundled into the base OS
51  * SunOS 5.4-5.x   2
52  * Trusted Solaris 2.5  3          To distinguish potential new tokens
53  * Trusted Solaris 7-8  4          Redefine X tokens that overlap with
54  *                  SunOS 5.7
55 */

57 #define TOKEN_VERSION 2
```

new/usr/src/uts/common/c2/audit_record.h

2

```
59 /*
60  * Audit record token type codes
61 */

63 /*
64  * Control token types
65 */

67 #define AUT_INVALID          ((char)0x00)
68 #define AUT_OTHER_FILE      ((char)0x11)
69 #define AUT_OTHER_FILE32    AUT_OTHER_FILE
70 #define AUT_OHEADER         ((char)0x12)
71 #define AUT_TRAILER         ((char)0x13)
72 #define AUT_HEADER          ((char)0x14)
73 #define AUT_HEADER32        AUT_HEADER
74 #define AUT_HEADER32_EX     ((char)0x15)
75 #define AUT_TRAILER_MAGIC   ((short)0xB105)

77 /*
78  * Data token types
79 */

81 #define AUT_FMRI             ((char)0x20)
82 #define AUT_DATA            ((char)0x21)
83 #define AUT_IPC             ((char)0x22)
84 #define AUT_PATH            ((char)0x23)
85 #define AUT_SUBJECT         ((char)0x24)
86 #define AUT_SUBJECT32      AUT_SUBJECT
87 #define AUT_XATPATH         ((char)0x25)
88 #define AUT_PROCESS         ((char)0x26)
89 #define AUT_PROCESS32      AUT_PROCESS
90 #define AUT_RETURN         ((char)0x27)
91 #define AUT_RETURN32       AUT_RETURN
92 #define AUT_TEXT            ((char)0x28)
93 #define AUT_OPAQUE          ((char)0x29)
94 #define AUT_IN_ADDR         ((char)0x2A)
95 #define AUT_IP              ((char)0x2B)
96 #define AUT_IPORT           ((char)0x2C)
97 #define AUT_ARG             ((char)0x2D)
98 #define AUT_ARG32          AUT_ARG
99 #define AUT_SOCKET         ((char)0x2E)
100 #define AUT_SEQ             ((char)0x2F)
101 #define AUT_USER            ((char)0x36) /* out of order */
102 #define AUT_TID             ((char)0x61) /* out of order */

104 /*
105  * Modifier token types
106 */

108 #define AUT_ACL              ((char)0x30)
109 #define AUT_ATTR            ((char)0x31)
110 #define AUT_IPC_PERM        ((char)0x32)
111 #define AUT_LABEL           ((char)0x33)
112 #define AUT_GROUPS          ((char)0x34)
113 #define AUT_ACE             ((char)0x35)
114 /* 0x37 unused */
115 #define AUT_PRIV            ((char)0x38)
116 #define AUT_UPRIV           ((char)0x39)
117 #define AUT_LIAISON         ((char)0x3A)
118 #define AUT_NEWGROUPS      ((char)0x3B)
119 #define AUT_EXEC_ARGS       ((char)0x3C)
120 #define AUT_EXEC_ENV        ((char)0x3D)
121 #define AUT_ATTR32         ((char)0x3E)
122 #define AUT_UAUTH           ((char)0x3F)
123 #define AUT_ZONENAME        ((char)0x60) /* out of order */
124 #define AUT_SECFLAGS        ((char)0x62) /* out of order */
```

```

125 #endif /* ! codereview */
127 /*
128 * X windows token types
129 */

131 #define AUT_XATOM          ((char)0x40)
132 #define AUT_XOBJ          ((char)0x41)
133 #define AUT_XPROTO        ((char)0x42)
134 #define AUT_XSELECT       ((char)0x43)

136 #if TOKEN_VERSION != 3
137 #define AUT_XCOLORMAP     ((char)0x44)
138 #define AUT_XCURSOR      ((char)0x45)
139 #define AUT_XFONT         ((char)0x46)
140 #define AUT_XGC           ((char)0x47)
141 #define AUT_XPIXMAP       ((char)0x48)
142 #define AUT_XPROPERTY     ((char)0x49)
143 #define AUT_XWINDOW       ((char)0x4A)
144 #define AUT_XCLIENT       ((char)0x4B)
145 #else /* TOKEN_VERSION == 3 */
146 #define AUT_XCOLORMAP     ((char)0x74)
147 #define AUT_XCURSOR      ((char)0x75)
148 #define AUT_XFONT         ((char)0x76)
149 #define AUT_XGC           ((char)0x77)
150 #define AUT_XPIXMAP       ((char)0x78)
151 #define AUT_XPROPERTY     ((char)0x79)
152 #define AUT_XWINDOW       ((char)0x7A)
153 #define AUT_XCLIENT       ((char)0x7B)
154 #endif /* TOKEN_VERSION != 3 */

156 /*
157 * Command token types
158 */

160 #define AUT_CMD            ((char)0x51)
161 #define AUT_EXIT           ((char)0x52)

163 /*
164 * Miscellaneous token types
165 */

167 #define AUT_HOST           ((char)0x70)

169 /*
170 * Solaris64 token types
171 */

173 #define AUT_ARG64          ((char)0x71)
174 #define AUT_RETURN64      ((char)0x72)
175 #define AUT_ATTR64        ((char)0x73)
176 #define AUT_HEADER64      ((char)0x74)
177 #define AUT_SUBJECT64     ((char)0x75)
178 #define AUT_PROCESS64     ((char)0x77)
179 #define AUT_OTHER_FILE64  ((char)0x78)

181 /*
182 * Extended network address token types
183 */

185 #define AUT_HEADER64_EX   ((char)0x79)
186 #define AUT_SUBJECT32_EX  ((char)0x7a)
187 #define AUT_PROCESS32_EX  ((char)0x7b)
188 #define AUT_SUBJECT64_EX  ((char)0x7c)
189 #define AUT_PROCESS64_EX  ((char)0x7d)
190 #define AUT_IN_ADDR_EX    ((char)0x7e)

```

```

191 #define AUT_SOCKET_EX     ((char)0x7f)

194 /*
195 * Audit print suggestion types.
196 */

198 #define AUP_BINARY        ((char)0)
199 #define AUP_OCTAL         ((char)1)
200 #define AUP_DECIMAL       ((char)2)
201 #define AUP_HEX           ((char)3)
202 #define AUP_STRING        ((char)4)

204 /*
205 * Audit data member types.
206 */

208 #define AUR_BYTE          ((char)0)
209 #define AUR_CHAR          ((char)0)
210 #define AUR_SHORT         ((char)1)
211 #define AUR_INT           ((char)2)
212 #define AUR_INT32        ((char)2)
213 #define AUR_INT64        ((char)3)

215 /*
216 * Adr structures
217 */

219 struct adr_s {
220     char *adr_stream;      /* The base of the stream */
221     char *adr_now;        /* The location within the stream */
222 };

224 typedef struct adr_s adr_t;

227 #ifdef _KERNEL

229 #include <sys/param.h>
230 #include <sys/systm.h>    /* for rval */
231 #include <sys/time.h>
232 #include <sys/types.h>
233 #include <sys/vnode.h>
234 #include <sys/mode.h>
235 #include <sys/user.h>
236 #include <sys/session.h>
237 #include <sys/ipc_impl.h>
238 #include <netinet/in_systm.h>
239 #include <netinet/in.h>
240 #include <netinet/ip.h>
241 #include <sys/socket.h>
242 #include <net/route.h>
243 #include <netinet/in_pcb.h>

245 /*
246 * au_close flag arguments
247 */

249 #define AU_OK              0x1    /* Good audit record */
250 #define AU_DONTBLOCK      0x2    /* Don't block or discard if queue full */
251 #define AU_DEFER          0x4    /* Defer record queueing to syscall end */

253 /*
254 * Audit token type is really an au_membuf pointer
255 */
256 typedef au_buff_t token_t;

```

```

257 /*
258  * token generation functions
259  */
260 token_t *au_append_token(token_t *, token_t *);
261 token_t *au_set(caddr_t, uint_t);

263 void au_free_rec(au_buff_t *);

265 #define au_getclr()          ((token_t *)au_get_buff())
266 #define au_toss_token(tok)  (au_free_rec((au_buff_t *) (tok)))

268 token_t *au_to_acl();
269 token_t *au_to_ace();
270 token_t *au_to_attr(struct vattr *);
271 token_t *au_to_data(char, char, char, char *);
272 token_t *au_to_header(int, au_event_t, au_emod_t);
273 token_t *au_to_header_ex(int, au_event_t, au_emod_t);
274 token_t *au_to_ipc(char, int);
275 token_t *au_to_ipc_perm(kipc_perm_t *);
276 token_t *au_to_iport(ushort_t);
277 token_t *au_to_in_addr(struct in_addr *);
278 token_t *au_to_in_addr_ex(int32_t *);
279 token_t *au_to_ip(struct ip *);
280 token_t *au_to_groups(const gid_t *, uint_t);
281 token_t *au_to_path(struct audit_path *);
282 token_t *au_to_seq();
283 token_t *au_to_process(uid_t, gid_t, uid_t, gid_t, pid_t,
284                      au_id_t, au_asid_t, const au_tid_addr_t *);
285 token_t *au_to_subject(uid_t, gid_t, uid_t, gid_t, pid_t,
286                      au_id_t, au_asid_t, const au_tid_addr_t *);
287 token_t *au_to_return32(int, int32_t);
288 token_t *au_to_return64(int, int64_t);
289 token_t *au_to_text(const char *);
290 /* token_t *au_to_tid(au_generic_tid_t *); no kernel implementation */
291 token_t *au_to_trailer(int);
292 token_t *au_to_uauth(char *);
293 size_t  au_zonename_length(zone_t *);
294 token_t *au_to_zonename(size_t, zone_t *);
295 token_t *au_to_arg32(char, char *, uint32_t);
296 token_t *au_to_arg64(char, char *, uint64_t);
297 token_t *au_to_socket_ex(short, short, char *, char *);
298 token_t *au_to_sock_inet(struct sockaddr_in *);
299 token_t *au_to_exec_args(const char *, ssize_t);
300 token_t *au_to_exec_env(const char *, ssize_t);
301 token_t *au_to_label(bslabel_t *);
302 token_t *au_to_privset(const char *, const priv_set_t *, char, int);
303 token_t *au_to_secflags(const char *, secflagset_t);
304 #endif /* ! codereview */

306 void  au_uwrite();
307 void  au_close(au_kcontext_t *, caddr_t *, int, au_event_t, au_emod_t,
308              timestruc_t *);
309 void  au_close_defer(token_t *, int, au_event_t, au_emod_t, timestruc_t *);
310 void  au_close_time(au_kcontext_t *, token_t *, int, au_event_t, au_emod_t,
311                   timestruc_t *);
312 void  au_free_rec(au_buff_t *);
313 void  au_write(caddr_t *, token_t *);
314 void  au_mem_init(void);
315 void  au_zone_setup();
316 void  au_enqueue(au_kcontext_t *, au_buff_t *, adr_t *, adr_t *, int, int);
317 int   au_doorio(au_kcontext_t *);
318 int   au_doormsg(au_kcontext_t *, uint32_t, void *);
319 int   au_token_size(token_t *);
320 int   au_append_rec(au_buff_t *, au_buff_t *, int);
321 int   au_append_buf(const char *, int, au_buff_t *);

```

```

323 #else /* !_KERNEL */

325 #include <limits.h>
326 #include <sys/types.h>
327 #include <sys/vnode.h>
328 #include <netinet/in_system.h>
329 #include <netinet/in.h>
330 #include <netinet/ip.h>
331 #include <sys/ipc.h>

333 struct token_s {
334     struct token_s *tt_next;      /* Next in the list */
335     short          tt_size;       /* Size of data */
336     char           tt_data;       /* The data */
337 };
338 typedef struct token_s token_t;

340 /*
341  * Old socket structure definition, formerly in <sys/socketvar.h>
342  */
343 struct oldsocket {
344     short  so_type;                /* generic type, see socket.h */
345     short  so_options;            /* from socket call, see socket.h */
346     short  so_linger;             /* time to linger while closing */
347     short  so_state;             /* internal state flags SS_*, below */
348     struct inpcb *so_pcb;        /* protocol control block */
349     struct protosw *so_proto;    /* protocol handle */
350 /*
351  * Variables for connection queuing.
352  * Socket where accepts occur is so_head in all subsidiary sockets.
353  * If so_head is 0, socket is not related to an accept.
354  * For head socket so_q0 queues partially completed connections,
355  * while so_q is a queue of connections ready to be accepted.
356  * If a connection is aborted and it has so_head set, then
357  * it has to be pulled out of either so_q0 or so_q.
358  * We allow connections to queue up based on current queue lengths
359  * and limit on number of queued connections for this socket.
360  */
361     struct oldsocket *so_head;    /* back pointer to accept socket */
362     struct oldsocket *so_q0;     /* queue of partial connections */
363     struct oldsocket *so_q;      /* queue of incoming connections */
364     short  so_q0len;            /* partials on so_q0 */
365     short  so_qlen;            /* number of connections on so_q */
366     short  so_qlimit;          /* max number queued connections */
367     short  so_timeo;           /* connection timeout */
368     ushort_t so_error;         /* error affecting connection */
369     short  so_pgrp;           /* pgrp for signals */
370     ulong_t so_oobmark;       /* chars to oob mark */
371 /*
372  * Variables for socket buffering.
373  */
374     struct sockbuf {
375         ulong_t sb_cc;          /* actual chars in buffer */
376         ulong_t sb_hiwat;      /* max actual char count */
377         ulong_t sb_mbcnt;      /* chars of mbufs used */
378         ulong_t sb_mbxmax;     /* max chars of mbufs to use */
379         ulong_t sb_lowat;      /* low water mark (not used yet) */
380         struct mbuf *sb_mb;     /* the mbuf chain */
381         struct proc *sb_sel;    /* process selecting read/write */
382         short  sb_timeo;       /* timeout (not used yet) */
383         short  sb_flags;       /* flags, see below */
384     } so_rcv, so_snd;
385 /*
386  * Hooks for alternative wakeup strategies.
387  * These are used by kernel subsystems wishing to access the socket
388  * abstraction. If so_wupfunc is nonnull, it is called in place of

```

```

389 * wakeup any time that wakeup would otherwise be called with an
390 * argument whose value is an address lying within a socket structure.
391 */
392     struct wupalt    *so_wupalt;
393 };
394 extern token_t *au_to_arg32(char, char *, uint32_t);
395 extern token_t *au_to_arg64(char, char *, uint64_t);
396 extern token_t *au_to_acl(struct acl *);
397 extern token_t *au_to_attr(struct vatrr *);
398 extern token_t *au_to_cmd(uint_t, char **, char **);
399 extern token_t *au_to_data(char, char, char, char *);
400 extern token_t *au_to_exec_args(char **);
401 extern token_t *au_to_exec_env(char **);
402 extern token_t *au_to_exit(int, int);
403 extern token_t *au_to_fmri(char *);
404 extern token_t *au_to_groups(int *);
405 extern token_t *au_to_newgroups(int, gid_t *);
406 extern token_t *au_to_header(au_event_t, au_emod_t);
407 extern token_t *au_to_header_ex(au_event_t, au_emod_t);
408 extern token_t *au_to_in_addr(struct in_addr *);
409 extern token_t *au_to_in_addr_ex(struct in6_addr *);
410 extern token_t *au_to_ipc(char, int);
411 extern token_t *au_to_ipc_perm(struct ipc_perm *);
412 extern token_t *au_to_iport(ushort_t);
413 extern token_t *au_to_me(void);
414 extern token_t *au_to_mylabel(void);
415 extern token_t *au_to_opaque(char *, short);
416 extern token_t *au_to_path(char *);
417 extern token_t *au_to_privset(const char *, const priv_set_t *);
418 extern token_t *au_to_process(au_id_t, uid_t, gid_t, uid_t, gid_t,
419                             pid_t, au_asid_t, au_tid_t *);
420 extern token_t *au_to_process_ex(au_id_t, uid_t, gid_t, uid_t, gid_t,
421                                pid_t, au_asid_t, au_tid_addr_t *);
422 extern token_t *au_to_return32(char, uint32_t);
423 extern token_t *au_to_return64(char, uint64_t);
424 extern token_t *au_to_seq(int);
425 extern token_t *au_to_label(m_label_t *);
426 extern token_t *au_to_socket(struct oldsocket *);
427 extern token_t *au_to_subject(au_id_t, uid_t, gid_t, uid_t, gid_t,
428                              pid_t, au_asid_t, au_tid_t *);
429 extern token_t *au_to_subject_ex(au_id_t, uid_t, gid_t, uid_t, gid_t,
430                                 pid_t, au_asid_t, au_tid_addr_t *);
431 extern token_t *au_to_text(char *);
432 extern token_t *au_to_tid(au_generic_tid_t *);
433 extern token_t *au_to_trailer(void);
434 extern token_t *au_to_uauth(char *);
435 extern token_t *au_to_upriv(char, char *);
436 extern token_t *au_to_user(uid_t, char *);
437 extern token_t *au_to_xatom(char *);
438 extern token_t *au_to_xselect(char *, char *, char *);
439 extern token_t *au_to_xcolormap(int32_t, uid_t);
440 extern token_t *au_to_xcursor(int32_t, uid_t);
441 extern token_t *au_to_xfont(int32_t, uid_t);
442 extern token_t *au_to_xgc(int32_t, uid_t);
443 extern token_t *au_to_xpixmap(int32_t, uid_t);
444 extern token_t *au_to_xwindow(int32_t, uid_t);
445 extern token_t *au_to_xproperty(int32_t, uid_t, char *);
446 extern token_t *au_to_xclient(uint32_t);
447 extern token_t *au_to_zonename(char *);
448 #endif /* _KERNEL */

450 #ifdef _KERNEL

452 void    adr_char(adr_t *, char *, int);
453 void    adr_int32(adr_t *, int32_t *, int);
454 void    adr_uint32(adr_t *, uint32_t *, int);

```

```

455 void    adr_int64(adr_t *, int64_t *, int);
456 void    adr_uint64(adr_t *, uint64_t *, int);
457 void    adr_short(adr_t *, short *, int);
458 void    adr_ushort(adr_t *, ushort_t *, int);
459 void    adr_start(adr_t *, char *);

461 char    *adr_getchar(adr_t *, char *);
462 char    *adr_getshort(adr_t *, short *);
463 char    *adr_getushort(adr_t *, ushort_t *);
464 char    *adr_getint32(adr_t *, int32_t *);
465 char    *adr_getuint32(adr_t *, uint32_t *);
466 char    *adr_getint64(adr_t *, int64_t *);
467 char    *adr_getuint64(adr_t *, uint64_t *);

469 int     adr_count(adr_t *);

471 #endif /* _KERNEL */

473 #ifdef __cplusplus
474 }
475 #endif

477 #endif /* _BSM_AUDIT_RECORD_H */

```

```

*****
27674 Thu Jun 30 22:03:17 2016
new/usr/src/uts/common/c2/audit_token.c
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
_____unchanged_portion_omitted_

```

```

1152 token_t *
1153 au_to_secflags(const char *which, secflagset_t set)
1154 {
1155     token_t *token, *m;
1156     adr_t adr;
1157     char data_header = AUT_SECFLAGS;
1158     short sz;
1159     char secstr[1024];
1161     token = au_getclr();
1163     adr_start(&adr, memtod(token, char *));
1164     adr_char(&adr, &data_header, 1);
1166     sz = strlen(which) + 1;
1167     adr_short(&adr, &sz, 1);
1169     token->len = (uchar_t)adr_count(&adr);
1170     m = au_getclr();
1171     (void) au_append_buf(which, sz, m);
1172     (void) au_append_rec(token, m, AU_PACK);
1173     adr.adr_now += sz;
1175     secflags_to_str(set, secstr, sizeof (secstr));
1176     sz = strlen(secstr) + 1;
1177     adr_short(&adr, &sz, 1);
1178     token->len = (uchar_t)adr_count(&adr);
1179     m = au_getclr();
1180     (void) au_append_buf(secstr, sz, m);
1181     (void) au_append_rec(token, m, AU_PACK);
1183     return (token);
1184 }
1186 #endif /* ! codereview */
1187 /*
1188  * au_to_label
1189  * returns:
1190  *     pointer to au_membuf chain containing a label token.
1191  */
1192 token_t *
1193 au_to_label(bslabel_t *label)
1194 {
1195     token_t *m;           /* local au_membuf */
1196     adr_t adr;           /* adr memory stream header */
1197     char data_header = AUT_LABEL; /* header for this token */
1199     m = au_getclr();
1201     adr_start(&adr, memtod(m, char *));
1202     adr_char(&adr, &data_header, 1);
1203     adr_char(&adr, (char *)label, sizeof (_mac_label_impl_t));
1205     m->len = adr_count(&adr);
1207     return (m);

```

```

1208 }

```



```

*****
59259 Thu Jun 30 22:03:18 2016
new/usr/src/uts/common/exec/elf/elf.c
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */

22 /*
23  * Copyright (c) 1989, 2010, Oracle and/or its affiliates. All rights reserved.
24 */

26 /*      Copyright (c) 1984, 1986, 1987, 1988, 1989 AT&T */
27 /*      All Rights Reserved */
28 /*
29  * Copyright (c) 2013, Joyent, Inc. All rights reserved.
30 */

32 #include <sys/types.h>
33 #include <sys/param.h>
34 #include <sys/thread.h>
35 #include <sys/symmacros.h>
36 #include <sys/signal.h>
37 #include <sys/cred.h>
38 #include <sys/user.h>
39 #include <sys/errno.h>
40 #include <sys/vnode.h>
41 #include <sys/mman.h>
42 #include <sys/kmem.h>
43 #include <sys/proc.h>
44 #include <sys/pathname.h>
45 #include <sys/policy.h>
46 #endif /* ! codereview */
47 #include <sys/cmn_err.h>
48 #include <sys/system.h>
49 #include <sys/elf.h>
50 #include <sys/vmsystem.h>
51 #include <sys/debug.h>
52 #include <sys/auxv.h>
53 #include <sys/exec.h>
54 #include <sys/prsystem.h>
55 #include <vm/as.h>
56 #include <vm/rm.h>
57 #include <vm/seg.h>
58 #include <vm/seg_vn.h>

```

```

59 #include <sys/modctl.h>
60 #include <sys/systeminfo.h>
61 #include <sys/vmparam.h>
62 #include <sys/machelf.h>
63 #include <sys/shm_impl.h>
64 #include <sys/archsystem.h>
65 #include <sys/fasttrap.h>
66 #include <sys/brand.h>
67 #include "elf_impl.h"
68 #include <sys/sdt.h>
69 #include <sys/signinfo.h>
70 #include <sys/random.h>
71 #endif /* ! codereview */

73 extern int at_flags;
74 extern volatile size_t aslr_max_brk_skew;
75 #endif /* ! codereview */

77 #define ORIGIN_STR      "ORIGIN"
78 #define ORIGIN_STR_SIZE 6

80 static int getelfhead(vnode_t *, cred_t *, Ehdr *, int *, int *, int *);
81 static int getelfphdr(vnode_t *, cred_t *, const Ehdr *, int, caddr_t *,
82     ssize_t *);
83 static int getelfshdr(vnode_t *, cred_t *, const Ehdr *, int, int, caddr_t *,
84     ssize_t *, caddr_t *, ssize_t *);
85 static size_t elfsize(Ehdr *, int, caddr_t, uintptr_t *);
86 static int mapelfexec(vnode_t *, Ehdr *, int, caddr_t,
87     Phdr **, Phdr **, Phdr **, Phdr **, Phdr *,
88     caddr_t *, caddr_t *, intptr_t *, intptr_t *, size_t, long *, size_t *);

90 typedef enum {
91     STR_CTF,
92     STR_SYMTAB,
93     STR_DYNSYM,
94     STR_STRTAB,
95     STR_DYNSTR,
96     STR_SHSTRTAB,
97     STR_NUM
98 } shstrtype_t;

100 static const char *shstrtab_data[] = {
101     ".SUNW_ctf",
102     ".symtab",
103     ".dynsym",
104     ".strtab",
105     ".dynstr",
106     ".shstrtab"
107 };

109 typedef struct shstrtab {
110     int     sst_ndx[STR_NUM];
111     int     sst_cur;
112 } shstrtab_t;

114 static void
115 shstrtab_init(shstrtab_t *s)
116 {
117     bzero(&s->sst_ndx, sizeof (s->sst_ndx));
118     s->sst_cur = 1;
119 }

121 static int
122 shstrtab_ndx(shstrtab_t *s, shstrtype_t type)
123 {
124     int ret;

```

```

126     if ((ret = s->sst_ndx[type]) != 0)
127         return (ret);

129     ret = s->sst_ndx[type] = s->sst_cur;
130     s->sst_cur += strlen(shstrtab_data[type]) + 1;

132     return (ret);
133 }

135 static size_t
136 shstrtab_size(const shstrtab_t *s)
137 {
138     return (s->sst_cur);
139 }

141 static void
142 shstrtab_dump(const shstrtab_t *s, char *buf)
143 {
144     int i, ndx;

146     *buf = '\0';
147     for (i = 0; i < STR_NUM; i++) {
148         if ((ndx = s->sst_ndx[i]) != 0)
149             (void) strcpy(buf + ndx, shstrtab_data[i]);
150     }
151 }

153 static int
154 dtrace_safe_phdr(Phdr *phdrp, struct uarg *args, uintptr_t base)
155 {
156     ASSERT(phdrp->p_type == PT_SUNWDTRACE);

158     /*
159      * See the comment in fasttrap.h for information on how to safely
160      * update this program header.
161      */
162     if (phdrp->p_memsz < PT_SUNWDTRACE_SIZE ||
163         (phdrp->p_flags & (PF_R | PF_W | PF_X)) != (PF_R | PF_W | PF_X))
164         return (-1);

166     args->thrpтр = phdrp->p_vaddr + base;

168     return (0);
169 }

171 static int
172 handle_secflag_dt(proc_t *p, uint_t dt, uint_t val)
173 {
174     uint_t flag;

176     switch (dt) {
177     case DT_SUNW_ASLR:
178         flag = PROC_SEC_ASLR;
179         break;
180     default:
181         return (EINVAL);
182     }

184     if (val == 0) {
185         if (secflag_isset(p->p_secflags.psf_lower, flag))
186             return (EPERM);
187         if ((secpolicy_psecflags(CRED(), p, p) != 0) &&
188             secflag_isset(p->p_secflags.psf_inherit, flag))
189             return (EPERM);

```

```

191         secflag_clear(&p->p_secflags.psf_inherit, flag);
192         secflag_clear(&p->p_secflags.psf_effective, flag);
193     } else {
194         if (!secflag_isset(p->p_secflags.psf_upper, flag))
195             return (EPERM);

197         if ((secpolicy_psecflags(CRED(), p, p) != 0) &&
198             !secflag_isset(p->p_secflags.psf_inherit, flag))
199             return (EPERM);

201         secflag_set(&p->p_secflags.psf_inherit, flag);
202         secflag_set(&p->p_secflags.psf_effective, flag);
203     }

205     return (0);
206 }

208 #endif /* ! codereview */
209 /*
210  * Map in the executable pointed to by vp. Returns 0 on success.
211  */
212 int
213 mapexec_brand(vmode_t *vp, uarg_t *args, Ehdr *ehdr, Addr *uphdr_vaddr,
214               intptr_t *voffset, caddr_t exec_file, int *interp, caddr_t *bssbase,
215               caddr_t *brkbase, size_t *brksize, uintptr_t *lddatap)
216 {
217     size_t        len;
218     struct vattr  vat;
219     caddr_t       phdrbase = NULL;
220     ssize_t       phdrsize;
221     int           nshdrs, shstrndx, nphdrs;
222     int           error = 0;
223     Phdr          *uphdr = NULL;
224     Phdr          *junk = NULL;
225     Phdr          *dynphdr = NULL;
226     Phdr          *dtrphdr = NULL;
227     uintptr_t     lddata;
228     long          execsz;
229     intptr_t      minaddr;

231     if (lddatap != NULL)
232         *lddatap = NULL;

234     if (error = execpermissions(vp, &vat, args)) {
235         uprintf("%s: Cannot execute %s\n", exec_file, args->pathname);
236         return (error);
237     }

239     if ((error = getelfhead(vp, CRED(), ehdr, &nshdrs, &shstrndx,
240                           &nphdrs)) != 0 ||
241         (error = getelfphdr(vp, CRED(), ehdr, nphdrs, &phdrbase,
242                           &phdrsize)) != 0) {
243         uprintf("%s: Cannot read %s\n", exec_file, args->pathname);
244         return (error);
245     }

247     if ((len = elfsize(ehdr, nphdrs, phdrbase, &lddata)) == 0) {
248         uprintf("%s: Nothing to load in %s", exec_file, args->pathname);
249         kmem_free(phdrbase, phdrsize);
250         return (ENOEXEC);
251     }
252     if (lddatap != NULL)
253         *lddatap = lddata;

255     if (error = mapelfexec(vp, ehdr, nphdrs, phdrbase, &uphdr, &dynphdr,
256                           &junk, &dtrphdr, NULL, bssbase, brkbase, voffset, &minaddr,

```

```

257     len, &execsz, brksize) {
258         uprintf("%s: Cannot map %s\n", exec_file, args->pathname);
259         kmem_free(phdrbase, phdrsize);
260         return (error);
261     }
262
263     /*
264     * Inform our caller if the executable needs an interpreter.
265     */
266     *interp = (dynphdr == NULL) ? 0 : 1;
267
268     /*
269     * If this is a statically linked executable, voffset should indicate
270     * the address of the executable itself (it normally holds the address
271     * of the interpreter).
272     */
273     if (ehdr->e_type == ET_EXEC && *interp == 0)
274         *voffset = minaddr;
275
276     if (uphdr != NULL) {
277         *uphdr_vaddr = uphdr->p_vaddr;
278     } else {
279         *uphdr_vaddr = (Addr)-1;
280     }
281
282     kmem_free(phdrbase, phdrsize);
283     return (error);
284 }
285
286 /*ARGSUSED*/
287 int
288 elfexec(vnode_t *vp, execa_t *uap, uarg_t *args, intpdata_t *idatap,
289         int level, long *execsz, int setid, caddr_t exec_file, cred_t *cred,
290         int brand_action)
291 {
292     caddr_t     phdrbase = NULL;
293     caddr_t     bssbase = 0;
294     caddr_t     brkbase = 0;
295     size_t     brksize = 0;
296     ssize_t     dlntsize;
297     aux_entry_t *aux;
298     int         error;
299     ssize_t     resid;
300     int         fd = -1;
301     intptr_t   voffset;
302     Phdr       *intphdr = NULL;
303     Phdr       *dynamicphdr = NULL;
304     Phdr       *dyphdr = NULL;
305     Phdr       *stphdr = NULL;
306     Phdr       *uphdr = NULL;
307     Phdr       *junk = NULL;
308     size_t     len;
309     ssize_t     phdrsize;
310     int         postfixsize = 0;
311     int         i, hsize;
312     Phdr       *phdrp;
313     Phdr       *dataphdrp = NULL;
314     Phdr       *dtrphdr;
315     Phdr       *capphdr = NULL;
316     Cap        *cap = NULL;
317     size_t     capsiz;
318     Dyn        *dyn = NULL;
319 #endif /* ! codereview */
320     int         hasu = 0;
321     int         hasauxv = 0;
322     int         hasintp = 0;

```

```

59         int         hasdy = 0;
322         int         branded = 0;
323
324         struct proc *p = ttoproc(curthread);
325         struct user *up = PTOU(p);
326         struct bigwad {
327             Ehdr     ehdr;
328             aux_entry_t     elfargs[___KERN_NAUXV_IMPL];
329             char        dl_name[MAXPATHLEN];
330             char        pathbuf[MAXPATHLEN];
331             struct vattr     vattr;
332             struct execenv     exenv;
333         } *bigwad; /* kmem_alloc this behemoth so we don't blow stack */
334         Ehdr     *ehdrp;
335         int         nshdrs, shstrndx, nphdrs;
336         char        *dlnp;
337         char        *pathbufp;
338         rlim64_t     limit;
339         rlim64_t     roundlimit;
340
341         ASSERT(p->p_model == DATAMODEL_ILP32 || p->p_model == DATAMODEL_LP64);
342
343         bigwad = kmem_alloc(sizeof (struct bigwad), KM_SLEEP);
344         ehdrp = &bigwad->ehdr;
345         dlnp = bigwad->dln_name;
346         pathbufp = bigwad->pathbuf;
347
348         /*
349         * Obtain ELF and program header information.
350         */
351         if ((error = getelfhead(vp, CRED(), ehdrp, &nshdrs, &shstrndx,
352             &nphdrs)) != 0 ||
353             (error = getelfphdr(vp, CRED(), ehdrp, nphdrs, &phdrbase,
354             &phdrsize)) != 0)
355             goto out;
356
357         /*
358         * Prevent executing an ELF file that has no entry point.
359         */
360         if (ehdrp->e_entry == 0) {
361             uprintf("%s: Bad entry point\n", exec_file);
362             goto bad;
363         }
364
365         /*
366         * Put data model that we're exec-ing to into the args passed to
367         * exec_args(), so it will know what it is copying to on new stack.
368         * Now that we know whether we are exec-ing a 32-bit or 64-bit
369         * executable, we can set execsz with the appropriate NCARGS.
370         */
371 #ifdef _LP64
372         if (ehdrp->e_ident[EI_CLASS] == ELFCLASS32) {
373             args->to_model = DATAMODEL_ILP32;
374             *execsz = btopr(SINCR) + btopr(SSIZE) + btopr(NCARGS32-1);
375         } else {
376             args->to_model = DATAMODEL_LP64;
377             args->stk_prot &= ~PROT_EXEC;
378 #if defined(__i386) || defined(__amd64)
379             args->dat_prot &= ~PROT_EXEC;
380 #endif
381             *execsz = btopr(SINCR) + btopr(SSIZE) + btopr(NCARGS64-1);
382         }
383 #else /* _LP64 */
384         args->to_model = DATAMODEL_ILP32;
385         *execsz = btopr(SINCR) + btopr(SSIZE) + btopr(NCARGS-1);
386 #endif /* _LP64 */

```

```

388 /*
389  * We delay invoking the brand callback until we've figured out
390  * what kind of elf binary we're trying to run, 32-bit or 64-bit.
391  * We do this because now the brand library can just check
392  * args->to_model to see if the target is 32-bit or 64-bit without
393  * having to duplicate all the code above.
394  *
395  * The level checks associated with brand handling below are used to
396  * prevent a loop since the brand elfexec function typically comes back
397  * through this function. We must check <= here since the nested
398  * handling in the #! interpreter code will increment the level before
399  * calling gexec to run the final elfexec interpreter.
400  */
401 if ((level <= INTP_MAXDEPTH) &&
402     (brand_action != EBA_NATIVE) && (PROC_IS_BRANDED(p))) {
403     error = BROP(p)->b_elfexec(vp, uap, args,
404                               idatap, level + 1, execsz, setid, exec_file, cred,
405                               brand_action);
406     goto out;
407 }
408
409 /*
410  * Determine aux size now so that stack can be built
411  * in one shot (except actual copyout of aux image),
412  * determine any non-default stack protections,
413  * and still have this code be machine independent.
414  */
415 hsize = ehdrp->e_phentsize;
416 phdrp = (Phdr *)phdrbase;
417 for (i = nphdrs; i > 0; i--) {
418     switch (phdrp->p_type) {
419     case PT_INTERP:
420         hasauxv = hasintp = 1;
421         hasauxv = hasdy = 1;
422         break;
423     case PT_PHDR:
424         hasu = 1;
425         break;
426     case PT_SUNWSTACK:
427         args->stk_prot = PROT_USER;
428         if (phdrp->p_flags & PF_R)
429             args->stk_prot |= PROT_READ;
430         if (phdrp->p_flags & PF_W)
431             args->stk_prot |= PROT_WRITE;
432         if (phdrp->p_flags & PF_X)
433             args->stk_prot |= PROT_EXEC;
434         break;
435     case PT_LOAD:
436         dataphdrp = phdrp;
437         break;
438     case PT_SUNWCAP:
439         capphdr = phdrp;
440         break;
441     case PT_DYNAMIC:
442         dynamicphdr = phdrp;
443         break;
444     }
445     phdrp = (Phdr *)((caddr_t)phdrp + hsize);
446 }
447
448 if (ehdrp->e_type != ET_EXEC) {
449     dataphdrp = NULL;
450     hasauxv = 1;
451 }

```

```

453 /* Copy BSS permissions to args->dat_prot */
454 if (dataphdrp != NULL) {
455     args->dat_prot = PROT_USER;
456     if (dataphdrp->p_flags & PF_R)
457         args->dat_prot |= PROT_READ;
458     if (dataphdrp->p_flags & PF_W)
459         args->dat_prot |= PROT_WRITE;
460     if (dataphdrp->p_flags & PF_X)
461         args->dat_prot |= PROT_EXEC;
462 }
463
464 /*
465  * If a auxvector will be required - reserve the space for
466  * it now. This may be increased by exec_args if there are
467  * ISA-specific types (included in __KERN_NAUXV_IMPL).
468  */
469 if (hasauxv) {
470     /*
471      * If a AUX vector is being built - the base AUX
472      * entries are:
473      *
474      * AT_BASE
475      * AT_FLAGS
476      * AT_PAGESZ
477      * AT_SUN_AUXFLAGS
478      * AT_SUN_HWCAP
479      * AT_SUN_HWCAP2
480      * AT_SUN_PLATFORM (added in stk_copyout)
481      * AT_SUN_EXECNAME (added in stk_copyout)
482      * AT_NULL
483      *
484      * total == 9
485      */
486     if (hasintp && hasu) {
487         if (hasdy && hasu) {
488             /*
489              * Has PT_INTERP & PT_PHDR - the auxvectors that
490              * will be built are:
491              *
492              * AT_PHDR
493              * AT_PHENT
494              * AT_PHNUM
495              * AT_ENTRY
496              * AT_LDDATA
497              *
498              * total = 5
499              */
500             args->auxsize = (9 + 5) * sizeof (aux_entry_t);
501         } else if (hasintp) {
502             /*
503              * Has PT_INTERP but no PT_PHDR
504              *
505              * AT_EXECPD
506              * AT_LDDATA
507              *
508              * total = 2
509              */
510             args->auxsize = (9 + 2) * sizeof (aux_entry_t);
511         } else {
512             args->auxsize = 9 * sizeof (aux_entry_t);
513         }
514     } else {
515         args->auxsize = 0;
516     }
517 }

```

```

517 /*
518  * If this binary is using an emulator, we need to add an
519  * AT_SUN_EMULATOR aux entry.
520  */
521 if (args->emulator != NULL)
522     args->auxsize += sizeof (aux_entry_t);

524 if ((brand_action != EBA_NATIVE) && (PROC_IS_BRANDED(p))) {
525     branded = 1;
526     /*
527      * We will be adding 4 entries to the aux vectors. One for
528      * the the brandname and 3 for the brand specific aux vectors.
529      */
530     args->auxsize += 4 * sizeof (aux_entry_t);
531 }

533 /* If the binary has an explicit ASLR flag, it must be honoured */
534 if ((dynamicphdr != NULL) &&
535     (dynamicphdr->p_filesz > 0)) {
536     Dyn *dp;
537     off_t i = 0;

539 #define DYN_STRIDE    100
540     for (i = 0; i < dynamicphdr->p_filesz;
541          i += sizeof (*dyn) * DYN_STRIDE) {
542         int ndyns = (dynamicphdr->p_filesz - i) / sizeof (*dyn);
543         size_t dynsize;

545         ndyns = MIN(DYN_STRIDE, ndyns);
546         dynsize = ndyns * sizeof (*dyn);

548         dyn = kmem_alloc(dynsize, KM_SLEEP);

550         if ((error = vn_rdwr(UIO_READ, vp, (caddr_t)dyn,
551                             dynsize, (offset_t)(dynamicphdr->p_offset + i),
552                             UIO_SYSSPACE, 0, (rlim64_t)0,
553                             CRED(), &resid) != 0) {
554             uprintf("%s: cannot read .dynamic section\n",
555                     exec_file);
556             goto out;
557         }

559         for (dp = dyn; dp < (dyn + ndyns); dp++) {
560             if (dp->d_tag == DT_SUNW_ASRL) {
561                 if ((error = handle_secflag_dt(p,
562                                     DT_SUNW_ASRL,
563                                     dp->d_un.d_val)) != 0) {
564                     uprintf("%s: error setting "
565                             "security-flag from "
566                             "DT_SUNW_ASRL: %d\n",
567                             exec_file, error);
568                     goto out;
569                 }
570             }
571         }

573         kmem_free(dyn, dynsize);
574     }
575 }

577 #endif /* ! codereview */
578 /* Hardware/Software capabilities */
579 if (capphdr != NULL &&
580     (capsize = capphdr->p_filesz) > 0 &&
581     capsize <= 16 * sizeof (*cap)) {

```

```

582     int ncaps = capsize / sizeof (*cap);
583     Cap *cp;

585     cap = kmem_alloc(capsize, KM_SLEEP);
586     if ((error = vn_rdwr(UIO_READ, vp, (caddr_t)cap,
587                         capsize, (offset_t)capphdr->p_offset,
588                         UIO_SYSSPACE, 0, (rlim64_t)0, CRED(), &resid) != 0) {
589         uprintf("%s: Cannot read capabilities section\n",
590                 exec_file);
591         goto out;
592     }
593     for (cp = cap; cp < cap + ncaps; cp++) {
594         if (cp->c_tag == CA_SUNW_SF_1 &&
595             (cp->c_un.c_val & SF1_SUNW_ADDR32)) {
596             if (args->to_model == DATAMODEL_LP64)
597                 args->addr32 = 1;
598             break;
599         }
600     }
601 }

603 aux = bigwad->elfargs;
604 /*
605  * Move args to the user's stack.
606  * This can fill in the AT_SUN_PLATFORM and AT_SUN_EXECNAME aux entries.
607  */
608 if ((error = exec_args(uap, args, idatap, (void **)&aux)) != 0) {
609     if (error == -1) {
610         error = ENOEXEC;
611         goto bad;
612     }
613     goto out;
614 }
615 /* we're single threaded after this point */

617 /*
618  * If this is an ET_DYN executable (shared object),
619  * determine its memory size so that mapelfexec() can load it.
620  */
621 if (ehdrp->e_type == ET_DYN)
622     len = elfsize(ehdrp, nphdrs, phdrbase, NULL);
623 else
624     len = 0;

626 dtrphdr = NULL;

628 if ((error = mapelfexec(vp, ehdrp, nphdrs, phdrbase, &uphdr, &intphdr,
629                       if ((error = mapelfexec(vp, ehdrp, nphdrs, phdrbase, &uphdr, &dyphdr,
630                                             &stphdr, &dtrphdr, dataphdrp, &bssbase, &brkbase, &voffset, NULL,
631                                             len, excsz, &brksize)) != 0)
632                           goto bad;

633 if (uphdr != NULL && intphdr == NULL)
634     if (uphdr != NULL && dyphdr == NULL)
635         goto bad;

636 if (dtrphdr != NULL && dtrace_safe_phdr(dtrphdr, args, voffset) != 0) {
637     uprintf("%s: Bad DTrace phdr in %s\n", exec_file, exec_file);
638     goto bad;
639 }

641 if (intphdr != NULL) {
642     if (dyphdr != NULL) {
643         size_t len;
644         uintptr_t lddata;
645         char *p;

```

```

645     struct vnode      *nvp;
647     dlnsz = intphdr->p_filesz;
244     dlnsz = dyphdr->p_filesz;

649     if (dlnsz > MAXPATHLEN || dlnsz <= 0)
650         goto bad;

652     /*
653     * Read in "interpreter" pathname.
654     */
655     if ((error = vn_rdwr(UIO_READ, vp, dlnp, intphdr->p_filesz,
656         (offset_t)intphdr->p_offset, UIO_SYSSPACE, 0, (rlim64_t)0,
252     if ((error = vn_rdwr(UIO_READ, vp, dlnp, dyphdr->p_filesz,
253         (offset_t)dyphdr->p_offset, UIO_SYSSPACE, 0, (rlim64_t)0,
657         CRED(), &resid) != 0) {
658         uprintf("%s: Cannot obtain interpreter pathname\n",
659             exec_file);
660         goto bad;
661     }

663     if (resid != 0 || dlnp[dlnsz - 1] != '\0')
664         goto bad;

666     /*
667     * Search for '$ORIGIN' token in interpreter path.
668     * If found, expand it.
669     */
670     for (p = dlnp; p = strchr(p, '$'); ) {
671         uint_t len, curlen;
672         char    *_ptr;

674         if (strncmp(++p, ORIGIN_STR, ORIGIN_STR_SIZE))
675             continue;

677         /*
678         * We don't support $ORIGIN on setid programs to close
679         * a potential attack vector.
680         */
681         if ((setid & EXECSETID_SETID) != 0) {
682             error = ENOEXEC;
683             goto bad;
684         }

686         curlen = 0;
687         len = p - dlnp - 1;
688         if (len) {
689             bcopy(dlnp, pathbufp, len);
690             curlen += len;
691         }
692         if (_ptr = strrchr(args->pathname, '/')) {
693             len = _ptr - args->pathname;
694             if ((curlen + len) > MAXPATHLEN)
695                 break;

697             bcopy(args->pathname, &pathbufp[curlen], len);
698             curlen += len;
699         } else {
700             /*
701             * executable is a basename found in the
702             * current directory.  So - just substitute
703             * '.' for ORIGIN.
704             */
705             pathbufp[curlen] = '.';
706             curlen++;
707         }

```

```

708         p += ORIGIN_STR_SIZE;
709         len = strlen(p);

711         if ((curlen + len) > MAXPATHLEN)
712             break;
713         bcopy(p, &pathbufp[curlen], len);
714         curlen += len;
715         pathbufp[curlen++] = '\0';
716         bcopy(pathbufp, dlnp, curlen);
717     }

719     /*
720     * /usr/lib/ld.so.1 is known to be a symlink to /lib/ld.so.1
721     * (and /usr/lib/64/ld.so.1 is a symlink to /lib/64/ld.so.1).
722     * Just in case /usr is not mounted, change it now.
723     */
724     if (strcmp(dlnp, USR_LIB_RTLD) == 0)
725         dlnp += 4;
726     error = lookupname(dlnp, UIO_SYSSPACE, FOLLOW, NULLVPP, &nvp);
727     if (error && dlnp != bigwad->dl_name) {
728         /* new kernel, old user-level */
729         error = lookupname(dlnp - 4, UIO_SYSSPACE, FOLLOW,
730             NULLVPP, &nvp);
731     }
732     if (error) {
733         uprintf("%s: Cannot find %s\n", exec_file, dlnp);
734         goto bad;
735     }

737     /*
738     * Setup the "aux" vector.
739     */
740     if (uphdr) {
741         if (ehdrp->e_type == ET_DYN) {
742             /* don't use the first page */
743             bigwad->exenv.ex_brkbase = (caddr_t)PAGESIZE;
744             bigwad->exenv.ex_bssbase = (caddr_t)PAGESIZE;
745         } else {
746             bigwad->exenv.ex_bssbase = bssbase;
747             bigwad->exenv.ex_brkbase = brkbase;
748         }
749         bigwad->exenv.ex_brksize = brksize;
750         bigwad->exenv.ex_magic = elfmagic;
751         bigwad->exenv.ex_vp = vp;
752         setexecenv(&bigwad->exenv);

754         ADDAUX(aux, AT_PHDR, uphdr->p_vaddr + voffset)
755         ADDAUX(aux, AT_PHENT, ehdrp->e_phentsize)
756         ADDAUX(aux, AT_PHNUM, nphdrs)
757         ADDAUX(aux, AT_ENTRY, ehdrp->e_entry + voffset)
758     } else {
759         if ((error = execopen(&vp, &fd)) != 0) {
760             VN_RELE(nvp);
761             goto bad;
762         }

764         ADDAUX(aux, AT_EXECFD, fd)
765     }

767     if ((error = execpermissions(nvp, &bigwad->vattr, args)) != 0) {
768         VN_RELE(nvp);
769         uprintf("%s: Cannot execute %s\n", exec_file, dlnp);
770         goto bad;
771     }

773     /*

```

```

774     * Now obtain the ELF header along with the entire program
775     * header contained in "nvp".
776     */
777     kmem_free(phdrbase, phdrsize);
778     phdrbase = NULL;
779     if ((error = getelfhead(nvp, CRED(), ehdrp, &nshdrs,
780         &shstrndx, &nphdrs)) != 0 ||
781         (error = getelfphdr(nvp, CRED(), ehdrp, nphdrs, &phdrbase,
782             &phdrsize)) != 0) {
783         VN_RELE(nvp);
784         uprintf("%s: Cannot read %s\n", exec_file, dlnp);
785         goto bad;
786     }
787
788     /*
789     * Determine memory size of the "interpreter's" loadable
790     * sections. This size is then used to obtain the virtual
791     * address of a hole, in the user's address space, large
792     * enough to map the "interpreter".
793     */
794     if ((len = elfsize(ehdrp, nphdrs, phdrbase, &lddata)) == 0) {
795         VN_RELE(nvp);
796         uprintf("%s: Nothing to load in %s\n", exec_file, dlnp);
797         goto bad;
798     }
799
800     dtrphdr = NULL;
801
802     error = mapelfexec(nvp, ehdrp, nphdrs, phdrbase, &junk, &junk,
803         &junk, &dtrphdr, NULL, NULL, NULL, &voffset, NULL, len,
804         execsz, NULL);
805     if (error || junk != NULL) {
806         VN_RELE(nvp);
807         uprintf("%s: Cannot map %s\n", exec_file, dlnp);
808         goto bad;
809     }
810
811     /*
812     * We use the DTrace program header to initialize the
813     * architecture-specific user per-LWP location. The dtrace
814     * fasttrap provider requires ready access to per-LWP scratch
815     * space. We assume that there is only one such program header
816     * in the interpreter.
817     */
818     if (dtrphdr != NULL &&
819         dtrace_safe_phdr(dtrphdr, args, voffset) != 0) {
820         VN_RELE(nvp);
821         uprintf("%s: Bad DTrace phdr in %s\n", exec_file, dlnp);
822         goto bad;
823     }
824
825     VN_RELE(nvp);
826     ADDAUX(aux, AT_SUN_LDDATA, voffset + lddata)
827 }
828
829 if (hasauxv) {
830     int auxf = AF_SUN_HWCAPVERIFY;
831     /*
832     * Note: AT_SUN_PLATFORM and AT_SUN_EXECNAME were filled in via
833     * exec_args()
834     */
835     ADDAUX(aux, AT_BASE, voffset)
836     ADDAUX(aux, AT_FLAGS, at_flags)
837     ADDAUX(aux, AT_PAGESZ, PAGESIZE)
838     /*
839     * Linker flags. (security)

```

```

840     * p_flag not yet set at this time.
841     * We rely on gexec() to provide us with the information.
842     * If the application is set-uid but this is not reflected
843     * in a mismatch between real/effective uids/gids, then
844     * don't treat this as a set-uid exec. So we care about
845     * the EXECSETID_UGIDS flag but not the ...SETID flag.
846     */
847     if ((setid &= ~EXECSETID_SETID) != 0)
848         auxf |= AF_SUN_SETUGID;
849
850     /*
851     * If we're running a native process from within a branded
852     * zone under pfexec then we clear the AF_SUN_SETUGID flag so
853     * that the native ld.so.1 is able to link with the native
854     * libraries instead of using the brand libraries that are
855     * installed in the zone. We only do this for processes
856     * which we trust because we see they are already running
857     * under pfexec (where uid != euid). This prevents a
858     * malicious user within the zone from crafting a wrapper to
859     * run native suid commands with unsecure libraries interposed.
860     */
861     if ((brand_action == EBA_NATIVE) && (PROC_IS_BRANDED(p) &&
862         (setid &= ~EXECSETID_SETID) != 0))
863         auxf &= ~AF_SUN_SETUGID;
864
865     /*
866     * Record the user addr of the auxflags aux vector entry
867     * since brands may optionally want to manipulate this field.
868     */
869     args->auxp_auxflags =
870         (char *)((char *)args->stackend +
871             ((char *)&aux->a_type -
872             (char *)bigwad->elfargs));
873     ADDAUX(aux, AT_SUN_AUXFLAGS, auxf);
874
875 #endif /* ! codereview */
876     /*
877     * Hardware capability flag word (performance hints)
878     * Used for choosing faster library routines.
879     * (Potentially different between 32-bit and 64-bit ABIs)
880     */
881     #if defined(_LP64)
882     if (args->to_model == DATAMODEL_NATIVE) {
883         ADDAUX(aux, AT_SUN_HWCAP, auxv_hwcap)
884         ADDAUX(aux, AT_SUN_HWCAP2, auxv_hwcap_2)
885     } else {
886         ADDAUX(aux, AT_SUN_HWCAP, auxv_hwcap32)
887         ADDAUX(aux, AT_SUN_HWCAP2, auxv_hwcap32_2)
888     }
889 #else
890     ADDAUX(aux, AT_SUN_HWCAP, auxv_hwcap)
891     ADDAUX(aux, AT_SUN_HWCAP2, auxv_hwcap_2)
892 #endif
893 #endif
894     if (branded) {
895         /*
896         * Reserve space for the brand-private aux vectors,
897         * and record the user addr of that space.
898         */
899         args->auxp_brand =
900             (char *)((char *)args->stackend +
901                 ((char *)&aux->a_type -
902                 (char *)bigwad->elfargs));
903         ADDAUX(aux, AT_SUN_BRAND_AUX1, 0)
904         ADDAUX(aux, AT_SUN_BRAND_AUX2, 0)
905         ADDAUX(aux, AT_SUN_BRAND_AUX3, 0)

```

```

907      ADDAUX(aux, AT_NULL, 0)
908      postfixsize = (char *)aux - (char *)bigwad->elfargs;

910      /*
911       * We make assumptions above when we determine how many aux
912       * vector entries we will be adding. However, if we have an
913       * invalid elf file, it is possible that mapelfexec might
914       * behave differently (but not return an error), in which case
915       * the number of aux entries we actually add will be different.
916       * We detect that now and error out.
917       */
918      if (postfixsize != args->auxsize) {
919          DTRACE_PROBE2(elfexec_badaux, int, postfixsize,
920                      int, args->auxsize);
921          goto bad;
922      }
923      ASSERT(postfixsize <= __KERN_NAUXV_IMPL * sizeof (aux_entry_t));
924  }

926  /*
927   * For the 64-bit kernel, the limit is big enough that rounding it up
928   * to a page can overflow the 64-bit limit, so we check for btopr()
929   * overflowing here by comparing it with the unrounded limit in pages.
930   * If it hasn't overflowed, compare the exec size with the rounded up
931   * limit in pages. Otherwise, just compare with the unrounded limit.
932   */
933  limit = btopr(p->p_vmem_ctl);
934  roundlimit = btopr(p->p_vmem_ctl);
935  if ((roundlimit > limit && *execsz > roundlimit) ||
936      (roundlimit < limit && *execsz > limit)) {
937      mutex_enter(&p->p_lock);
938      (void) rctl_action(rctlproc_legacy[RLIMIT_VMEM], p->p_rctls, p,
939                      RCA_SAFE);
940      mutex_exit(&p->p_lock);
941      error = ENOMEM;
942      goto bad;
943  }

945  bzero(up->u_auxv, sizeof (up->u_auxv));
946  if (postfixsize) {
947      int num_auxv;

949      /*
950       * Copy the aux vector to the user stack.
951       */
952      error = execpoststack(args, bigwad->elfargs, postfixsize);
953      if (error)
954          goto bad;

956      /*
957       * Copy auxv to the process's user structure for use by /proc.
958       * If this is a branded process, the brand's exec routine will
959       * copy it's private entries to the user structure later. It
960       * relies on the fact that the blank entries are at the end.
961       */
962      num_auxv = postfixsize / sizeof (aux_entry_t);
963      ASSERT(num_auxv <= sizeof (up->u_auxv) / sizeof (auxv_t));
964      aux = bigwad->elfargs;
965      for (i = 0; i < num_auxv; i++) {
966          up->u_auxv[i].a_type = aux[i].a_type;
967          up->u_auxv[i].a_un.a_val = (aux_val_t)aux[i].a_un.a_val;
968      }
969  }

971  /*

```

```

972      * Pass back the starting address so we can set the program counter.
973      */
974      args->entry = (uintptr_t)(ehdrp->e_entry + voffset);

976      if (!phdr) {
977          if (ehdrp->e_type == ET_DYN) {
978              /*
979               * If we are executing a shared library which doesn't
980               * have a interpreter (probably ld.so.1) then
981               * we don't set the brkbase now. Instead we
982               * delay it's setting until the first call
983               * via grow.c::brk(). This permits ld.so.1 to
984               * initialize brkbase to the tail of the executable it
985               * loads (which is where it needs to be).
986               */
987              bigwad->exenv.ex_brkbase = (caddr_t)0;
988              bigwad->exenv.ex_bssbase = (caddr_t)0;
989              bigwad->exenv.ex_brksize = 0;
990          } else {
991              bigwad->exenv.ex_brkbase = brkbase;
992              bigwad->exenv.ex_bssbase = bssbase;
993              bigwad->exenv.ex_brksize = brksize;
994          }
995          bigwad->exenv.ex_magic = elfmagic;
996          bigwad->exenv.ex_vp = vp;
997          setexecenv(&bigwad->exenv);
998      }

1000     ASSERT(error == 0);
1001     goto out;

1003 bad:
1004     if (fd != -1) /* did we open the a.out yet */
1005         (void) execclose(fd);

1007     psignal(p, SIGKILL);

1009     if (error == 0)
1010         error = ENOEXEC;

1011 out:
1012     if (phdrbase != NULL)
1013         kmem_free(phdrbase, phdrsize);
1014     if (cap != NULL)
1015         kmem_free(cap, capsize);
1016     kmem_free(bigwad, sizeof (struct bigwad));
1017     return (error);
1018 }

1020 /*
1021  * Compute the memory size requirement for the ELF file.
1022  */
1023 static size_t
1024 elfsize(Ehdr *ehdrp, int nphdrs, caddr_t phdrbase, uintptr_t *ldata)
1025 {
1026     size_t len;
1027     Phdr *phdrp = (Phdr *)phdrbase;
1028     int hsize = ehdrp->e_phentsize;
1029     int first = 1;
1030     int dfirst = 1; /* first data segment */
1031     uintptr_t loadaddr = 0;
1032     uintptr_t hiaddr = 0;
1033     uintptr_t lo, hi;
1034     int i;

1036     for (i = nphdrs; i > 0; i--) {
1037         if (phdrp->p_type == PT_LOAD) {

```



```

1038     lo = phdrp->p_vaddr;
1039     hi = lo + phdrp->p_memsz;
1040     if (first) {
1041         loadaddr = lo;
1042         hiaddr = hi;
1043         first = 0;
1044     } else {
1045         if (loadaddr > lo)
1046             loadaddr = lo;
1047         if (hiaddr < hi)
1048             hiaddr = hi;
1049     }
1051     /*
1052     * save the address of the first data segment
1053     * of a object - used for the AT_SUNW_LDDATA
1054     * aux entry.
1055     */
1056     if ((lddata != NULL) && dfirst &&
1057         (phdrp->p_flags & PF_W)) {
1058         *lddata = lo;
1059         dfirst = 0;
1060     }
1061     }
1062     phdrp = (Phdr *)((caddr_t)phdrp + hsize);
1063 }
1065 len = hiaddr - (loadaddr & PAGEMASK);
1066 len = roundup(len, PAGESIZE);
1068 return (len);
1069 }
1071 /*
1072 * Read in the ELF header and program header table.
1073 * SUSV3 requires:
1074 * ENOEXEC File format is not recognized
1075 * EINVAL Format recognized but execution not supported
1076 */
1077 static int
1078 getelfhead(vnode_t *vp, cred_t *credp, Ehdr *ehdr, int *nshdrs, int *shstrndx,
1079            int *nphdrs)
1080 {
1081     int error;
1082     ssize_t resid;
1084     /*
1085     * We got here by the first two bytes in ident,
1086     * now read the entire ELF header.
1087     */
1088     if ((error = vn_rdwr(UIO_READ, vp, (caddr_t)ehdr,
1089         sizeof (Ehdr), (offset_t)0, UIO_SYSSPACE, 0,
1090         (rlim64_t)0, credp, &resid)) != 0)
1091         return (error);
1093     /*
1094     * Since a separate version is compiled for handling 32-bit and
1095     * 64-bit ELF executables on a 64-bit kernel, the 64-bit version
1096     * doesn't need to be able to deal with 32-bit ELF files.
1097     */
1098     if (resid != 0 ||
1099         ehdr->e_ident[EI_MAG2] != ELF_MAG2 ||
1100         ehdr->e_ident[EI_MAG3] != ELF_MAG3)
1101         return (ENOEXEC);
1103     if ((ehdr->e_type != ET_EXEC && ehdr->e_type != ET_DYN) ||

```

```

1104     #if defined(_ILP32) || defined(_ELF32_COMPAT)
1105         ehdr->e_ident[EI_CLASS] != ELFCLASS32 ||
1106     #else
1107         ehdr->e_ident[EI_CLASS] != ELFCLASS64 ||
1108     #endif
1109         !elfheadcheck(ehdr->e_ident[EI_DATA], ehdr->e_machine,
1110         ehdr->e_flags))
1111         return (EINVAL);
1113     *nshdrs = ehdr->e_shnum;
1114     *shstrndx = ehdr->e_shstrndx;
1115     *nphdrs = ehdr->e_phnum;
1117     /*
1118     * If e_shnum, e_shstrndx, or e_phnum is its sentinel value, we need
1119     * to read in the section header at index zero to access the true
1120     * values for those fields.
1121     */
1122     if ((*nshdrs == 0 && ehdr->e_shoff != 0) ||
1123         *shstrndx == SHN_XINDEX || *nphdrs == PN_XNUM) {
1124         Shdr shdr;
1126         if (ehdr->e_shoff == 0)
1127             return (EINVAL);
1129         if ((error = vn_rdwr(UIO_READ, vp, (caddr_t)&shdr,
1130             sizeof (shdr), (offset_t)ehdr->e_shoff, UIO_SYSSPACE, 0,
1131             (rlim64_t)0, credp, &resid)) != 0)
1132             return (error);
1134         if (*nshdrs == 0)
1135             *nshdrs = shdr.sh_size;
1136         if (*shstrndx == SHN_XINDEX)
1137             *shstrndx = shdr.sh_link;
1138         if (*nphdrs == PN_XNUM && shdr.sh_info != 0)
1139             *nphdrs = shdr.sh_info;
1140     }
1142     return (0);
1143 }
1145 #ifndef _ELF32_COMPAT
1146     extern size_t elf_nphdr_max;
1147 #else
1148     size_t elf_nphdr_max = 1000;
1149 #endif
1151 static int
1152 getelfphdr(vnode_t *vp, cred_t *credp, const Ehdr *ehdr, int nphdrs,
1153            caddr_t *phbasep, ssize_t *phsizep)
1154 {
1155     ssize_t resid, minsize;
1156     int err;
1158     /*
1159     * Since we're going to be using e_phentsize to iterate down the
1160     * array of program headers, it must be 8-byte aligned or else
1161     * a we might cause a misaligned access. We use all members through
1162     * p_flags on 32-bit ELF files and p_memsz on 64-bit ELF files so
1163     * e_phentsize must be at least large enough to include those
1164     * members.
1165     */
1166     #if !defined(_LP64) || defined(_ELF32_COMPAT)
1167         minsize = offsetof(Phdr, p_flags) + sizeof (((Phdr *)NULL)->p_flags);
1168     #else
1169         minsize = offsetof(Phdr, p_memsz) + sizeof (((Phdr *)NULL)->p_memsz);

```

```

1170 #endif
1171     if (ehdr->e_phentsize < minsize || (ehdr->e_phentsize & 3))
1172         return (EINVAL);
1174     *phsizep = nphdrs * ehdr->e_phentsize;
1176     if (*phsizep > sizeof (Phdr) * elf_nphdr_max) {
1177         if ((*phbasep = kmem_alloc(*phsizep, KM_NOSLEEP)) == NULL)
1178             return (ENOMEM);
1179     } else {
1180         *phbasep = kmem_alloc(*phsizep, KM_SLEEP);
1181     }
1183     if ((err = vn_rdwr(UIO_READ, vp, *phbasep, *phsizep,
1184         (offset_t)ehdr->e_phoff, UIO_SYSSPACE, 0, (rlim64_t)0,
1185         credp, &resid) != 0) {
1186         kmem_free(*phbasep, *phsizep);
1187         *phbasep = NULL;
1188         return (err);
1189     }
1191     return (0);
1192 }
1194 #ifdef _ELF32_COMPAT
1195 extern size_t elf_nshdr_max;
1196 extern size_t elf_shstrtab_max;
1197 #else
1198 size_t elf_nshdr_max = 10000;
1199 size_t elf_shstrtab_max = 100 * 1024;
1200 #endif
1203 static int
1204 getelfshdr(vnode_t *vp, cred_t *credp, const Ehdr *ehdr,
1205     int nshdrs, int shstrndx, caddr_t *shbasep, ssize_t *shsizep,
1206     char **shstrbasep, ssize_t *shstrsizep)
1207 {
1208     ssize_t resid, minsize;
1209     int err;
1210     Shdr *shdr;
1212     /*
1213     * Since we're going to be using e_shentsize to iterate down the
1214     * array of section headers, it must be 8-byte aligned or else
1215     * a we might cause a misaligned access. We use all members through
1216     * sh_entsize (on both 32- and 64-bit ELF files) so e_shentsize
1217     * must be at least large enough to include that member. The index
1218     * of the string table section must also be valid.
1219     */
1220     minsize = offsetof(Shdr, sh_entsize) + sizeof (shdr->sh_entsize);
1221     if (ehdr->e_shentsize < minsize || (ehdr->e_shentsize & 3) ||
1222         shstrndx >= nshdrs)
1223         return (EINVAL);
1225     *shsizep = nshdrs * ehdr->e_shentsize;
1227     if (*shsizep > sizeof (Shdr) * elf_nshdr_max) {
1228         if ((*shbasep = kmem_alloc(*shsizep, KM_NOSLEEP)) == NULL)
1229             return (ENOMEM);
1230     } else {
1231         *shbasep = kmem_alloc(*shsizep, KM_SLEEP);
1232     }
1234     if ((err = vn_rdwr(UIO_READ, vp, *shbasep, *shsizep,
1235         (offset_t)ehdr->e_shoff, UIO_SYSSPACE, 0, (rlim64_t)0,

```

```

1236         credp, &resid) != 0) {
1237             kmem_free(*shbasep, *shsizep);
1238             return (err);
1239         }
1241     /*
1242     * Pull the section string table out of the vnode; fail if the size
1243     * is zero.
1244     */
1245     shdr = (Shdr *)(*shbasep + shstrndx * ehdr->e_shentsize);
1246     if ((*shstrsizep = shdr->sh_size) == 0) {
1247         kmem_free(*shbasep, *shsizep);
1248         return (EINVAL);
1249     }
1251     if (*shstrsizep > elf_shstrtab_max) {
1252         if ((*shstrbasep = kmem_alloc(*shstrsizep,
1253             KM_NOSLEEP)) == NULL) {
1254             kmem_free(*shbasep, *shsizep);
1255             return (ENOMEM);
1256         }
1257     } else {
1258         *shstrbasep = kmem_alloc(*shstrsizep, KM_SLEEP);
1259     }
1261     if ((err = vn_rdwr(UIO_READ, vp, *shstrbasep, *shstrsizep,
1262         (offset_t)shdr->sh_offset, UIO_SYSSPACE, 0, (rlim64_t)0,
1263         credp, &resid) != 0) {
1264         kmem_free(*shbasep, *shsizep);
1265         kmem_free(*shstrbasep, *shstrsizep);
1266         return (err);
1267     }
1269     /*
1270     * Make sure the strtab is null-terminated to make sure we
1271     * don't run off the end of the table.
1272     */
1273     (*shstrbasep)[*shstrsizep - 1] = '\0';
1275     return (0);
1276 }
1278 static int
1279 mapelfexec(
1280     vnode_t *vp,
1281     Ehdr *ehdr,
1282     int nphdrs,
1283     caddr_t phdrbase,
1284     Phdr **uphdr,
1285     Phdr **intphdr,
1286     Phdr **dyphdr,
1287     Phdr **stphdr,
1288     Phdr **dtphdr,
1289     Phdr *dataphdrp,
1290     caddr_t *bssbase,
1291     caddr_t *brkbase,
1292     intptr_t *voffset,
1293     intptr_t *minaddr,
1294     size_t len,
1295     long *execsz,
1296     size_t *brksize)
1297 {
1298     Phdr *phdr;
1299     int i, prot, error;
1300     caddr_t addr = NULL;
1301     size_t zfodsz;

```

```

1301     int pload = 0;
1302     int page;
1303     off_t offset;
1304     int hsize = ehdr->e_phentsize;
1305     caddr_t mintmp = (caddr_t)-1;
1306     extern int use_brk_lpg;

1308     if (ehdr->e_type == ET_DYN) {
1309         secflagset_t flags = 0;
1310 #endif /* ! codereview */
1311         /*
1312          * Obtain the virtual address of a hole in the
1313          * address space to map the "interpreter".
1314          */
1315         if (secflag_enabled(curproc, PROC_SEC_ASLR))
1316             flags |= _MAP_RANDOMIZE;

1318         map_addr(&addr, len, (offset_t)0, 1, flags);
1319         map_addr(&addr, len, (offset_t)0, 1, 0);
1320         if (addr == NULL)
1321             return (ENOMEM);
1322         *voffset = (intptr_t)addr;

1323     /*
1324      * Calculate the minimum vaddr so it can be subtracted out.
1325      * According to the ELF specification, since PT_LOAD sections
1326      * must be sorted by increasing p_vaddr values, this is
1327      * guaranteed to be the first PT_LOAD section.
1328      */
1329     phdr = (Phdr *)phdrbase;
1330     for (i = nphdrs; i > 0; i--) {
1331         if (phdr->p_type == PT_LOAD) {
1332             *voffset -= (uintptr_t)phdr->p_vaddr;
1333             break;
1334         }
1335         phdr = (Phdr *)((caddr_t)phdr + hsize);
1336     }

1338 } else {
1339     *voffset = 0;
1340 }
1341 phdr = (Phdr *)phdrbase;
1342 for (i = nphdrs; i > 0; i--) {
1343     switch (phdr->p_type) {
1344     case PT_LOAD:
1345         if ((*intphdr != NULL) && (*uphdr == NULL))
1346             if ((*dyphdr != NULL) && (*uphdr == NULL))
1347                 return (0);

1348         pload = 1;
1349         prot = PROT_USER;
1350         if (phdr->p_flags & PF_R)
1351             prot |= PROT_READ;
1352         if (phdr->p_flags & PF_W)
1353             prot |= PROT_WRITE;
1354         if (phdr->p_flags & PF_X)
1355             prot |= PROT_EXEC;

1357         addr = (caddr_t)((uintptr_t)phdr->p_vaddr + *voffset);

1359     /*
1360      * Keep track of the segment with the lowest starting
1361      * address.
1362      */
1363     if (addr < mintmp)
1364         mintmp = addr;

```

```

1366         zfodsz = (size_t)phdr->p_memsz - phdr->p_filesz;

1368     offset = phdr->p_offset;
1369     if (((uintptr_t)offset & PAGEOFFSET) ==
1370         ((uintptr_t)addr & PAGEOFFSET) &&
1371         (!(vp->v_flag & VNOMAP))) {
1372         page = 1;
1373     } else {
1374         page = 0;
1375     }

1377     /*
1378      * Set the heap pagesize for OOB when the bss size
1379      * is known and use_brk_lpg is not 0.
1380      */
1381     if (brksize != NULL && use_brk_lpg &&
1382         zfodsz != 0 && phdr == dataphdrp &&
1383         (prot & PROT_WRITE)) {
1384         size_t tlen = P2NPHASE((uintptr_t)addr +
1385             phdr->p_filesz, PAGESIZE);

1387         if (zfodsz > tlen) {
1388             curproc->p_brkpagesz =
1389                 page_szc(map_pgsz(MAPPGSZ_HEAP,
1390                     curproc, addr + phdr->p_filesz +
1391                     tlen, zfodsz - tlen, 0));
1392         }
1393     }

1395     if (curproc->p_brkpagesz != 0 && phdr == dataphdrp &&
1396         (prot & PROT_WRITE)) {
1397         uint_t szc = curproc->p_brkpagesz;
1398         size_t pgsz = page_get_pagesize(szc);
1399         caddr_t ebss = addr + phdr->p_memsz;
1400         /*
1401          * If we need extra space to keep the BSS an
1402          * integral number of pages in size, some of
1403          * that space may fall beyond p_brkbase, so we
1404          * need to set p_brksize to account for it
1405          * being (logically) part of the brk.
1406          */
1407 #endif /* ! codereview */
1408         size_t extra_zfodsz;

1410         ASSERT(pgsz > PAGESIZE);

1412         extra_zfodsz = P2NPHASE((uintptr_t)ebss, pgsz);

1414         if (error = execmap(vp, addr, phdr->p_filesz,
1415             zfodsz + extra_zfodsz, phdr->p_offset,
1416             prot, page, szc))
1417             goto bad;
1418         if (brksize != NULL)
1419             *brksize = extra_zfodsz;
1420     } else {
1421         if (error = execmap(vp, addr, phdr->p_filesz,
1422             zfodsz, phdr->p_offset, prot, page, 0))
1423             goto bad;
1424     }

1426     if (bssbase != NULL && addr >= *bssbase &&
1427         phdr == dataphdrp) {
1428         *bssbase = addr + phdr->p_filesz;
1429     }
1430     if (brkbase != NULL && addr >= *brkbase) {

```

```

1431         *brkbase = addr + phdr->p_memsz;
1432     }
1434     *execsz += btopr(phdr->p_memsz);
1435     break;
1437     case PT_INTERP:
1438         if (pload)
1439             goto bad;
1440         *intphdr = phdr;
1441         *dyphdr = phdr;
1442         break;
1443     case PT_SHLIB:
1444         *stphdr = phdr;
1445         break;
1447     case PT_PHDR:
1448         if (pload)
1449             goto bad;
1450         *uphdr = phdr;
1451         break;
1453     case PT_NULL:
1454     case PT_DYNAMIC:
1455     case PT_NOTE:
1456         break;
1458     case PT_SUNWDTRACE:
1459         if (dtphdr != NULL)
1460             *dtphdr = phdr;
1461         break;
1463     default:
1464         break;
1465     }
1466     phdr = (Phdr *)((caddr_t)phdr + hsize);
1467 }
1469 if (minaddr != NULL) {
1470     ASSERT(mintmp != (caddr_t)-1);
1471     *minaddr = (intptr_t)mintmp;
1472 }
1474 if (brkbase != NULL && secflag_enabled(curproc, PROC_SEC_ASLR)) {
1475     size_t off;
1476     uintptr_t base = (uintptr_t)*brkbase;
1477     uintptr_t oend = base + *brksize;
1479     ASSERT(ISP2(aslr_max_brk_skew));
1481     (void) random_get_pseudo_bytes((uint8_t *)&off, sizeof(off));
1482     base += P2PHASE(off, aslr_max_brk_skew);
1483     base = P2ROUNDUP(base, PAGE_SIZE);
1484     *brkbase = (caddr_t)base;
1485     /*
1486     * Above, we set *brksize to account for the possibility we
1487     * had to grow the 'brk' in padding out the BSS to a page
1488     * boundary.
1489     *
1490     * We now need to adjust that based on where we now are
1491     * actually putting the brk.
1492     */
1493     if (oend > base)
1494         *brksize = oend - base;
1495     else

```

```

1496         *brksize = 0;
1497     }
1499 #endif /* ! codereview */
1500     return (0);
1501 bad:
1502     if (error == 0)
1503         error = EINVAL;
1504     return (error);
1505 }
1507 int
1508 elfnote(vnode_t *vp, offset_t *offsetp, int type, int descsz, void *desc,
1509         rlim64_t rlimit, cred_t *credp)
1510 {
1511     Note note;
1512     int error;
1514     bzero(&note, sizeof(note));
1515     bcopy("CORE", note.name, 4);
1516     note.nhdr.n_type = type;
1517     /*
1518     * The System V ABI states that n_namesz must be the length of the
1519     * string that follows the Nhdr structure including the terminating
1520     * null. The ABI also specifies that sufficient padding should be
1521     * included so that the description that follows the name string
1522     * begins on a 4- or 8-byte boundary for 32- and 64-bit binaries
1523     * respectively. However, since this change was not made correctly
1524     * at the time of the 64-bit port, both 32- and 64-bit binaries
1525     * descriptions are only guaranteed to begin on a 4-byte boundary.
1526     */
1527     note.nhdr.n_namesz = 5;
1528     note.nhdr.n_descsz = roundup(descsz, sizeof(Word));
1530     if (error = core_write(vp, UIO_SYSSPACE, *offsetp, &note,
1531         sizeof(note), rlimit, credp))
1532         return (error);
1534     *offsetp += sizeof(note);
1536     if (error = core_write(vp, UIO_SYSSPACE, *offsetp, desc,
1537         note.nhdr.n_descsz, rlimit, credp))
1538         return (error);
1540     *offsetp += note.nhdr.n_descsz;
1541     return (0);
1542 }
1544 /*
1545  * Copy the section data from one vnode to the section of another vnode.
1546  */
1547 static void
1548 copy_scn(Shdr *src, vnode_t *src_vp, Shdr *dst, vnode_t *dst_vp, Off *doffset,
1549         void *buf, size_t size, cred_t *credp, rlim64_t rlimit)
1550 {
1551     ssize_t resid;
1552     size_t len, n = src->sh_size;
1553     offset_t off = 0;
1555     while (n != 0) {
1556         len = MIN(size, n);
1557         if (vn_rdwr(UIO_READ, src_vp, buf, len, src->sh_offset + off,
1558             UIO_SYSSPACE, 0, (rlim64_t)0, credp, &resid) != 0 ||
1559             resid >= len ||
1560             core_write(dst_vp, UIO_SYSSPACE, *doffset + off,
1561                 buf, len - resid, rlimit, credp) != 0) {

```

```

1562         dst->sh_size = 0;
1563         dst->sh_offset = 0;
1564         return;
1565     }
1567     ASSERT(n >= len - resid);
1569     n -= len - resid;
1570     off += len - resid;
1571 }
1573     *doffset += src->sh_size;
1574 }
1576 #ifdef _ELF32_COMPAT
1577 extern size_t elf_datasz_max;
1578 #else
1579 size_t elf_datasz_max = 1 * 1024 * 1024;
1580 #endif
1582 /*
1583  * This function processes mappings that correspond to load objects to
1584  * examine their respective sections for elfcore(). It's called once with
1585  * v set to NULL to count the number of sections that we're going to need
1586  * and then again with v set to some allocated buffer that we fill in with
1587  * all the section data.
1588  */
1589 static int
1590 process_scns(core_content_t content, proc_t *p, cred_t *credp, vnode_t *vp,
1591             shdr *v, int nv, rlim64_t rlimit, Off *doffsetp, int *nshdrsp)
1592 {
1593     vnode_t *lastvp = NULL;
1594     struct seg *seg;
1595     int i, j;
1596     void *data = NULL;
1597     size_t datasz = 0;
1598     shstrtab_t shstrtab;
1599     struct as *as = p->p_as;
1600     int error = 0;
1602     if (v != NULL)
1603         shstrtab_init(&shstrtab);
1605     i = 1;
1606     for (seg = AS_SEGFIRST(as); seg != NULL; seg = AS_SEGNEXT(as, seg)) {
1607         uint_t prot;
1608         vnode_t *mvp;
1609         void *tmp = NULL;
1610         caddr_t saddr = seg->s_base;
1611         caddr_t naddr;
1612         caddr_t eaddr;
1613         size_t segsize;
1615         Ehdr ehdr;
1616         int nshdrs, shstrndx, nphdrs;
1617         caddr_t shbase;
1618         ssize_t shsize;
1619         char *shstrbase;
1620         ssize_t shstrsize;
1622         Shdr *shdr;
1623         const char *name;
1624         size_t sz;
1625         uintptr_t off;
1627         int ctf_ndx = 0;

```

```

1628         int symtab_ndx = 0;
1630         /*
1631          * Since we're just looking for text segments of load
1632          * objects, we only care about the protection bits; we don't
1633          * care about the actual size of the segment so we use the
1634          * reserved size. If the segment's size is zero, there's
1635          * something fishy going on so we ignore this segment.
1636          */
1637         if (seg->s_ops != &segvn_ops ||
1638             SEGOP_GETVP(seg, seg->s_base, &mvp) != 0 ||
1639             mvp == lastvp || mvp == NULL || mvp->v_type != VREG ||
1640             (segsize = pr_getsegsz(seg, 1)) == 0)
1641             continue;
1643         eaddr = saddr + segsize;
1644         prot = pr_getprot(seg, 1, &tmp, &saddr, &naddr, eaddr);
1645         pr_getprot_done(&tmp);
1647         /*
1648          * Skip this segment unless the protection bits look like
1649          * what we'd expect for a text segment.
1650          */
1651         if ((prot & (PROT_WRITE | PROT_EXEC)) != PROT_EXEC)
1652             continue;
1654         if (getelfhead(mvp, credp, &ehdr, &nshdrs, &shstrndx,
1655                     &nphdrs) != 0 ||
1656             getelfshdr(mvp, credp, &ehdr, nshdrs, shstrndx,
1657                      &shbase, &shsize, &shstrbase, &shstrsize) != 0)
1658             continue;
1660         off = ehdr.e_shentsize;
1661         for (j = 1; j < nshdrs; j++, off += ehdr.e_shentsize) {
1662             Shdr *symtab = NULL, *strtab;
1664             shdr = (Shdr *) (shbase + off);
1666             if (shdr->sh_name >= shstrsize)
1667                 continue;
1669             name = shstrbase + shdr->sh_name;
1671             if (strcmp(name, shstrtab_data[STR_CTF]) == 0) {
1672                 if ((content & CC_CONTENT_CTF) == 0 ||
1673                     ctf_ndx != 0)
1674                     continue;
1676                 if (shdr->sh_link > 0 &&
1677                     shdr->sh_link < nshdrs) {
1678                     symtab = (Shdr *) (shbase +
1679                                         shdr->sh_link * ehdr.e_shentsize);
1680                 }
1682                 if (v != NULL && i < nv - 1) {
1683                     if (shdr->sh_size > datasz &&
1684                         shdr->sh_size <= elf_datasz_max) {
1685                         if (data != NULL)
1686                             kmem_free(data, datasz);
1688                         datasz = shdr->sh_size;
1689                         data = kmem_alloc(datasz,
1690                                         KM_SLEEP);
1691                     }
1693                     v[i].sh_name = shstrtab_ndx & shstrtab,

```

```

1694         STR_CTF);
1695         v[i].sh_addr = (Addr)(uintptr_t)saddr;
1696         v[i].sh_type = SHT_PROGBITS;
1697         v[i].sh_addralign = 4;
1698         *doffsetp = roundup(*doffsetp,
1699                             v[i].sh_addralign);
1700         v[i].sh_offset = *doffsetp;
1701         v[i].sh_size = shdr->sh_size;
1702         if (symtab == NULL) {
1703             v[i].sh_link = 0;
1704         } else if (symtab->sh_type ==
1705                 SHT_SYMTAB &&
1706                 symtab_ndx != 0) {
1707             v[i].sh_link =
1708                 symtab_ndx;
1709         } else {
1710             v[i].sh_link = i + 1;
1711         }
1712
1713         copy_scn(shdr,.mvp, &v[i], vp,
1714                doffsetp, data, datasz, credp,
1715                rlimit);
1716     }
1717
1718     ctf_ndx = i++;
1719
1720     /*
1721     * We've already dumped the symtab.
1722     */
1723     if (symtab != NULL &&
1724         symtab->sh_type == SHT_SYMTAB &&
1725         symtab_ndx != 0)
1726         continue;
1727
1728     } else if (strcmp(name,
1729                     shstrtab_data[STR_SYMTAB]) == 0) {
1730         if ((content & CC_CONTENT_SYMTAB) == 0 ||
1731             symtab != 0)
1732             continue;
1733
1734         symtab = shdr;
1735     }
1736
1737     if (symtab != NULL) {
1738         if ((symtab->sh_type != SHT_DYNSYM &&
1739             symtab->sh_type != SHT_SYMTAB) ||
1740             symtab->sh_link == 0 ||
1741             symtab->sh_link >= nshdrs)
1742             continue;
1743
1744         strtab = (Shdr *) (shbase +
1745                          symtab->sh_link * ehdr.e_shentsize);
1746
1747         if (strtab->sh_type != SHT_STRTAB)
1748             continue;
1749
1750         if (v != NULL && i < nv - 2) {
1751             sz = MAX(symtab->sh_size,
1752                    strtab->sh_size);
1753             if (sz > datasz &&
1754                 sz <= elf_datasz_max) {
1755                 if (data != NULL)
1756                     kmem_free(data, datasz);
1757
1758                 datasz = sz;
1759                 data = kmem_alloc(datasz,

```

```

1760             KM_SLEEP);
1761         }
1762
1763         if (symtab->sh_type == SHT_DYNSYM) {
1764             v[i].sh_name = shstrtab_ndx(
1765                 &shstrtab, STR_DYNSYM);
1766             v[i + 1].sh_name = shstrtab_ndx(
1767                 &shstrtab, STR_DYNSTR);
1768         } else {
1769             v[i].sh_name = shstrtab_ndx(
1770                 &shstrtab, STR_SYMTAB);
1771             v[i + 1].sh_name = shstrtab_ndx(
1772                 &shstrtab, STR_STRTAB);
1773         }
1774
1775         v[i].sh_type = symtab->sh_type;
1776         v[i].sh_addr = symtab->sh_addr;
1777         if (ehdr.e_type == ET_DYN ||
1778             v[i].sh_addr == 0)
1779             v[i].sh_addr +=
1780                 (Addr)(uintptr_t)saddr;
1781         v[i].sh_addralign =
1782             symtab->sh_addralign;
1783         *doffsetp = roundup(*doffsetp,
1784                             v[i].sh_addralign);
1785         v[i].sh_offset = *doffsetp;
1786         v[i].sh_size = symtab->sh_size;
1787         v[i].sh_link = i + 1;
1788         v[i].sh_entsize = symtab->sh_entsize;
1789         v[i].sh_info = symtab->sh_info;
1790
1791         copy_scn(symtab,.mvp, &v[i], vp,
1792                doffsetp, data, datasz, credp,
1793                rlimit);
1794
1795         v[i + 1].sh_type = SHT_STRTAB;
1796         v[i + 1].sh_flags = SHF_STRINGS;
1797         v[i + 1].sh_addr = symtab->sh_addr;
1798         if (ehdr.e_type == ET_DYN ||
1799             v[i + 1].sh_addr == 0)
1800             v[i + 1].sh_addr +=
1801                 (Addr)(uintptr_t)saddr;
1802         v[i + 1].sh_addralign =
1803             strtab->sh_addralign;
1804         *doffsetp = roundup(*doffsetp,
1805                             v[i + 1].sh_addralign);
1806         v[i + 1].sh_offset = *doffsetp;
1807         v[i + 1].sh_size = strtab->sh_size;
1808
1809         copy_scn(strtab,.mvp, &v[i + 1], vp,
1810                doffsetp, data, datasz, credp,
1811                rlimit);
1812     }
1813
1814     if (symtab->sh_type == SHT_SYMTAB)
1815         symtab_ndx = i;
1816     i += 2;
1817 }
1818
1819 }
1820
1821 kmem_free(shstrbase, shstrsize);
1822 kmem_free(shbase, shsize);
1823
1824 lastvp =.mvp;
1825 }

```

```

1826     if (v == NULL) {
1827         if (i == 1)
1828             *nshdrsp = 0;
1829         else
1830             *nshdrsp = i + 1;
1831         goto done;
1832     }
1833
1834     if (i != nv - 1) {
1835         cmn_err(CE_WARN, "elfcore: core dump failed for "
1836             "process %d; address space is changing", p->p_pid);
1837         error = EIO;
1838         goto done;
1839     }
1840
1841     v[i].sh_name = shstrtab_ndx(&shstrtab, STR_SHSTRTAB);
1842     v[i].sh_size = shstrtab_size(&shstrtab);
1843     v[i].sh_addralign = 1;
1844     *doffsetp = roundup(*doffsetp, v[i].sh_addralign);
1845     v[i].sh_offset = *doffsetp;
1846     v[i].sh_flags = SHF_STRINGS;
1847     v[i].sh_type = SHT_STRTAB;
1848
1849     if (v[i].sh_size > datasz) {
1850         if (data != NULL)
1851             kmem_free(data, datasz);
1852
1853         datasz = v[i].sh_size;
1854         data = kmem_alloc(datasz,
1855             KM_SLEEP);
1856     }
1857
1858     shstrtab_dump(&shstrtab, data);
1859
1860     if ((error = core_write(vp, UIO_SYSSPACE, *doffsetp,
1861         data, v[i].sh_size, rlimit, credp)) != 0)
1862         goto done;
1863
1864     *doffsetp += v[i].sh_size;
1865
1866 done:
1867     if (data != NULL)
1868         kmem_free(data, datasz);
1869
1870     return (error);
1871 }
1872
1873 int
1874 elfcore(vnode_t *vp, proc_t *p, cred_t *credp, rlim64_t rlimit, int sig,
1875     core_content_t content)
1876 {
1877     offset_t poffset, soffset;
1878     Off doffset;
1879     int error, i, nphdrs, nshdrs;
1880     int overflow = 0;
1881     struct seg *seg;
1882     struct as *as = p->p_as;
1883     union {
1884         Ehdr ehdr;
1885         Phdr phdr[1];
1886         Shdr shdr[1];
1887     } *bigwad;
1888     size_t bigsize;
1889     size_t phdrsz, shdrsz;
1890     Ehdr *ehdr;
1891     Phdr *p;

```

```

1892         caddr_t brkbase;
1893         size_t brksize;
1894         caddr_t stkbase;
1895         size_t stksize;
1896         int ntries = 0;
1897         klwp_t *lwp = ttolwp(curthread);
1898
1899     top:
1900         /*
1901          * Make sure we have everything we need (registers, etc.).
1902          * All other lwps have already stopped and are in an orderly state.
1903          */
1904         ASSERT(p == ttoproc(curthread));
1905         prstop(0, 0);
1906
1907         AS_LOCK_ENTER(as, RW_WRITER);
1908         nphdrs = prnsegs(as, 0) + 2;          /* two CORE note sections */
1909
1910         /*
1911          * Count the number of section headers we're going to need.
1912          */
1913         nshdrs = 0;
1914         if (content & (CC_CONTENT_CTF | CC_CONTENT_SYMTAB)) {
1915             (void) process_scns(content, p, credp, NULL, NULL, NULL, 0,
1916                 NULL, &nshdrs);
1917         }
1918         AS_LOCK_EXIT(as);
1919
1920         ASSERT(nshdrs == 0 || nshdrs > 1);
1921
1922         /*
1923          * The core file contents may require zero section headers, but if
1924          * we overflow the 16 bits allotted to the program header count in
1925          * the ELF header, we'll need that program header at index zero.
1926          */
1927         if (nshdrs == 0 && nphdrs >= PN_XNUM)
1928             nshdrs = 1;
1929
1930         phdrsz = nphdrs * sizeof (Phdr);
1931         shdrsz = nshdrs * sizeof (Shdr);
1932
1933         bigsize = MAX(sizeof (*bigwad), MAX(phdrsz, shdrsz));
1934         bigwad = kmem_alloc(bigsize, KM_SLEEP);
1935
1936         ehdr = &bigwad->ehdr;
1937         bzero(ehdr, sizeof (*ehdr));
1938
1939         ehdr->e_ident[EI_MAG0] = ELF_MAG0;
1940         ehdr->e_ident[EI_MAG1] = ELF_MAG1;
1941         ehdr->e_ident[EI_MAG2] = ELF_MAG2;
1942         ehdr->e_ident[EI_MAG3] = ELF_MAG3;
1943         ehdr->e_ident[EI_CLASS] = ELFCLASS;
1944         ehdr->e_type = ET_CORE;
1945
1946 #if !defined(_LP64) || defined(_ELF32_COMPAT)
1947
1948 #if defined(__sparc)
1949         ehdr->e_ident[EI_DATA] = ELFDATA2MSB;
1950         ehdr->e_machine = EM_SPARC;
1951 #elif defined(__i386) || defined(__i386_COMPAT)
1952         ehdr->e_ident[EI_DATA] = ELFDATA2LSB;
1953         ehdr->e_machine = EM_386;
1954 #else
1955 #error "no recognized machine type is defined"
1956 #endif

```

```

1958 #else /* !defined(_LP64) || defined(_ELF32_COMPAT) */
1960 #if defined(__sparc)
1961     ehdr->e_ident[EI_DATA] = ELFDATA2MSB;
1962     ehdr->e_machine = EM_SPARCV9;
1963 #elif defined(__amd64)
1964     ehdr->e_ident[EI_DATA] = ELFDATA2LSB;
1965     ehdr->e_machine = EM_AMD64;
1966 #else
1967 #error "no recognized 64-bit machine type is defined"
1968 #endif
1970 #endif /* !defined(_LP64) || defined(_ELF32_COMPAT) */
1972 /*
1973  * If the count of program headers or section headers or the index
1974  * of the section string table can't fit in the mere 16 bits
1975  * shortsightedly allotted to them in the ELF header, we use the
1976  * extended formats and put the real values in the section header
1977  * as index 0.
1978  */
1979 ehdr->e_version = EV_CURRENT;
1980 ehdr->e_ehsize = sizeof (Ehdr);
1982 if (nphdrs >= PN_XNUM)
1983     ehdr->e_phnum = PN_XNUM;
1984 else
1985     ehdr->e_phnum = (unsigned short)nphdrs;
1987 ehdr->e_phoff = sizeof (Ehdr);
1988 ehdr->e_phentsize = sizeof (Phdr);
1990 if (nshdrs > 0) {
1991     if (nshdrs >= SHN_LORESERVE)
1992         ehdr->e_shnum = 0;
1993     else
1994         ehdr->e_shnum = (unsigned short)nshdrs;
1996     if (nshdrs - 1 >= SHN_LORESERVE)
1997         ehdr->e_shstrndx = SHN_XINDEX;
1998     else
1999         ehdr->e_shstrndx = (unsigned short)(nshdrs - 1);
2001     ehdr->e_shoff = ehdr->e_phoff + ehdr->e_phentsize * nphdrs;
2002     ehdr->e_shentsize = sizeof (Shdr);
2003 }
2005 if (error = core_write(vp, UIO_SYSSPACE, (offset_t)0, ehdr,
2006     sizeof (Ehdr), rlimit, credp))
2007     goto done;
2009 poffset = sizeof (Ehdr);
2010 soffset = sizeof (Ehdr) + phdrsz;
2011 doffset = sizeof (Ehdr) + phdrsz + shdrsz;
2013 v = &bigwad->phdr[0];
2014 bzero(v, phdrsz);
2016 setup_old_note_header(&v[0], p);
2017 v[0].p_offset = doffset = roundup(doffset, sizeof (Word));
2018 doffset += v[0].p_filesz;
2020 setup_note_header(&v[1], p);
2021 v[1].p_offset = doffset = roundup(doffset, sizeof (Word));
2022 doffset += v[1].p_filesz;

```

```

2024     mutex_enter(&p->p_lock);
2026     brkbase = p->p_brkbase;
2027     brksize = p->p_brksize;
2029     stkbase = p->p_usrstack - p->p_stksize;
2030     stksize = p->p_stksize;
2032     mutex_exit(&p->p_lock);
2034     AS_LOCK_ENTER(as, RW_WRITER);
2035     i = 2;
2036     for (seg = AS_SEGFIRST(as); seg != NULL; seg = AS_SEGNEXT(as, seg)) {
2037         caddr_t eaddr = seg->s_base + pr_getsegsz(seg, 0);
2038         caddr_t saddr, naddr;
2039         void *tmp = NULL;
2040         extern struct seg_ops segspt_shmops;
2042         for (saddr = seg->s_base; saddr < eaddr; saddr = naddr) {
2043             uint_t prot;
2044             size_t size;
2045             int type;
2046             vnode_t *mvp;
2048             prot = pr_getprot(seg, 0, &tmp, &saddr, &naddr, eaddr);
2049             prot &= PROT_READ | PROT_WRITE | PROT_EXEC;
2050             if ((size = (size_t)(naddr - saddr)) == 0)
2051                 continue;
2052             if (i == nphdrs) {
2053                 overflow++;
2054                 continue;
2055             }
2056             v[i].p_type = PT_LOAD;
2057             v[i].p_vaddr = (Addr)(uintptr_t)saddr;
2058             v[i].p_memsz = size;
2059             if (prot & PROT_READ)
2060                 v[i].p_flags |= PF_R;
2061             if (prot & PROT_WRITE)
2062                 v[i].p_flags |= PF_W;
2063             if (prot & PROT_EXEC)
2064                 v[i].p_flags |= PF_X;
2066             /*
2067              * Figure out which mappings to include in the core.
2068              */
2069             type = SEGOP_GETTYPE(seg, saddr);
2071             if (saddr == stkbase && size == stksize) {
2072                 if (!(content & CC_CONTENT_STACK))
2073                     goto exclude;
2075             } else if (saddr == brkbase && size == brksize) {
2076                 if (!(content & CC_CONTENT_HEAP))
2077                     goto exclude;
2079             } else if (seg->s_ops == &segspt_shmops) {
2080                 if (type & MAP_NORESERVE) {
2081                     if (!(content & CC_CONTENT_DISM))
2082                         goto exclude;
2083                 } else {
2084                     if (!(content & CC_CONTENT_ISM))
2085                         goto exclude;
2086                 }
2088             } else if (seg->s_ops != &segvn_ops) {
2089                 goto exclude;

```



```

2091         } else if (type & MAP_SHARED) {
2092             if (shmgetid(p, saddr) != SHMID_NONE) {
2093                 if (!(content & CC_CONTENT_SHM))
2094                     goto exclude;
2096             } else if (SEGOP_GETVP(seg, seg->s_base,
2097 &mvp) != 0 || mvp == NULL ||
2098 mvp->v_type != VREG) {
2099                 if (!(content & CC_CONTENT_SHANON))
2100                     goto exclude;
2102             } else {
2103                 if (!(content & CC_CONTENT_SHFILE))
2104                     goto exclude;
2105             }
2107         } else if (SEGOP_GETVP(seg, seg->s_base, &mvp) != 0 ||
2108 mvp == NULL || mvp->v_type != VREG) {
2109             if (!(content & CC_CONTENT_ANON))
2110                 goto exclude;
2112         } else if (prot == (PROT_READ | PROT_EXEC)) {
2113             if (!(content & CC_CONTENT_TEXT))
2114                 goto exclude;
2116         } else if (prot == PROT_READ) {
2117             if (!(content & CC_CONTENT_RODATA))
2118                 goto exclude;
2120         } else {
2121             if (!(content & CC_CONTENT_DATA))
2122                 goto exclude;
2123         }
2125         doffset = roundup(doffset, sizeof (Word));
2126         v[i].p_offset = doffset;
2127         v[i].p_filesz = size;
2128         doffset += size;
2129     exclude:
2130         i++;
2131     }
2132     ASSERT(tmp == NULL);
2133 }
2134 AS_LOCK_EXIT(as);
2136 if (overflow || i != nphdrs) {
2137     if (ntries++ == 0) {
2138         kmem_free(bigwad, bigsize);
2139         overflow = 0;
2140         goto top;
2141     }
2142     cmn_err(CE_WARN, "elfcore: core dump failed for "
2143 "process %d; address space is changing", p->p_pid);
2144     error = EIO;
2145     goto done;
2146 }
2148 if ((error = core_write(vp, UIO_SYSSPACE, poffset,
2149 v, phdrsz, rlimit, credp)) != 0)
2150     goto done;
2152 if ((error = write_old_elfnotes(p, sig, vp, v[0].p_offset, rlimit,
2153 credp)) != 0)
2154     goto done;

```

```

2156         if ((error = write_elfnotes(p, sig, vp, v[i].p_offset, rlimit,
2157 credp, content)) != 0)
2158             goto done;
2160         for (i = 2; i < nphdrs; i++) {
2161             prkillinfo_t killinfo;
2162             sigqueue_t *sq;
2163             int sig, j;
2165             if (v[i].p_filesz == 0)
2166                 continue;
2168             /*
2169              * If dumping out this segment fails, rather than failing
2170              * the core dump entirely, we reset the size of the mapping
2171              * to zero to indicate that the data is absent from the core
2172              * file and or in the PF_SUNW_FAILURE flag to differentiate
2173              * this from mappings that were excluded due to the core file
2174              * content settings.
2175              */
2176             if ((error = core_seg(p, vp, v[i].p_offset,
2177 (caddr_t)(uintptr_t)v[i].p_vaddr, v[i].p_filesz,
2178 rlimit, credp)) == 0) {
2179                 continue;
2180             }
2182             if ((sig = lwp->lwp_cursig) == 0) {
2183                 /*
2184                  * We failed due to something other than a signal.
2185                  * Since the space reserved for the segment is now
2186                  * unused, we stash the errno in the first four
2187                  * bytes. This undocumented interface will let us
2188                  * understand the nature of the failure.
2189                  */
2190                 (void) core_write(vp, UIO_SYSSPACE, v[i].p_offset,
2191 &error, sizeof (error), rlimit, credp);
2193                 v[i].p_filesz = 0;
2194                 v[i].p_flags |= PF_SUNW_FAILURE;
2195                 if ((error = core_write(vp, UIO_SYSSPACE,
2196 poffset + sizeof (v[i]) * i, &v[i], sizeof (v[i]),
2197 rlimit, credp)) != 0)
2198                     goto done;
2200             }
2201             continue;
2203             /*
2204              * We took a signal. We want to abort the dump entirely, but
2205              * we also want to indicate what failed and why. We therefore
2206              * use the space reserved for the first failing segment to
2207              * write our error (which, for purposes of compatability with
2208              * older core dump readers, we set to EINTR) followed by any
2209              * siginfo associated with the signal.
2210              */
2211             bzero(&killinfo, sizeof (killinfo));
2212             killinfo.prk_error = EINTR;
2214             sq = sig == SIGKILL ? curproc->p_killsq : lwp->lwp_curinfo;
2216             if (sq != NULL) {
2217                 bcopy(&sq->sq_info, &killinfo.prk_info,
2218 sizeof (sq->sq_info));
2219             } else {
2220                 killinfo.prk_info.si_signo = lwp->lwp_cursig;
2221                 killinfo.prk_info.si_code = SI_NOINFO;

```

```

2222     }
2224 #if (defined(_SYSCALL32_IMPL) || defined(_LP64))
2225     /*
2226     * If this is a 32-bit process, we need to translate from the
2227     * native siginfo to the 32-bit variant. (Core readers must
2228     * always have the same data model as their target or must
2229     * be aware of -- and compensate for -- data model differences.)
2230     */
2231     if (curproc->p_model == DATAMODEL_ILP32) {
2232         siginfo32_t si32;
2233
2234         siginfo_kto32((k_siginfo_t *)&killinfo.prk_info, &si32);
2235         bcopy(&si32, &killinfo.prk_info, sizeof(si32));
2236     }
2237 #endif
2239     (void) core_write(vp, UIO_SYSSPACE, v[i].p_offset,
2240                    &killinfo, sizeof(killinfo), rlimit, credp);
2242     /*
2243     * For the segment on which we took the signal, indicate that
2244     * its data now refers to a siginfo.
2245     */
2246     v[i].p_filesz = 0;
2247     v[i].p_flags |= PF_SUNW_FAILURE | PF_SUNW_KILLED |
2248                 PF_SUNW_SIGINFO;
2250     /*
2251     * And for every other segment, indicate that its absence
2252     * is due to a signal.
2253     */
2254     for (j = i + 1; j < nphdrs; j++) {
2255         v[j].p_filesz = 0;
2256         v[j].p_flags |= PF_SUNW_FAILURE | PF_SUNW_KILLED;
2257     }
2259     /*
2260     * Finally, write out our modified program headers.
2261     */
2262     if ((error = core_write(vp, UIO_SYSSPACE,
2263                    poffset + sizeof(v[i]) * i, &v[i],
2264                    sizeof(v[i]) * (nphdrs - i), rlimit, credp)) != 0)
2265         goto done;
2267     break;
2268 }
2270 if (nshdrs > 0) {
2271     bzero(&bigwad->shdr[0], shdrsz);
2273     if (nshdrs >= SHN_LORESERVE)
2274         bigwad->shdr[0].sh_size = nshdrs;
2276     if (nshdrs - 1 >= SHN_LORESERVE)
2277         bigwad->shdr[0].sh_link = nshdrs - 1;
2279     if (nphdrs >= PN_XNUM)
2280         bigwad->shdr[0].sh_info = nphdrs;
2282     if (nshdrs > 1) {
2283         AS_LOCK_ENTER(as, RW_WRITER);
2284         if ((error = process_scns(content, p, credp, vp,
2285                 &bigwad->shdr[0], nshdrs, rlimit, &doffset,
2286                 NULL)) != 0) {
2287             AS_LOCK_EXIT(as);

```

```

2288         goto done;
2289     }
2290     AS_LOCK_EXIT(as);
2291 }
2293     if ((error = core_write(vp, UIO_SYSSPACE, soffset,
2294                    &bigwad->shdr[0], shdrsz, rlimit, credp)) != 0)
2295         goto done;
2296 }
2298 done:
2299     kmem_free(bigwad, bigsize);
2300     return (error);
2301 }
2303 #ifndef _ELF32_COMPAT
2305 static struct execsw esw = {
2306 #ifdef _LP64
2307     elf64magicstr,
2308 #else /* _LP64 */
2309     elf32magicstr,
2310 #endif /* _LP64 */
2311     0,
2312     5,
2313     elfexec,
2314     elfcore
2315 };
2317 static struct modlexec modlexec = {
2318     &mod_execops, "exec module for elf", &esw
2319 };
2321 #ifdef _LP64
2322 extern int elf32exec(vnode_t *vp, execa_t *uap, uarg_t *args,
2323                    intpdata_t *idatap, int level, long *execsz,
2324                    int setid, caddr_t exec_file, cred_t *cred,
2325                    int brand_action);
2326 extern int elf32core(vnode_t *vp, proc_t *p, cred_t *credp,
2327                    rlim64_t rlimit, int sig, core_content_t content);
2329 static struct execsw esw32 = {
2330     elf32magicstr,
2331     0,
2332     5,
2333     elf32exec,
2334     elf32core
2335 };
2337 static struct modlexec modlexec32 = {
2338     &mod_execops, "32-bit exec module for elf", &esw32
2339 };
2340 #endif /* _LP64 */
2342 static struct modlinkage modlinkage = {
2343     MODREV_1,
2344     (void *)&modlexec,
2345 #ifdef _LP64
2346     (void *)&modlexec32,
2347 #endif /* _LP64 */
2348     NULL
2349 };
2351 int
2352 _init(void)
2353 {

```

```
2354     return (mod_install(&modlinkage));
2355 }

2357 int
2358 _fini(void)
2359 {
2360     return (mod_remove(&modlinkage));
2361 }

2363 int
2364 _info(struct modinfo *modinfop)
2365 {
2366     return (mod_info(&modlinkage, modinfop));
2367 }

2369 #endif /* !_ELF32_COMPAT */
```

```

*****
15707 Thu Jun 30 22:03:19 2016
new/usr/src/uts/common/exec/elf/elf_notes.c
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
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, Version 1.0 only
6  * (the "License"). You may not use this file except in compliance
7  * with the License.
8  *
9  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
10 * or http://www.opensolaris.org/os/licensing.
11 * See the License for the specific language governing permissions
12 * and limitations under the License.
13 *
14 * When distributing Covered Code, include this CDDL HEADER in each
15 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
16 * If applicable, add the following below this CDDL HEADER, with the
17 * fields enclosed by brackets "[]" replaced with your own identifying
18 * information: Portions Copyright [yyyy] [name of copyright owner]
19 *
20 * CDDL HEADER END
21 */
22 /*
23  * Copyright 2004 Sun Microsystems, Inc. All rights reserved.
24  * Use is subject to license terms.
25 */

27 /*
28  * Copyright 2012 DEY Storage Systems, Inc. All rights reserved.
29  * Copyright (c) 2014, Joyent, Inc. All rights reserved.
30 */

32 #include <sys/types.h>
33 #include <sys/param.h>
34 #include <sys/thread.h>
35 #include <sys/symmacros.h>
36 #include <sys/signal.h>
37 #include <sys/cred.h>
38 #include <sys/priv.h>
39 #include <sys/user.h>
40 #include <sys/file.h>
41 #include <sys/errno.h>
42 #include <sys/vnode.h>
43 #include <sys/mode.h>
44 #include <sys/vfs.h>
45 #include <sys/mman.h>
46 #include <sys/kmem.h>
47 #include <sys/proc.h>
48 #include <sys/pathname.h>
49 #include <sys/cmn_err.h>
50 #include <sys/system.h>
51 #include <sys/elf.h>
52 #include <sys/vmsystem.h>
53 #include <sys/debug.h>
54 #include <sys/procfs.h>
55 #include <sys/regset.h>
56 #include <sys/auxv.h>
57 #include <sys/exec.h>
58 #include <sys/prsystem.h>

```

```

59 #include <sys/utsname.h>
60 #include <sys/zone.h>
61 #include <vm/as.h>
62 #include <vm/xm.h>
63 #include <sys/modctl.h>
64 #include <sys/systeminfo.h>
65 #include <sys/machelf.h>
66 #include <sys/sunddi.h>
67 #include "elf_impl.h"
68 #if defined(__i386) || defined(__i386_COMPAT)
69 #include <sys/sysi86.h>
70 #endif

72 void
73 setup_note_header(Phdr *v, proc_t *p)
74 {
75     int nlwp = p->p_lwpcnt;
76     int nzomb = p->p_zombcnt;
77     int nfd;
78     size_t size;
79     pcred_t *pcrp;
80     uf_info_t *fip;
81     uf_entry_t *ufp;
82     int fd;

84     fip = P_FINFO(p);
85     nfd = 0;
86     mutex_enter(&fip->fi_lock);
87     for (fd = 0; fd < fip->fi_nfiles; fd++) {
88         UF_ENTER(ufp, fip, fd);
89         if ((ufp->uf_file != NULL) && (ufp->uf_file->f_count > 0))
90             nfd++;
91         UF_EXIT(ufp);
92     }
93     mutex_exit(&fip->fi_lock);

95     v[0].p_type = PT_NOTE;
96     v[0].p_flags = PF_R;
97     v[0].p_filesz = (sizeof (Note) * (10 + 2 * nlwp + nzomb + nfd))
98     v[0].p_filesz = (sizeof (Note) * (9 + 2 * nlwp + nzomb + nfd))
99     + roundup(sizeof (psinfo_t), sizeof (Word))
100     + roundup(sizeof (pstatus_t), sizeof (Word))
101     + roundup(prgetprivsize(), sizeof (Word))
102     + roundup(priv_get_implinfo_size(), sizeof (Word))
103     + roundup(strlen(platform) + 1, sizeof (Word))
104     + roundup(strlen(p->p_zone->zone_name) + 1, sizeof (Word))
105     + roundup(__KERN_NAUXV_IMPL * sizeof (aux_entry_t), sizeof (Word))
106     + roundup(sizeof (utsname), sizeof (Word))
107     + roundup(sizeof (core_content_t), sizeof (Word))
108     + roundup(sizeof (prsecflags_t), sizeof (Word))
109 #endif /* ! codereview */
110     + (nlwp + nzomb) * roundup(sizeof (lwpstatus_t), sizeof (Word))
111     + nlwp * roundup(sizeof (lwpstatus_t), sizeof (Word))
112     + nfd * roundup(sizeof (prfdinfo_t), sizeof (Word));

113     if (curproc->p_agenttp != NULL) {
114         v[0].p_filesz += sizeof (Note) +
115             roundup(sizeof (psinfo_t), sizeof (Word));
116     }

118     size = sizeof (pcred_t) + sizeof (gid_t) * (ngroups_max - 1);
119     pcrp = kmem_alloc(size, KM_SLEEP);
120     prgetcred(p, pcrp);
121     if (pcrp->pr_ngroups != 0) {
122         v[0].p_filesz += sizeof (Note) + roundup(sizeof (pcred_t) +
123             sizeof (gid_t) * (pcrp->pr_ngroups - 1), sizeof (Word));

```

```

124     } else {
125         v[0].p_filesz += sizeof (Note) +
126             roundup(sizeof (prcred_t), sizeof (Word));
127     }
128     kmem_free(pcrp, size);

131 #if defined(__i386) || defined(__i386_COMPAT)
132     mutex_enter(&p->p_ldtlock);
133     size = prnldt(p) * sizeof (struct ssd);
134     mutex_exit(&p->p_ldtlock);
135     if (size != 0)
136         v[0].p_filesz += sizeof (Note) + roundup(size, sizeof (Word));
137 #endif /* __i386 || __i386_COMPAT */

139     if ((size = prhasx(p)? prgetprxregsize(p) : 0) != 0)
140         v[0].p_filesz += nlwp * sizeof (Note)
141             + nlwp * roundup(size, sizeof (Word));

143 #if defined(__sparc)
144     /*
145      * Figure out the number and sizes of register windows.
146      */
147     {
148         kthread_t *t = p->p_tlist;
149         do {
150             if (((size = prnwindows(ttolwp(t))) != 0) {
151                 size = sizeof (gwindows_t) -
152                     (SPARC_MAXREGWINDOW - size) *
153                     sizeof (struct rwindow);
154                 v[0].p_filesz += sizeof (Note) +
155                     roundup(size, sizeof (Word));
156             }
157         } while ((t = t->t_forw) != p->p_tlist);
158     }
159     /*
160      * Space for the Ancillary State Registers.
161      */
162     if (p->p_model == DATAMODEL_LP64)
163         v[0].p_filesz += nlwp * sizeof (Note)
164             + nlwp * roundup(sizeof (asrset_t), sizeof (Word));
165 #endif /* __sparc */
166 }

168 int
169 write_elfnotes(proc_t *p, int sig, vnode_t *vp, offset_t offset,
170             rlim64_t rlimit, cred_t *credp, core_content_t content)
171 {
172     union {
173         psinfo_t      psinfo;
174         pstatus_t     pstatus;
175         lwpsinfo_t    lwpsinfo;
176         lwpstatus_t   lwpstatus;
177 #if defined(__sparc)
178         gwindows_t    gwindows;
179         asrset_t       asrset;
180 #endif /* __sparc */
181         char           xregs[1];
182         aux_entry_t    auxv[__KERN_NAUXV_IMPL];
183         prcred_t       pcred;
184         prpriv_t       ppriv;
185         priv_impl_info_t prinfo;
186         struct utsname uts;
187         prsecflags_t   psecflags;
188 #endif /* ! codereview */
189     } *bigwad;

```

```

191     size_t xregsize = prhasx(p)? prgetprxregsize(p) : 0;
192     size_t crsize = sizeof (prcred_t) + sizeof (gid_t) * (ngroups_max - 1);
193     size_t psize = prgetprivsize();
194     size_t bigsize = MAX(psize, MAX(sizeof (*bigwad),
195         MAX(xregsize, crsize)));

197     priv_impl_info_t *prii;

199     lwpdir_t *ldp;
200     lwpent_t *lep;
201     kthread_t *t;
202     klwp_t *lwp;
203     user_t *up;
204     int i;
205     int nlwp;
206     int nzomb;
207     int error;
208     uchar_t oldsig;
209     uf_info_t *fip;
210     int fd;
211     vnode_t *vroot;

213 #if defined(__i386) || defined(__i386_COMPAT)
214     struct ssd *ssd;
215     size_t ssdsize;
216 #endif /* __i386 || __i386_COMPAT */

218     bigsize = MAX(bigsize, priv_get_implinfo_size());

220     bigwad = kmem_alloc(bigsize, KM_SLEEP);

222     /*
223      * The order of the elfnote entries should be same here
224      * and in the gcore(1) command. Synchronization is
225      * needed between the kernel and gcore(1).
226      */

228     /*
229      * Get the psinfo, and set the wait status to indicate that a core was
230      * dumped. We have to forge this since p->p_wcode is not set yet.
231      */
232     mutex_enter(&p->p_lock);
233     prgetpsinfo(p, &bigwad->psinfo);
234     mutex_exit(&p->p_lock);
235     bigwad->psinfo.pr_wstat = wstat(CLD_DUMPED, sig);

237     error = elfnote(vp, &offset, NT_PSINFO, sizeof (bigwad->psinfo),
238         (caddr_t)&bigwad->psinfo, rlimit, credp);
239     if (error)
240         goto done;

242     /*
243      * Modify t_whystop and lwp_cursig so it appears that the current LWP
244      * is stopped after faulting on the signal that caused the core dump.
245      * As a result, prgetstatus() will record that signal, the saved
246      * lwp_siginfo, and its signal handler in the core file status. We
247      * restore lwp_cursig in case a subsequent signal was received while
248      * dumping core.
249      */
250     mutex_enter(&p->p_lock);
251     lwp = ttolwp(curthread);

253     oldsig = lwp->lwp_cursig;
254     lwp->lwp_cursig = (uchar_t)sig;
255     curthread->t_whystop = PR_FAULTED;

```

```

257 prgetstatus(p, &bigwad->pstatus, p->p_zone);
258 bigwad->pstatus.pr_lwp.pr_why = 0;

260 curthread->t_whystop = 0;
261 lwp->lwp_cursig = oldsigs;
262 mutex_exit(&p->p_lock);

264 error = elfnote(vp, &offset, NT_PSTATUS, sizeof (bigwad->pstatus),
265 (caddr_t)&bigwad->pstatus, rlimit, credp);
266 if (error)
267     goto done;

269 error = elfnote(vp, &offset, NT_PLATFORM, strlen(platform) + 1,
270 platform, rlimit, credp);
271 if (error)
272     goto done;

274 up = PTOU(p);
275 for (i = 0; i < __KERN_NAUXV_IMPL; i++) {
276     bigwad->auxv[i].a_type = up->u_auxv[i].a_type;
277     bigwad->auxv[i].a_un.a_val = up->u_auxv[i].a_un.a_val;
278 }
279 error = elfnote(vp, &offset, NT_AUXV, sizeof (bigwad->auxv),
280 (caddr_t)bigwad->auxv, rlimit, credp);
281 if (error)
282     goto done;

284 bcopy(&utsname, &bigwad->uts, sizeof (struct utsname));
285 if (!INGLOBALZONE(p)) {
286     bcopy(p->p_zone->zone_nodename, &bigwad->uts.nodename,
287         _SYS_NMLN);
288 }
289 error = elfnote(vp, &offset, NT_UTSNAME, sizeof (struct utsname),
290 (caddr_t)&bigwad->uts, rlimit, credp);
291 if (error)
292     goto done;

294 prgetsecflags(p, &bigwad->psecflags);
295 error = elfnote(vp, &offset, NT_SECFLAGS, sizeof (prsecflags_t),
296 (caddr_t)&bigwad->psecflags, rlimit, credp);
297 if (error)
298     goto done;

300 #endif /* ! codereview */
301 prgetcred(p, &bigwad->pcred);

303 if (bigwad->pcred.pr_ngroups != 0) {
304     crsize = sizeof (prcred_t) +
305         sizeof (gid_t) * (bigwad->pcred.pr_ngroups - 1);
306 } else
307     crsize = sizeof (prcred_t);

309 error = elfnote(vp, &offset, NT_PRCRED, crsize,
310 (caddr_t)&bigwad->pcred, rlimit, credp);
311 if (error)
312     goto done;

314 error = elfnote(vp, &offset, NT_CONTENT, sizeof (core_content_t),
315 (caddr_t)&content, rlimit, credp);
316 if (error)
317     goto done;

319 prgetpriv(p, &bigwad->ppriv);

321 error = elfnote(vp, &offset, NT_PRRIV, psize,

```

```

322     (caddr_t)&bigwad->ppriv, rlimit, credp);
323 if (error)
324     goto done;

326 prii = priv_hold_implinfo();
327 error = elfnote(vp, &offset, NT_PRRIVINFO, priv_get_implinfo_size(),
328 (caddr_t)prii, rlimit, credp);
329 priv_release_implinfo();
330 if (error)
331     goto done;

333 /* zone can't go away as long as process exists */
334 error = elfnote(vp, &offset, NT_ZONENAME,
335     strlen(p->p_zone->zone_name) + 1, p->p_zone->zone_name,
336     rlimit, credp);
337 if (error)
338     goto done;

341 /* open file table */
342 vroot = PTOU(p)->u_rdir;
343 if (vroot == NULL)
344     vroot = rootdir;

346 VN_HOLD(vroot);

348 fip = P_FINFO(p);

350 for (fd = 0; fd < fip->fi_nfiles; fd++) {
351     uf_entry_t *ufp;
352     vnode_t *fvp;
353     struct file *fp;
354     vattr_t vattr;
355     prfdinfo_t fdinfo;

357     bzero(&fdinfo, sizeof (fdinfo));

359     mutex_enter(&fip->fi_lock);
360     UF_ENTER(ufp, fip, fd);
361     if (((fp = ufp->uf_file) == NULL) || (fp->f_count < 1)) {
362         UF_EXIT(ufp);
363         mutex_exit(&fip->fi_lock);
364         continue;
365     }

367     fdinfo.pr_fd = fd;
368     fdinfo.pr_fdflags = ufp->uf_flag;
369     fdinfo.pr_fileflags = fp->f_flag2;
370     fdinfo.pr_fileflags <= 16;
371     fdinfo.pr_fileflags |= fp->f_flag;
372     if ((fdinfo.pr_fileflags & (FSEARCH | FEEXEC)) == 0)
373         fdinfo.pr_fileflags += FOPEN;
374     fdinfo.pr_offset = fp->f_offset;

377     fvp = fp->f_vnode;
378     VN_HOLD(fvp);
379     UF_EXIT(ufp);
380     mutex_exit(&fip->fi_lock);

382     /*
383     * There are some vnodes that have no corresponding
384     * path. Its reasonable for this to fail, in which
385     * case the path will remain an empty string.
386     */
387     (void) vnodetopath(vroot, fvp, fdinfo.pr_path,

```

```

388         sizeof (fdinfo.pr_path), credp);
390     if (VOP_GETATTR(fvp, &vattr, 0, credp, NULL) != 0) {
391         /*
392          * Try to write at least a subset of information
393          */
394         fdinfo.pr_major = 0;
395         fdinfo.pr_minor = 0;
396         fdinfo.pr_ino = 0;
397         fdinfo.pr_mode = 0;
398         fdinfo.pr_uid = (uid_t)-1;
399         fdinfo.pr_gid = (gid_t)-1;
400         fdinfo.pr_rmajor = 0;
401         fdinfo.pr_rminor = 0;
402         fdinfo.pr_size = -1;
404
405         error = elfnote(vp, &offset, NT_FDINFO,
406             sizeof (fdinfo), &fdinfo, rlimit, credp);
407         VN_RELE(fvp);
408         VN_RELE(vroot);
409         if (error)
410             goto done;
411         continue;
413     if (fvp->v_type == VSOCK)
414         fdinfo.pr_fileflags |= sock_getfasync(fvp);
416     VN_RELE(fvp);
418     /*
419     * This logic mirrors fstat(), which we cannot use
420     * directly, as it calls copyout().
421     */
422     fdinfo.pr_major = getmajor(vattr.va_fsid);
423     fdinfo.pr_minor = getminor(vattr.va_fsid);
424     fdinfo.pr_ino = (ino64_t)vattr.va_nodeid;
425     fdinfo.pr_mode = VTTOIF(vattr.va_type) | vattr.va_mode;
426     fdinfo.pr_uid = vattr.va_uid;
427     fdinfo.pr_gid = vattr.va_gid;
428     fdinfo.pr_rmajor = getmajor(vattr.va_rdev);
429     fdinfo.pr_rminor = getminor(vattr.va_rdev);
430     fdinfo.pr_size = (off64_t)vattr.va_size;
432     error = elfnote(vp, &offset, NT_FDINFO,
433         sizeof (fdinfo), &fdinfo, rlimit, credp);
434     if (error) {
435         VN_RELE(vroot);
436         goto done;
437     }
438 }
440     VN_RELE(vroot);
442 #if defined(__i386) || defined(__i386_COMPAT)
443     mutex_enter(&p->p_ldtlock);
444     ssdsize = prnldt(p) * sizeof (struct ssd);
445     if (ssdsize != 0) {
446         ssd = kmem_alloc(ssdsize, KM_SLEEP);
447         prgetldt(p, ssd);
448         error = elfnote(vp, &offset, NT_LDT, ssdsize,
449             (caddr_t)ssd, rlimit, credp);
450         kmem_free(ssd, ssdsize);
451     }
452     mutex_exit(&p->p_ldtlock);
453     if (error)

```

```

454         goto done;
455 #endif /* __i386 || defined(__i386_COMPAT) */
457     nlwp = p->p_lwpcnt;
458     nzomb = p->p_zombcnt;
459     /* for each entry in the lwp directory ... */
460     for (ldp = p->p_lwpdir; nlwp + nzomb != 0; ldp++) {
462         if ((lep = ldp->ld_entry) == NULL) /* empty slot */
463             continue;
465         if ((t = lep->le_thread) != NULL) { /* active lwp */
466             ASSERT(nlwp != 0);
467             nlwp--;
468             lwp = ttolwp(t);
469             mutex_enter(&p->p_lock);
470             prgetlwpsinfo(t, &bigwad->lwpsinfo);
471             mutex_exit(&p->p_lock);
472         } else { /* zombie lwp */
473             ASSERT(nzomb != 0);
474             nzomb--;
475             bzero(&bigwad->lwpsinfo, sizeof (bigwad->lwpsinfo));
476             bigwad->lwpsinfo.pr_lwpid = lep->le_lwpid;
477             bigwad->lwpsinfo.pr_state = SZOMB;
478             bigwad->lwpsinfo.pr_sname = 'Z';
479             bigwad->lwpsinfo.pr_start.tv_sec = lep->le_start;
480         }
481         error = elfnote(vp, &offset, NT_LWPSINFO,
482             sizeof (bigwad->lwpsinfo), (caddr_t)&bigwad->lwpsinfo,
483             rlimit, credp);
484         if (error)
485             goto done;
486         if (t == NULL) /* nothing more to do for a zombie */
487             continue;
489         mutex_enter(&p->p_lock);
490         if (t == curthread) {
491             /*
492              * Modify t_whystop and lwp_cursig so it appears that
493              * the current LWP is stopped after faulting on the
494              * signal that caused the core dump. As a result,
495              * prgetlwpstatus() will record that signal, the saved
496              * lwp_siginfo, and its signal handler in the core file
497              * status. We restore lwp_cursig in case a subsequent
498              * signal was received while dumping core.
499              */
500             oldsig = lwp->lwp_cursig;
501             lwp->lwp_cursig = (uchar_t)sig;
502             t->t_whystop = PR_FAULTED;
504             prgetlwpstatus(t, &bigwad->lwpstatus, p->p_zone);
505             bigwad->lwpstatus.pr_why = 0;
507             t->t_whystop = 0;
508             lwp->lwp_cursig = oldsig;
509         } else {
510             prgetlwpstatus(t, &bigwad->lwpstatus, p->p_zone);
511         }
512         mutex_exit(&p->p_lock);
513         error = elfnote(vp, &offset, NT_LWPSTATUS,
514             sizeof (bigwad->lwpstatus), (caddr_t)&bigwad->lwpstatus,
515             rlimit, credp);
516         if (error)
517             goto done;
519 #if defined(__sparc)

```

```

520     /*
521     * Unspilled SPARC register windows.
522     */
523     {
524         size_t size = prnwindows(lwp);
525
526         if (size != 0) {
527             size = sizeof (gwindows_t) -
528                 (SPARC_MAXREGWINDOW - size) *
529                 sizeof (struct rwindow);
530             prgetwindows(lwp, &bigwad->gwindows);
531             error = elfnote(vp, &offset, NT_GWINDOWS,
532                 size, (caddr_t)&bigwad->gwindows,
533                 rlimit, credp);
534             if (error)
535                 goto done;
536         }
537     }
538     /*
539     * Ancillary State Registers.
540     */
541     if (p->p_model == DATAMODEL_LP64) {
542         prgetasregs(lwp, bigwad->asrset);
543         error = elfnote(vp, &offset, NT_ASRS,
544             sizeof (asrset_t), (caddr_t)bigwad->asrset,
545             rlimit, credp);
546         if (error)
547             goto done;
548     }
549 #endif /* __sparc */
550
551     if (xregsize) {
552         prgetprxregs(lwp, bigwad->xregs);
553         error = elfnote(vp, &offset, NT_PRXREG,
554             xregsize, bigwad->xregs, rlimit, credp);
555         if (error)
556             goto done;
557     }
558
559     if (t->t_lwp->lwp_spymaster != NULL) {
560         void *psaddr = t->t_lwp->lwp_spymaster;
561 #ifdef _ELF32_COMPAT
562         /*
563          * On a 64-bit kernel with 32-bit ELF compatibility,
564          * this file is compiled into two different objects:
565          * one is compiled normally, and the other is compiled
566          * with _ELF32_COMPAT set -- and therefore with a
567          * psinfo_t defined to be a psinfo32_t. However, the
568          * psinfo_t denoting our spymaster is always of the
569          * native type; if we are in the _ELF32_COMPAT case,
570          * we need to explicitly convert it.
571          */
572         if (p->p_model == DATAMODEL_ILP32) {
573             psinfo_kto32(psaddr, &bigwad->psinfo);
574             psaddr = &bigwad->psinfo;
575         }
576 #endif
577
578         error = elfnote(vp, &offset, NT_SPYMASTER,
579             sizeof (psinfo_t), psaddr, rlimit, credp);
580         if (error)
581             goto done;
582     }
583 }
584 ASSERT(nlwp == 0);

```

```

586 done:
587     kmem_free(bigwad, bigsize);
588     return (error);
589 }

```



```

*****
15008 Thu Jun 30 22:03:20 2016
new/usr/src/uts/common/fs/proc/prdata.h
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
_____unchanged_portion_omitted_____

85 /* prc_flags */
86 #define PRC_DESTROY 0x01 /* process or lwp is being destroyed */
87 #define PRC_LWP 0x02 /* structure refers to an lwp */
88 #define PRC_SYS 0x04 /* process is a system process */
89 #define PRC_POLL 0x08 /* poll() in progress on this process/lwp */
90 #define PRC_EXCL 0x10 /* exclusive access granted (old /proc) */

92 /*
93 * Macros for mapping between i-numbers and pids.
94 */
95 #define pmkino(tslot, pslot, nodetype) \
96     (((ino_t)(tslot) << nproc_highbit) | \
97      (ino_t)(pslot) << 6) | \
98      (nodetype) + 2)

100 /* for old /proc interface */
101 #define PRBIAS 64
102 #define ptoi(n) ((int)((n) + PRBIAS)) /* pid to i-number */

104 /*
105 * Node types for /proc files (directories and files contained therein).
106 */
107 typedef enum prnodetype {
108     PR_PROCDIR, /* /proc */
109     PR_SELF, /* /proc/self */
110     PR_PIDDIR, /* /proc/<pid> */
111     PR_AS, /* /proc/<pid>/as */
112     PR_CTL, /* /proc/<pid>/ctl */
113     PR_STATUS, /* /proc/<pid>/status */
114     PR_LSTATUS, /* /proc/<pid>/lstatus */
115     PR_PSINFO, /* /proc/<pid>/psinfo */
116     PR_LPSINFO, /* /proc/<pid>/lpsinfo */
117     PR_MAP, /* /proc/<pid>/map */
118     PR_RMAP, /* /proc/<pid>/rmap */
119     PR_XMAP, /* /proc/<pid>/xmap */
120     PR_CRED, /* /proc/<pid>/cred */
121     PR_SIGACT, /* /proc/<pid>/sigact */
122     PR_AUXV, /* /proc/<pid>/auxv */
123     #if defined(__i386) || defined(__amd64)
124     PR_LDT, /* /proc/<pid>/ldt */
125     #endif
126     PR_USAGE, /* /proc/<pid>/usage */
127     PR_LUSAGE, /* /proc/<pid>/lusage */
128     PR_PAGEDATA, /* /proc/<pid>/pagedata */
129     PR_WATCH, /* /proc/<pid>/watch */
130     PR_CURDIR, /* /proc/<pid>/cwd */
131     PR_ROOTDIR, /* /proc/<pid>/root */
132     PR_FDDIR, /* /proc/<pid>/fd */
133     PR_FD, /* /proc/<pid>/fd/nn */
134     PR_OBJECTDIR, /* /proc/<pid>/object */
135     PR_OBJECT, /* /proc/<pid>/object/xxx */
136     PR_LWDIR, /* /proc/<pid>/lwp */
137     PR_LWPIDDIR, /* /proc/<pid>/lwp/<lwpid> */
138     PR_LWPCTL, /* /proc/<pid>/lwp/<lwpid>/lwpctl */
139     PR_LWPSTATUS, /* /proc/<pid>/lwp/<lwpid>/lwpstatus */
140     PR_LWPSINFO, /* /proc/<pid>/lwp/<lwpid>/lwpsinfo */

```

```

141     PR_LWPUSE, /* /proc/<pid>/lwp/<lwpid>/lwpusage */
142     PR_XREGS, /* /proc/<pid>/lwp/<lwpid>/xregs */
143     PR_TMPLDIR, /* /proc/<pid>/lwp/<lwpid>/templates */
144     PR_TMPL, /* /proc/<pid>/lwp/<lwpid>/templates/<id> */
145     PR_SPYMASTER, /* /proc/<pid>/lwp/<lwpid>/spymaster */
146     #if defined(__sparc)
147     PR_GWINDOWS, /* /proc/<pid>/lwp/<lwpid>/gwindows */
148     PR_ASRS, /* /proc/<pid>/lwp/<lwpid>/asrs */
149     #endif
150     PR_PRIV, /* /proc/<pid>/priv */
151     PR_PATHDIR, /* /proc/<pid>/path */
152     PR_PATH, /* /proc/<pid>/path/xxx */
153     PR_CTDIR, /* /proc/<pid>/contracts */
154     PR_CT, /* /proc/<pid>/contracts/<ctid> */
155     PR_SECFLAGS, /* /proc/<pid>/secflags */
156     #endif /* ! codereview */
157     PR_PIDFILE, /* old process file */
158     PR_LWPIDFILE, /* old lwp file */
159     PR_OPAGEDATA, /* old page data file */
160     PR_NFILES, /* number of /proc node types */
161 } prnodetype_t;

163 typedef struct prnode {
164     vnode_t *pr_next; /* list of all vnodes for process */
165     uint_t pr_flags; /* private flags */
166     kmutex_t pr_mutex; /* locks pr_files and child pr_flags */
167     prnodetype_t pr_type; /* node type */
168     mode_t pr_mode; /* file mode */
169     ino_t pr_ino; /* node id (for stat(2)) */
170     uint_t pr_hatid; /* hat layer id for page data files */
171     prcommon_t *pr_common; /* common data structure */
172     prcommon_t *pr_pcommon; /* process common data structure */
173     vnode_t *pr_parent; /* parent directory */
174     vnode_t **pr_files; /* contained files array (directory) */
175     uint_t pr_index; /* position within parent */
176     vnode_t *pr_pidfile; /* substitute vnode for old /proc */
177     vnode_t *pr_realvp; /* real vnode, file in object, fd dirs */
178     proc_t pr_owner; /* the process that created this node */
179     vnode_t *pr_vnode; /* pointer to vnode */
180     struct contract *pr_contract; /* contract pointer */
181     int pr_cttype; /* active template type */
182 } prnode_t;

184 /*
185 * Values for pr_flags.
186 */
187 #define PR_INVALID 0x01 /* vnode is invalidated */
188 #define PR_ISSELF 0x02 /* vnode is a self-open */
189 #define PR_AOUT 0x04 /* vnode is for an a.out path */
190 #define PR_OFFMAX 0x08 /* vnode is a large file open */

192 /*
193 * Conversion macros.
194 */
195 #define VTOP(vp) ((struct prnode *) (vp)->v_data)
196 #define PTOV(pnp) ((pnp)->pr_vnode)

198 /*
199 * Flags to prlock().
200 */
201 #define ZNO 0 /* Fail on encountering a zombie process. */
202 #define ZYES 1 /* Allow zombies. */

204 /*
205 * Assign one set to another (possible different sizes).
206 */

```

```

207 * Assigning to a smaller set causes members to be lost.
208 * Assigning to a larger set causes extra members to be cleared.
209 */
210 #define prassignset(ap, sp) \
211 { \
212     register int _i = sizeof (*(ap))/sizeof (uint32_t); \
213     while (--_i >= 0) \
214         ((uint32_t *) (ap))[_i] = \
215             (_i >= sizeof (*(sp))/sizeof (uint32_t)) ? \
216             0 : ((uint32_t *) (sp))[_i]; \
217 }
219 /*
220 * Determine whether or not a set (of arbitrary size) is empty.
221 */
222 #define prisemply(sp) \
223     setisemply((uint32_t *) (sp), \
224         (uint_t)(sizeof (*(sp)) / sizeof (uint32_t)))
226 /*
227 * Resource usage with times as hrtime_t rather than timestruc_t.
228 * Each member exactly matches the corresponding member in prusage_t.
229 * This is for convenience of internal computation.
230 */
231 typedef struct prusage {
232     id_t      pr_lwpid;      /* lwp id.  0: process or defunct */
233     int       pr_count;     /* number of contributing lwps */
234     hrtime_t  pr_tstamp;    /* current time stamp */
235     hrtime_t  pr_create;   /* process/lwp creation time stamp */
236     hrtime_t  pr_term;     /* process/lwp termination time stamp */
237     hrtime_t  pr_rtime;    /* total lwp real (elapsed) time */
238     hrtime_t  pr_utime;    /* user level CPU time */
239     hrtime_t  pr_stime;    /* system call CPU time */
240     hrtime_t  pr_ttime;    /* other system trap CPU time */
241     hrtime_t  pr_tftime;   /* text page fault sleep time */
242     hrtime_t  pr_dftime;   /* data page fault sleep time */
243     hrtime_t  pr_kftime;   /* kernel page fault sleep time */
244     hrtime_t  pr_ltime;    /* user lock wait sleep time */
245     hrtime_t  pr_slptime;  /* all other sleep time */
246     hrtime_t  pr_wtime;    /* wait-cpu (latency) time */
247     hrtime_t  pr_stoptime; /* stopped time */
248     hrtime_t  pr_filltime[6]; /* filler for future expansion */
249     uint64_t  pr_minf;     /* minor page faults */
250     uint64_t  pr_majf;    /* major page faults */
251     uint64_t  pr_nswap;   /* swaps */
252     uint64_t  pr_inblk;   /* input blocks */
253     uint64_t  pr_oublk;   /* output blocks */
254     uint64_t  pr_msnd;    /* messages sent */
255     uint64_t  pr_mrcv;    /* messages received */
256     uint64_t  pr_sigs;    /* signals received */
257     uint64_t  pr_vctx;    /* voluntary context switches */
258     uint64_t  pr_ictx;    /* involuntary context switches */
259     uint64_t  pr_sysc;    /* system calls */
260     uint64_t  pr_ioch;    /* chars read and written */
261     uint64_t  pr_filler[10]; /* filler for future expansion */
262 } prusage_t;
264 #if defined(_KERNEL)
266 /* Exclude system processes from this test */
267 #define PROCESS_NOT_32BIT(p) \
268     (!(p->p_flag & SSYS) && (p->p_as != &kas && \
269     (p->p_model != DATAMODEL_ILP32)
271 extern int prnwatch; /* number of supported watchpoints */
272 extern int nproc_highbit; /* highbit(v.v_nproc) */

```

```

274 extern struct vnodeops *prvnodeops;
276 /*
277 * Generic chained copyout buffers for procfs use.
278 * In order to prevent procfs from making huge oversize kmem_alloc calls,
279 * a list of smaller buffers can be concatenated and copied to userspace in
280 * sequence.
281 *
282 * The implementation is opaque.
283 *
284 * A user of this will perform the following steps:
285 *
286 *     list_t listhead;
287 *     struct my *mp;
288 *
289 *     pr_iol_initlist(&listhead, sizeof (*mp), n);
290 *     while (whatever) {
291 *         mp = pr_iol_newbuf(&listhead, sizeof (*mp);
292 *         ...
293 *         error = ...
294 *     }
295 *
296 * When done, depending on whether copyout() or uiomove() is supposed to
297 * be used for transferring the buffered data to userspace, call either:
298 *
299 *     error = pr_iol_copyout_and_free(&listhead, &caddr, error);
300 *
301 * or else:
302 *
303 *     error = pr_iol_uiomove_and_free(&listhead, uiop, error);
304 *
305 * These two functions will in any case kmem_free() all list items, but
306 * if an error occurred before they will not perform the copyout/uiomove.
307 * If copyout/uiomove are done, the passed target address / uiop_t
308 * are updated. The error returned will either be the one passed in, or
309 * the error that occurred during copyout/uiomove.
310 */
312 extern void pr_iol_initlist(list_t *head, size_t itemsize, int nitems);
313 extern void * pr_iol_newbuf(list_t *head, size_t itemsize);
314 extern int pr_iol_copyout_and_free(list_t *head, caddr_t *tgt, int errin);
315 extern int pr_iol_uiomove_and_free(list_t *head, uiop_t *uiop, int errin);
317 #if defined(_SYSCALL32_IMPL)
319 extern int prwritel32(vnode_t *, struct uio *, cred_t *);
320 extern void prgetaction32(proc_t *, user_t *, uint_t, struct sigaction32 *);
321 extern void prcvtusage32(struct prusage *, prusage32_t *);
323 #endif /* _SYSCALL32_IMPL */
325 /* kludge to support old /proc interface */
326 #if !defined(_SYS_OLD_PROCFS_H)
327 extern int prgetmap(proc_t *, int, list_t *);
328 extern int prgetxmap(proc_t *, list_t *);
329 #if defined(_SYSCALL32_IMPL)
330 extern int prgetmap32(proc_t *, int, list_t *);
331 extern int prgetxmap32(proc_t *, list_t *);
332 #endif /* _SYSCALL32_IMPL */
333 #endif /* !_SYS_OLD_PROCFS_H */
335 extern proc_t *pr_p_lock(prnode_t *);
336 extern kthread_t *pr_thread(prnode_t *);
337 extern void pr_stop(prnode_t *);
338 extern int pr_wait_stop(prnode_t *, time_t);

```

```

339 extern int pr_setrun(prnode_t *, ulong_t);
340 extern int pr_wait(prcommon_t *, timestruc_t *, int);
341 extern void pr_wait_die(prnode_t *);
342 extern int pr_setsig(prnode_t *, siginfo_t *);
343 extern int pr_kill(prnode_t *, int, cred_t *);
344 extern int pr_unkill(prnode_t *, int);
345 extern int pr_nice(proc_t *, int, cred_t *);
346 extern void pr_setentryexit(proc_t *, sysset_t *, int);
347 extern int pr_set(proc_t *, long);
348 extern int pr_unset(proc_t *, long);
349 extern void pr_sethold(prnode_t *, sigset_t *);
350 extern void pr_setfault(proc_t *, fltset_t *);
351 extern int prusrrio(proc_t *, enum uio_rw, struct uio *, int);
352 extern int prwritectl(vnode_t *, struct uio *, cred_t *);
353 extern int prlock(prnode_t *, int);
354 extern void prunmark(proc_t *);
355 extern void prunlock(prnode_t *);
356 extern size_t prpdsz(struct as *);
357 extern int prpdread(proc_t *, uint_t, struct uio *);
358 extern size_t oprpdsz(struct as *);
359 extern int oprpdread(struct as *, uint_t, struct uio *);
360 extern void prgetaction(proc_t *, user_t *, uint_t, struct sigaction *);
361 extern void prgetusage(kthread_t *, struct prusage *);
362 extern void praddusage(kthread_t *, struct prusage *);
363 extern void prcvusage(struct prusage *, prusage_t *);
364 extern void prscaleusage(prusage_t *);
365 extern kthread_t *prchoose(proc_t *);
366 extern void allsetrun(proc_t *);
367 extern int setisempty(uint32_t *, uint_t);
368 extern int pr_u32tos(uint32_t, char *, int);
369 extern vnode_t *prlwnode(prnode_t *, uint_t);
370 extern prnode_t *prgetnode(vnode_t *, prnodetype_t);
371 extern void prfreenode(prnode_t *);
372 extern void pr_object_name(char *, vnode_t *, struct vattr *);
373 extern int set_watched_area(proc_t *, struct watched_area *);
374 extern int clear_watched_area(proc_t *, struct watched_area *);
375 extern void pr_free_watchpoints(proc_t *);
376 extern proc_t *pr_cancel_watch(prnode_t *);
377 extern struct seg *break_seg(proc_t *);

379 /*
380  * Machine-dependent routines (defined in prmachdep.c).
381  */
382 extern void prgetprregs(klwp_t *, prgregset_t);
383 extern void prsetprregs(klwp_t *, prgregset_t, int);

385 #if defined(_SYSCALL32_IMPL)
386 extern void prgetprregs32(klwp_t *, prgregset32_t);
387 extern void prgregset_32ton(klwp_t *, prgregset32_t, prgregset_t);
388 extern void prgetprfpregs32(klwp_t *, prfpregsset32_t *);
389 extern void prsetprfpregs32(klwp_t *, prfpregsset32_t *);
390 extern size_t prpdsz32(struct as *);
391 extern int prpdread32(proc_t *, uint_t, struct uio *);
392 extern size_t oprpdsz32(struct as *);
393 extern int oprpdread32(struct as *, uint_t, struct uio *);
394 #endif

396 extern void prpokethread(kthread_t *);
397 extern int prgetrvals(klwp_t *, long *, long *);
398 extern void prgetprfpregs(klwp_t *, prfpregsset_t *);
399 extern void prsetprfpregs(klwp_t *, prfpregsset_t *);
400 extern void prgetprxregs(klwp_t *, caddr_t);
401 extern void prsetprxregs(klwp_t *, caddr_t);
402 extern int prgetprxregsize(proc_t *);
403 extern int prhasfp(void);
404 extern int prhasx(proc_t *);

```

```

405 extern caddr_t prgetstackbase(proc_t *);
406 extern caddr_t prgetpsaddr(proc_t *);
407 extern int prisstep(klwp_t *);
408 extern void prsvaddr(klwp_t *, caddr_t);
409 extern int prfetchinstr(klwp_t *, ulong_t *);
410 extern ushort_t prgetpctcpu(uint64_t);

412 #endif /* _KERNEL */

414 #ifdef __cplusplus
415 }
416 #endif

418 #endif /* _SYS_PROC_PRDATA_H */

```

```

*****
112542 Thu Jun 30 22:03:21 2016
new/usr/src/uts/common/fs/proc/prsubr.c
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
_____unchanged_portion_omitted_____

4158 void
4159 prgetsecflags(proc_t *p, prsecflags_t *psfp)
4160 {
4161     ASSERT(psfp != NULL);

4163     psfp->pr_version = PRSECFLAGS_VERSION_CURRENT;
4164     psfp->pr_lower = p->p_secflags.psf_lower;
4165     psfp->pr_upper = p->p_secflags.psf_upper;
4166     psfp->pr_effective = p->p_secflags.psf_effective;
4167     psfp->pr_inherit = p->p_secflags.psf_inherit;
4168 }

4170 #endif /* ! codereview */
4171 /*
4172  * Compute actual size of the prpriv_t structure.
4173  */

4175 size_t
4176 prgetprivsize(void)
4177 {
4178     return (priv_prgetprivsize(NULL));
4179 }

4181 /*
4182  * Return the process's privileges. We don't need a 32-bit equivalent of
4183  * this function because prpriv_t and prpriv32_t are actually the same.
4184  */
4185 void
4186 prgetpriv(proc_t *p, prpriv_t *pprp)
4187 {
4188     mutex_enter(&p->p_crlock);
4189     cred2prpriv(p->p_cred, pprp);
4190     mutex_exit(&p->p_crlock);
4191 }

4193 #ifdef _SYSCALL32_IMPL
4194 /*
4195  * Return an array of structures with HAT memory map information.
4196  * We allocate here; the caller must deallocate.
4197  */
4198 int
4199 prgetxmap32(proc_t *p, list_t *iolhead)
4200 {
4201     struct as *as = p->p_as;
4202     prxmap32_t *mp;
4203     struct seg *seg;
4204     struct seg *brkseg, *stkseg;
4205     struct vnode *vp;
4206     struct vattr vattr;
4207     uint_t prot;

4209     ASSERT(as != &kas && AS_WRITE_HELD(as));

4211     /*
4212      * Request an initial buffer size that doesn't waste memory
4213      * if the address space has only a small number of segments.

```

```

4214     /*
4215     pr_iol_initlist(iolhead, sizeof (*mp), avl_numnodes(&as->a_segtree));

4217     if ((seg = AS_SEGFIRST(as)) == NULL)
4218         return (0);

4220     brkseg = break_seg(p);
4221     stkseg = as_segat(as, prgetstackbase(p));

4223     do {
4224         caddr_t eaddr = seg->s_base + pr_getsegsz(seg, 0);
4225         caddr_t saddr, naddr, baddr;
4226         void *tmp = NULL;
4227         ssize_t psz;
4228         char *parr;
4229         uint64_t npages;
4230         uint64_t pagenum;

4232         /*
4233          * Segment loop part one: iterate from the base of the segment
4234          * to its end, pausing at each address boundary (baddr) between
4235          * ranges that have different virtual memory protections.
4236          */
4237         for (saddr = seg->s_base; saddr < eaddr; saddr = baddr) {
4238             prot = pr_getprot(seg, 0, &tmp, &saddr, &baddr, eaddr);
4239             ASSERT(baddr >= saddr && baddr <= eaddr);

4241             /*
4242              * Segment loop part two: iterate from the current
4243              * position to the end of the protection boundary,
4244              * pausing at each address boundary (naddr) between
4245              * ranges that have different underlying page sizes.
4246              */
4247             for (; saddr < baddr; saddr = naddr) {
4248                 psz = pr_getpagesize(seg, saddr, &naddr, baddr);
4249                 ASSERT(naddr >= saddr && naddr <= baddr);

4251                 mp = pr_iol_newbuf(iolhead, sizeof (*mp));

4253                 mp->pr_vaddr = (caddr32_t)(uintptr_t)saddr;
4254                 mp->pr_size = (size32_t)(naddr - saddr);
4255                 mp->pr_offset = SEGOP_GETOFFSET(seg, saddr);
4256                 mp->pr_mflags = 0;
4257                 if (prot & PROT_READ)
4258                     mp->pr_mflags |= MA_READ;
4259                 if (prot & PROT_WRITE)
4260                     mp->pr_mflags |= MA_WRITE;
4261                 if (prot & PROT_EXEC)
4262                     mp->pr_mflags |= MA_EXEC;
4263                 if (SEGOP_GETTYPE(seg, saddr) & MAP_SHARED)
4264                     mp->pr_mflags |= MA_SHARED;
4265                 if (SEGOP_GETTYPE(seg, saddr) & MAP_NORESERVE)
4266                     mp->pr_mflags |= MA_NORESERVE;
4267                 if (seg->s_ops == &segspt_shmops ||
4268                     (seg->s_ops == &segvn_ops &&
4269                     (SEGOP_GETVP(seg, saddr, &vp) != 0 ||
4270                     vp == NULL)))
4271                     mp->pr_mflags |= MA_ANON;
4272                 if (seg == brkseg)
4273                     mp->pr_mflags |= MA_BREAK;
4274                 else if (seg == stkseg)
4275                     mp->pr_mflags |= MA_STACK;
4276                 if (seg->s_ops == &segspt_shmops)
4277                     mp->pr_mflags |= MA_ISM | MA_SHM;

4279                 mp->pr_pagesize = PAGESIZE;

```

```

4280     if (psz == -1) {
4281         mp->pr_hatpagesize = 0;
4282     } else {
4283         mp->pr_hatpagesize = psz;
4284     }
4285
4286     /*
4287      * Manufacture a filename for the "object" dir.
4288      */
4289     mp->pr_dev = PRNODEV32;
4290     vattr.va_mask = AT_FSID|AT_NODEID;
4291     if (seg->s_ops == &segn_ops &&
4292         SEGOP_GETVP(seg, saddr, &vp) == 0 &&
4293         vp != NULL && vp->v_type == VREG &&
4294         VOP_GETATTR(vp, &vattr, 0, CRED(),
4295             NULL) == 0) {
4296         (void) cmldev(&mp->pr_dev,
4297             vattr.va_fsid);
4298         mp->pr_ino = vattr.va_nodeid;
4299         if (vp == p->p_exec)
4300             (void) strcpy(mp->pr_mapname,
4301                 "a.out");
4302         else
4303             pr_object_name(mp->pr_mapname,
4304                 vp, &vattr);
4305     }
4306
4307     /*
4308      * Get the SysV shared memory id, if any.
4309      */
4310     if ((mp->pr_mflags & MA_SHARED) &&
4311         p->p_segacct && (mp->pr_shmid = shmgetid(p,
4312             seg->s_base) != SHMID_NONE) {
4313         if (mp->pr_shmid == SHMID_FREE)
4314             mp->pr_shmid = -1;
4315
4316         mp->pr_mflags |= MA_SHM;
4317     } else {
4318         mp->pr_shmid = -1;
4319     }
4320
4321     npages = ((uintptr_t)(naddr - saddr)) >>
4322         PAGESHIFT;
4323     parr = kmem_zalloc(npages, KM_SLEEP);
4324
4325     SEGOP_INCORE(seg, saddr, naddr - saddr, parr);
4326
4327     for (pagenum = 0; pagenum < npages; pagenum++) {
4328         if (parr[pagenum] & SEG_PAGE_INCORE)
4329             mp->pr_rss++;
4330         if (parr[pagenum] & SEG_PAGE_ANON)
4331             mp->pr_anon++;
4332         if (parr[pagenum] & SEG_PAGE_LOCKED)
4333             mp->pr_locked++;
4334     }
4335     kmem_free(parr, npages);
4336 }
4337 }
4338     ASSERT(tmp == NULL);
4339 } while ((seg = AS_SEGNEXT(as, seg)) != NULL);
4340
4341     return (0);
4342 }
4343 #endif /* _SYSCALL32_IMPL */

```

```

*****
143316 Thu Jun 30 22:03:23 2016
new/usr/src/uts/common/fs/proc/prvnops.c
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
    unchanged_portion_omitted_

108 /*
109  * Contents of a /proc/<pid> directory.
110  * Reuse d_ino field for the /proc file type.
111  */
112 static prdirent_t piddir[] = {
113     { PR_PIDDIR, 1 * sizeof (prdirent_t), sizeof (prdirent_t),
114       " " },
115     { PR_PROCDIR, 2 * sizeof (prdirent_t), sizeof (prdirent_t),
116       "." },
117     { PR_AS, "as" }, 3 * sizeof (prdirent_t), sizeof (prdirent_t),
118     { PR_CTL, "ctl" }, 4 * sizeof (prdirent_t), sizeof (prdirent_t),
119     { PR_STATUS, "status" }, 5 * sizeof (prdirent_t), sizeof (prdirent_t),
120     { PR_LSTATUS, "lstatus" }, 6 * sizeof (prdirent_t), sizeof (prdirent_t),
121     { PR_PSINFO, "psinfo" }, 7 * sizeof (prdirent_t), sizeof (prdirent_t),
122     { PR_LPSINFO, "lpsinfo" }, 8 * sizeof (prdirent_t), sizeof (prdirent_t),
123     { PR_MAP, "map" }, 9 * sizeof (prdirent_t), sizeof (prdirent_t),
124     { PR_RMAP, "rmap" }, 10 * sizeof (prdirent_t), sizeof (prdirent_t),
125     { PR_XMAP, "xmap" }, 11 * sizeof (prdirent_t), sizeof (prdirent_t),
126     { PR_CRED, "cred" }, 12 * sizeof (prdirent_t), sizeof (prdirent_t),
127     { PR_SIGACT, "sigact" }, 13 * sizeof (prdirent_t), sizeof (prdirent_t),
128     { PR_AUXV, "auxv" }, 14 * sizeof (prdirent_t), sizeof (prdirent_t),
129     { PR_USAGE, "usage" }, 15 * sizeof (prdirent_t), sizeof (prdirent_t),
130     { PR_LUSAGE, "lusage" }, 16 * sizeof (prdirent_t), sizeof (prdirent_t),
131     { PR_PAGEDATA, "pagedata" }, 17 * sizeof (prdirent_t), sizeof (prdirent_t),
132     { PR_WATCH, "watch" }, 18 * sizeof (prdirent_t), sizeof (prdirent_t),
133     { PR_CURDIR, "cwd" }, 19 * sizeof (prdirent_t), sizeof (prdirent_t),
134     { PR_ROOTDIR, "root" }, 20 * sizeof (prdirent_t), sizeof (prdirent_t),
135     { PR_FDDIR, "fd" }, 21 * sizeof (prdirent_t), sizeof (prdirent_t),
136     { PR_OBJCTDIR, "object" }, 22 * sizeof (prdirent_t), sizeof (prdirent_t),
137     { PR_LWPDIR, "lwp" }, 23 * sizeof (prdirent_t), sizeof (prdirent_t),
138     { PR_PRIV, "priv" }, 24 * sizeof (prdirent_t), sizeof (prdirent_t),
139     { PR_PATHDIR, "path" }, 25 * sizeof (prdirent_t), sizeof (prdirent_t),
140     { PR_CTDIR, "contracts" }, 26 * sizeof (prdirent_t), sizeof (prdirent_t),
141     { PR_SECFLAGS, "secflags" }, 27 * sizeof (prdirent_t), sizeof (prdirent_t),
142     #endif /* ! codereview */
143     #if defined(__x86)
144     { PR_LDT, "ldt" }, 28 * sizeof (prdirent_t), sizeof (prdirent_t),
145     { PR_READ, "read" }, 29 * sizeof (prdirent_t), sizeof (prdirent_t),
146     #endif
147 };
    unchanged_portion_omitted_

```

```

164     "contracts" },
165     { PR_SECFLAGS, 27 * sizeof (prdirent_t), sizeof (prdirent_t),
166       "secflags" },
167 #endif /* ! codereview */
168 #if defined(__x86)
169     { PR_LDT, 28 * sizeof (prdirent_t), sizeof (prdirent_t),
170       "ldt" },
171 #endif
172 };
    unchanged_portion_omitted_

578 /*
579  * Array of read functions, indexed by /proc file type.
580  */
581 static int pr_read_inval(), pr_read_as(), pr_read_status(),
582         pr_read_lstatus(), pr_read_psinfo(), pr_read_lpsinfo(),
583         pr_read_map(), pr_read_rmap(), pr_read_xmap(),
584         pr_read_cred(), pr_read_sigact(), pr_read_auxv(),
585 #if defined(__x86)
586         pr_read_ldt(),
587 #endif
588         pr_read_usage(), pr_read_lusage(), pr_read_pagedata(),
589         pr_read_watch(), pr_read_lwpstatus(), pr_read_lwpsinfo(),
590         pr_read_lwpusage(), pr_read_xregs(), pr_read_priv(),
591         pr_read_spymaster(), pr_read_secflags(),
592         pr_read_spymaster(),
593 #if defined(__sparc)
594         pr_read_gwindows(), pr_read_asrs(),
595 #endif
596         pr_read_piddir(), pr_read_pidfile(), pr_read_opagedata();

597 static int (*pr_read_function[PR_NFILES])() = {
598     pr_read_inval, /* /proc */
599     pr_read_inval, /* /proc/self */
600     pr_read_piddir, /* /proc/<pid> (old /proc read()) */
601     pr_read_as, /* /proc/<pid>/as */
602     pr_read_inval, /* /proc/<pid>/ctl */
603     pr_read_status, /* /proc/<pid>/status */
604     pr_read_lstatus, /* /proc/<pid>/lstatus */
605     pr_read_psinfo, /* /proc/<pid>/psinfo */
606     pr_read_lpsinfo, /* /proc/<pid>/lpsinfo */
607     pr_read_map, /* /proc/<pid>/map */
608     pr_read_rmap, /* /proc/<pid>/rmap */
609     pr_read_xmap, /* /proc/<pid>/xmap */
610     pr_read_cred, /* /proc/<pid>/cred */
611     pr_read_sigact, /* /proc/<pid>/sigact */
612     pr_read_auxv, /* /proc/<pid>/auxv */
613 #if defined(__x86)
614     pr_read_ldt, /* /proc/<pid>/ldt */
615 #endif
616     pr_read_usage, /* /proc/<pid>/usage */
617     pr_read_lusage, /* /proc/<pid>/lusage */
618     pr_read_pagedata, /* /proc/<pid>/pagedata */
619     pr_read_watch, /* /proc/<pid>/watch */
620     pr_read_inval, /* /proc/<pid>/cwd */
621     pr_read_inval, /* /proc/<pid>/root */
622     pr_read_inval, /* /proc/<pid>/fd */
623     pr_read_inval, /* /proc/<pid>/fd/nn */
624     pr_read_inval, /* /proc/<pid>/object */
625     pr_read_inval, /* /proc/<pid>/object/xxx */
626     pr_read_inval, /* /proc/<pid>/lwp */
627     pr_read_inval, /* /proc/<pid>/lwp/<lwpid> */
628     pr_read_inval, /* /proc/<pid>/lwp/<lwpid>/lwpctl */
629     pr_read_lwpstatus, /* /proc/<pid>/lwp/<lwpid>/lwpstatus */
630     pr_read_lwpsinfo, /* /proc/<pid>/lwp/<lwpid>/lwpsinfo */

```

```

631     pr_read_lwpusage,      /* /proc/<pid>/lwp/<lwpid>/lwpusage */
632     pr_read_xregs,        /* /proc/<pid>/lwp/<lwpid>/xregs */
633     pr_read_inval,        /* /proc/<pid>/lwp/<lwpid>/templates */
634     pr_read_inval,        /* /proc/<pid>/lwp/<lwpid>/templates/<id> */
635     pr_read_spymaster,    /* /proc/<pid>/lwp/<lwpid>/spymaster */
636 #if defined(__sparc)
637     pr_read_gwindows,     /* /proc/<pid>/lwp/<lwpid>/gwindows */
638     pr_read_asrs,        /* /proc/<pid>/lwp/<lwpid>/asrs */
639 #endif
640     pr_read_priv,         /* /proc/<pid>/priv */
641     pr_read_inval,        /* /proc/<pid>/path */
642     pr_read_inval,        /* /proc/<pid>/path/xxx */
643     pr_read_inval,        /* /proc/<pid>/contracts */
644     pr_read_inval,        /* /proc/<pid>/contracts/<ctid> */
645     pr_read_secflags,     /* /proc/<pid>/secflags */
646 #endif /* ! codereview */
647     pr_read_pidfile,      /* old process file */
648     pr_read_pidfile,      /* old lwp file */
649     pr_read_opagedata,    /* old pagedata file */
650 };

652 /* ARGSUSED */
653 static int
654 pr_read_inval(prnode_t *pnp, uio_t *uiop)
655 {
656     /*
657      * No read() on any /proc directory, use getdents(2) instead.
658      * Cannot read a control file either.
659      * An underlying mapped object file cannot get here.
660      */
661     return (EINVAL);
662 }

664 static int
665 pr_uioread(void *base, long count, uio_t *uiop)
666 {
667     int error = 0;

669     ASSERT(count >= 0);
670     count -= uiop->uio_offset;
671     if (count > 0 && uiop->uio_offset >= 0) {
672         error = uiomove((char *)base + uiop->uio_offset,
673             count, UIO_READ, uiop);
674     }

676     return (error);
677 }

679 static int
680 pr_read_as(prnode_t *pnp, uio_t *uiop)
681 {
682     int error;

684     ASSERT(pnp->pr_type == PR_AS);

686     if ((error = prlock(pnp, ZNO)) == 0) {
687         proc_t *p = pnp->pr_common->prc_proc;
688         struct as *as = p->p_as;

690         /*
691          * /proc I/O cannot be done to a system process.
692          * A 32-bit process cannot read a 64-bit process.
693          */
694         if ((p->p_flag & SSYS) || as == &kas) {
695             error = 0;
696 #ifndef _SYSCALL32_IMPL

```

```

697     } else if (curproc->p_model == DATAMODEL_ILP32 &&
698         PROCESS_NOT_32BIT(p)) {
699         error = EOVERFLOW;
700 #endif
701     } else {
702         /*
703          * We don't hold p_lock over an i/o operation because
704          * that could lead to deadlock with the clock thread.
705          */
706         mutex_exit(&p->p_lock);
707         error = prurio(p, UIO_READ, uiop, 0);
708         mutex_enter(&p->p_lock);
709     }
710     prunlock(pnp);
711 }

713     return (error);
714 }

716 static int
717 pr_read_status(prnode_t *pnp, uio_t *uiop)
718 {
719     pstatus_t *sp;
720     int error;

722     ASSERT(pnp->pr_type == PR_STATUS);

724     /*
725      * We kmem_alloc() the pstatus structure because
726      * it is so big it might blow the kernel stack.
727      */
728     sp = kmem_alloc(sizeof (*sp), KM_SLEEP);
729     if ((error = prlock(pnp, ZNO)) == 0) {
730         prgetstatus(pnp->pr_common->prc_proc, sp, VTOZONE(PTOV(pnp)));
731         prunlock(pnp);
732         error = pr_uioread(sp, sizeof (*sp), uiop);
733     }
734     kmem_free(sp, sizeof (*sp));
735     return (error);
736 }

738 static int
739 pr_read_lstatus(prnode_t *pnp, uio_t *uiop)
740 {
741     proc_t *p;
742     kthread_t *t;
743     lwpdir_t *ldp;
744     size_t size;
745     prheader_t *php;
746     lwpstatus_t *sp;
747     int error;
748     int nlwp;
749     int i;

751     ASSERT(pnp->pr_type == PR_LSTATUS);

753     if ((error = prlock(pnp, ZNO)) != 0)
754         return (error);
755     p = pnp->pr_common->prc_proc;
756     nlwp = p->p_lwpcnt;
757     size = sizeof (prheader_t) + nlwp * LSPAN(lwpstatus_t);

759     /* drop p->p_lock to do kmem_alloc(KM_SLEEP) */
760     mutex_exit(&p->p_lock);
761     php = kmem_zalloc(size, KM_SLEEP);
762     mutex_enter(&p->p_lock);

```

```

763 /* p->p_lwpcnt can't change while process is locked */
764 ASSERT(nlwp == p->p_lwpcnt);

766 php->pr_nent = nlwp;
767 php->pr_entsize = LSPAN(lwpstatus_t);

769 sp = (lwpstatus_t *) (php + 1);
770 for (ldp = p->p_lwpdir, i = 0; i < p->p_lwpdir_sz; i++, ldp++) {
771     if (ldp->ld_entry == NULL ||
772         (t = ldp->ld_entry->le_thread) == NULL)
773         continue;
774     prgetlwpstatus(t, sp, VTOZONE(PTOV(pnp)));
775     sp = (lwpstatus_t *) ((caddr_t)sp + LSPAN(lwpstatus_t));
776 }
777 prunlock(pnp);

779 error = pr_uioread(php, size, uiop);
780 kmem_free(php, size);
781 return (error);
782 }

784 static int
785 pr_read_psinfnfo(prnode_t *pnp, uio_t *uiop)
786 {
787     psinfo_t psinfo;
788     proc_t *p;
789     int error = 0;

791     ASSERT(pnp->pr_type == PR_PSINFO);

793     /*
794      * We don't want the full treatment of prlock(pnp) here.
795      * This file is world-readable and never goes invalid.
796      * It doesn't matter if we are in the middle of an exec().
797      */
798     p = pr_p_lock(pnp);
799     mutex_exit(&pr_pidlock);
800     if (p == NULL)
801         error = ENOENT;
802     else {
803         ASSERT(p == pnp->pr_common->prc_proc);
804         prgetpsinfo(p, &psinfo);
805         prunlock(pnp);
806         error = pr_uioread(&psinfo, sizeof (psinfo), uiop);
807     }
808     return (error);
809 }

811 static int
812 pr_read_lpsinfo(prnode_t *pnp, uio_t *uiop)
813 {
814     proc_t *p;
815     kthread_t *t;
816     lwpdir_t *ldp;
817     lwpent_t *lep;
818     size_t size;
819     prheader_t *php;
820     lwpsinfo_t *sp;
821     int error;
822     int nlwp;
823     int i;

825     ASSERT(pnp->pr_type == PR_LPSINFO);

827     /*
828      * We don't want the full treatment of prlock(pnp) here.

```

```

829     * This file is world-readable and never goes invalid.
830     * It doesn't matter if we are in the middle of an exec().
831     */
832     p = pr_p_lock(pnp);
833     mutex_exit(&pr_pidlock);
834     if (p == NULL)
835         return (ENOENT);
836     ASSERT(p == pnp->pr_common->prc_proc);
837     if ((nlwp = p->p_lwpcnt + p->p_zombcnt) == 0) {
838         prunlock(pnp);
839         return (ENOENT);
840     }
841     size = sizeof (prheader_t) + nlwp * LSPAN(lwpsinfo_t);

843     /* drop p->p_lock to do kmem_alloc(KM_SLEEP) */
844     mutex_exit(&p->p_lock);
845     php = kmem_zalloc(size, KM_SLEEP);
846     mutex_enter(&p->p_lock);
847     /* p->p_lwpcnt can't change while process is locked */
848     ASSERT(nlwp == p->p_lwpcnt + p->p_zombcnt);

850     php->pr_nent = nlwp;
851     php->pr_entsize = LSPAN(lwpsinfo_t);

853     sp = (lwpsinfo_t *) (php + 1);
854     for (ldp = p->p_lwpdir, i = 0; i < p->p_lwpdir_sz; i++, ldp++) {
855         if ((lep = ldp->ld_entry) == NULL)
856             continue;
857         if ((t = lep->le_thread) != NULL)
858             prgetlwpsinfo(t, sp);
859         else {
860             bzero(sp, sizeof (*sp));
861             sp->pr_lwpid = lep->le_lwpid;
862             sp->pr_state = SZOMB;
863             sp->pr_sname = 'Z';
864             sp->pr_start.tv_sec = lep->le_start;
865             sp->pr_bindpro = PBIND_NONE;
866             sp->pr_bindpset = PS_NONE;
867         }
868         sp = (lwpsinfo_t *) ((caddr_t)sp + LSPAN(lwpsinfo_t));
869     }
870     prunlock(pnp);

872     error = pr_uioread(php, size, uiop);
873     kmem_free(php, size);
874     return (error);
875 }

877 static int
878 pr_read_map_common(prnode_t *pnp, uio_t *uiop, prnodetype_t type)
879 {
880     proc_t *p;
881     struct as *as;
882     list_t iolhead;
883     int error;

885     readmap_common:
886     if ((error = prlock(pnp, ZNO)) != 0)
887         return (error);

889     p = pnp->pr_common->prc_proc;
890     as = p->p_as;

892     if ((p->p_flag & SSYS) || as == &kas) {
893         prunlock(pnp);
894         return (0);

```



```

895     }
897     if (!AS_LOCK_TRYENTER(as, RW_WRITER)) {
898         prunlock(pnp);
899         delay(1);
900         goto readmap_common;
901     }
902     mutex_exit(&p->p_lock);

904     switch (type) {
905     case PR_XMAP:
906         error = prgetxmap(p, &iolhead);
907         break;
908     case PR_RMAP:
909         error = prgetmap(p, 1, &iolhead);
910         break;
911     case PR_MAP:
912         error = prgetmap(p, 0, &iolhead);
913         break;
914     }

916     AS_LOCK_EXIT(as);
917     mutex_enter(&p->p_lock);
918     prunlock(pnp);

920     error = pr_iol_uiomove_and_free(&iolhead, uiop, error);

922     return (error);
923 }

925 static int
926 pr_read_map(prnode_t *pnp, uio_t *uiop)
927 {
928     ASSERT(pnp->pr_type == PR_MAP);
929     return (pr_read_map_common(pnp, uiop, pnp->pr_type));
930 }

932 static int
933 pr_read_rmap(prnode_t *pnp, uio_t *uiop)
934 {
935     ASSERT(pnp->pr_type == PR_RMAP);
936     return (pr_read_map_common(pnp, uiop, pnp->pr_type));
937 }

939 static int
940 pr_read_xmap(prnode_t *pnp, uio_t *uiop)
941 {
942     ASSERT(pnp->pr_type == PR_XMAP);
943     return (pr_read_map_common(pnp, uiop, pnp->pr_type));
944 }

946 static int
947 pr_read_cred(prnode_t *pnp, uio_t *uiop)
948 {
949     proc_t *p;
950     prcred_t *pcrp;
951     int error;
952     size_t count;

954     ASSERT(pnp->pr_type == PR_CRED);

956     /*
957      * We kmem_alloc() the prcred_t structure because
958      * the number of supplementary groups is variable.
959      */
960     pcrp =

```

```

961         kmem_alloc(sizeof (prcred_t) + sizeof (gid_t) * (ngroups_max - 1),
962                 KM_SLEEP);

964     if ((error = prlock(pnp, ZNO)) != 0)
965         goto out;
966     p = pnp->pr_common->prc_proc;
967     ASSERT(p != NULL);

969     prgetcred(p, pcrp);
970     prunlock(pnp);

972     count = sizeof (prcred_t);
973     if (pcrp->pr_ngroups > 1)
974         count += sizeof (gid_t) * (pcrp->pr_ngroups - 1);
975     error = pr_uioread(pcrp, count, uiop);
976 out:
977     kmem_free(pcrp, sizeof (prcred_t) + sizeof (gid_t) * (ngroups_max - 1));
978     return (error);
979 }

981 static int
982 pr_read_priv(prnode_t *pnp, uio_t *uiop)
983 {
984     proc_t *p;
985     size_t psize = prgetprivsize();
986     ppriv_t *ppriv = kmem_alloc(psize, KM_SLEEP);
987     int error;

989     ASSERT(pnp->pr_type == PR_PRIV);

991     if ((error = prlock(pnp, ZNO)) != 0)
992         goto out;
993     p = pnp->pr_common->prc_proc;
994     ASSERT(p != NULL);

996     prgetpriv(p, ppriv);
997     prunlock(pnp);

999     error = pr_uioread(ppriv, psize, uiop);
1000 out:
1001     kmem_free(ppriv, psize);
1002     return (error);
1003 }

1005 static int
1006 pr_read_sigact(prnode_t *pnp, uio_t *uiop)
1007 {
1008     int nsig = PROC_IS_BRANDED(curproc)? BROP(curproc)->b_nsig : NSIG;
1009     proc_t *p;
1010     struct sigaction *sap;
1011     int sig;
1012     int error;
1013     user_t *up;

1015     ASSERT(pnp->pr_type == PR_SIGACT);

1017     /*
1018      * We kmem_alloc() the sigaction array because
1019      * it is so big it might blow the kernel stack.
1020      */
1021     sap = kmem_alloc((nsig-1) * sizeof (struct sigaction), KM_SLEEP);

1023     if ((error = prlock(pnp, ZNO)) != 0)
1024         goto out;
1025     p = pnp->pr_common->prc_proc;
1026     ASSERT(p != NULL);

```

```

1028     if (uiop->uio_offset >= (nsig-1)*sizeof (struct sigaction)) {
1029         prunlock(pnp);
1030         goto out;
1031     }
1033     up = PTOU(p);
1034     for (sig = 1; sig < nsig; sig++)
1035         prgetaction(p, up, sig, &sap[sig-1]);
1036     prunlock(pnp);
1038     error = pr_uioread(sap, (nsig - 1) * sizeof (struct sigaction), uiop);
1039 out:
1040     kmem_free(sap, (nsig-1) * sizeof (struct sigaction));
1041     return (error);
1042 }
1044 static int
1045 pr_read_auxv(prnode_t *pnp, uio_t *uiop)
1046 {
1047     auxv_t auxv[__KERN_NAUXV_IMPL];
1048     proc_t *p;
1049     user_t *up;
1050     int error;
1052     ASSERT(pnp->pr_type == PR_AUXV);
1054     if ((error = prlock(pnp, ZNO)) != 0)
1055         return (error);
1057     if (uiop->uio_offset >= sizeof (auxv)) {
1058         prunlock(pnp);
1059         return (0);
1060     }
1062     p = pnp->pr_common->prc_proc;
1063     up = PTOU(p);
1064     bcopy(up->u_auxv, auxv, sizeof (auxv));
1065     prunlock(pnp);
1067     return (pr_uioread(auxv, sizeof (auxv), uiop));
1068 }
1070 #if defined(__x86)
1071 /*
1072  * XX64
1073  * This is almost certainly broken for the amd64 kernel, because
1074  * we have two kinds of LDT structures to export -- one for compatibility
1075  * mode, and one for long mode, sigh.
1076  *
1077  * For now lets just have a ldt of size 0 for 64-bit processes.
1078  */
1079 static int
1080 pr_read_ldt(prnode_t *pnp, uio_t *uiop)
1081 {
1082     proc_t *p;
1083     struct ssd *ssd;
1084     size_t size;
1085     int error;
1087     ASSERT(pnp->pr_type == PR_LDT);
1089     if ((error = prlock(pnp, ZNO)) != 0)
1090         return (error);
1091     p = pnp->pr_common->prc_proc;

```

```

1093     mutex_exit(&p->p_lock);
1094     mutex_enter(&p->p_ldtlock);
1095     size = prldt(p) * sizeof (struct ssd);
1096     if (uiop->uio_offset >= size) {
1097         mutex_exit(&p->p_ldtlock);
1098         mutex_enter(&p->p_lock);
1099         prunlock(pnp);
1100         return (0);
1101     }
1103     ssd = kmem_alloc(size, KM_SLEEP);
1104     prgetldt(p, ssd);
1105     mutex_exit(&p->p_ldtlock);
1106     mutex_enter(&p->p_lock);
1107     prunlock(pnp);
1109     error = pr_uioread(ssd, size, uiop);
1110     kmem_free(ssd, size);
1111     return (error);
1112 }
1113 #endif /* __x86 */
1115 static int
1116 pr_read_usage(prnode_t *pnp, uio_t *uiop)
1117 {
1118     prhusage_t *pup;
1119     prusage_t *upup;
1120     proc_t *p;
1121     kthread_t *t;
1122     int error;
1124     ASSERT(pnp->pr_type == PR_USAGE);
1126     /* allocate now, before locking the process */
1127     pup = kmem_zalloc(sizeof (*pup), KM_SLEEP);
1128     upup = kmem_alloc(sizeof (*upup), KM_SLEEP);
1130     /*
1131      * We don't want the full treatment of prlock(pnp) here.
1132      * This file is world-readable and never goes invalid.
1133      * It doesn't matter if we are in the middle of an exec().
1134      */
1135     p = pr_p_lock(pnp);
1136     mutex_exit(&pr_pidlock);
1137     if (p == NULL) {
1138         error = ENOENT;
1139         goto out;
1140     }
1141     ASSERT(p == pnp->pr_common->prc_proc);
1143     if (uiop->uio_offset >= sizeof (prusage_t)) {
1144         prunlock(pnp);
1145         error = 0;
1146         goto out;
1147     }
1149     pup->pr_tstamp = gethrtime();
1151     pup->pr_count = p->p_defunct;
1152     pup->pr_create = p->p_mstart;
1153     pup->pr_term = p->p_mterm;
1155     pup->pr_rtime = p->p_mlreal;
1156     pup->pr_utime = p->p_acct[LMS_USER];
1157     pup->pr_stime = p->p_acct[LMS_SYSTEM];
1158     pup->pr_ttime = p->p_acct[LMS_TRAP];

```

```

1159     pup->pr_tftime   = p->p_acct[LMS_TFAULT];
1160     pup->pr_dftime   = p->p_acct[LMS_DFAULT];
1161     pup->pr_kftime   = p->p_acct[LMS_KFAULT];
1162     pup->pr_ltime    = p->p_acct[LMS_USER_LOCK];
1163     pup->pr_slptime  = p->p_acct[LMS_SLEEP];
1164     pup->pr_wtime    = p->p_acct[LMS_WAIT_CPU];
1165     pup->pr_stoptime = p->p_acct[LMS_STOPPED];

1167     pup->pr_minf    = p->p_ru.minflt;
1168     pup->pr_majf    = p->p_ru.majflt;
1169     pup->pr_nswap   = p->p_ru.nswap;
1170     pup->pr_inblk   = p->p_ru.inblock;
1171     pup->pr_oublk   = p->p_ru.oublock;
1172     pup->pr_msnd    = p->p_ru.msgsnd;
1173     pup->pr_mrcv    = p->p_ru.msgrcv;
1174     pup->pr_sigs    = p->p_ru.nsignals;
1175     pup->pr_vctx    = p->p_ru.nvcs;
1176     pup->pr_ictx    = p->p_ru.nivcs;
1177     pup->pr_sysc    = p->p_ru.sysc;
1178     pup->pr_ioch    = p->p_ru.ioch;

1180     /*
1181     * Add the usage information for each active lwp.
1182     */
1183     if ((t = p->p_tlist) != NULL &&
1184         !(pnp->pr_pcommon->prc_flags & PRC_DESTROY)) {
1185         do {
1186             if (t->t_proc_flag & TP_LWPEXIT)
1187                 continue;
1188             pup->pr_count++;
1189             praddusage(t, pup);
1190         } while ((t = t->t_forw) != p->p_tlist);
1191     }

1193     prunlock(pnp);

1195     prcvtsusage(pup, upup);

1197     error = pr_uioread(upup, sizeof (prusage_t), uiop);
1198 out:
1199     kmem_free(pup, sizeof (*pup));
1200     kmem_free(upup, sizeof (*upup));
1201     return (error);
1202 }

1204 static int
1205 pr_read_lusage(prnode_t *pnp, uio_t *uiop)
1206 {
1207     int nlwp;
1208     prusage_t *pup;
1209     prheader_t *php;
1210     prusage_t *upup;
1211     size_t size;
1212     hrtime_t curtime;
1213     proc_t *p;
1214     kthread_t *t;
1215     lwpdir_t *ldp;
1216     int error;
1217     int i;

1219     ASSERT(pnp->pr_type == PR_LUSAGE);

1221     /*
1222     * We don't want the full treatment of prlock(pnp) here.
1223     * This file is world-readable and never goes invalid.
1224     * It doesn't matter if we are in the middle of an exec().

```

```

1225     /*
1226     p = pr_p_lock(pnp);
1227     mutex_exit(&pr_pidlock);
1228     if (p == NULL)
1229         return (ENOENT);
1230     ASSERT(p == pnp->pr_common->prc_proc);
1231     if ((nlwp = p->p_lwpcnt) == 0) {
1232         prunlock(pnp);
1233         return (ENOENT);
1234     }

1236     size = sizeof (prheader_t) + (nlwp + 1) * LSPAN(prusage_t);
1237     if (uiop->uio_offset >= size) {
1238         prunlock(pnp);
1239         return (0);
1240     }

1242     /* drop p->p_lock to do kmem_alloc(KM_SLEEP) */
1243     mutex_exit(&p->p_lock);
1244     pup = kmem_zalloc(size + sizeof (prusage_t), KM_SLEEP);
1245     mutex_enter(&p->p_lock);
1246     /* p->p_lwpcnt can't change while process is locked */
1247     ASSERT(nlwp == p->p_lwpcnt);

1249     php = (prheader_t *) (pup + 1);
1250     upup = (prusage_t *) (php + 1);

1252     php->pr_nent = nlwp + 1;
1253     php->pr_entsize = LSPAN(prusage_t);

1255     curtime = gethrtime();

1257     /*
1258     * First the summation over defunct lwps.
1259     */
1260     pup->pr_count = p->p_defunct;
1261     pup->pr_tstamp = curtime;
1262     pup->pr_create = p->p_mstart;
1263     pup->pr_term = p->p_mterm;

1265     pup->pr_rtime    = p->p_mlreal;
1266     pup->pr_utime    = p->p_acct[LMS_USER];
1267     pup->pr_stime    = p->p_acct[LMS_SYSTEM];
1268     pup->pr_ttime    = p->p_acct[LMS_TRAP];
1269     pup->pr_tftime   = p->p_acct[LMS_TFAULT];
1270     pup->pr_dftime   = p->p_acct[LMS_DFAULT];
1271     pup->pr_kftime   = p->p_acct[LMS_KFAULT];
1272     pup->pr_ltime    = p->p_acct[LMS_USER_LOCK];
1273     pup->pr_slptime  = p->p_acct[LMS_SLEEP];
1274     pup->pr_wtime    = p->p_acct[LMS_WAIT_CPU];
1275     pup->pr_stoptime = p->p_acct[LMS_STOPPED];

1277     pup->pr_minf    = p->p_ru.minflt;
1278     pup->pr_majf    = p->p_ru.majflt;
1279     pup->pr_nswap   = p->p_ru.nswap;
1280     pup->pr_inblk   = p->p_ru.inblock;
1281     pup->pr_oublk   = p->p_ru.oublock;
1282     pup->pr_msnd    = p->p_ru.msgsnd;
1283     pup->pr_mrcv    = p->p_ru.msgrcv;
1284     pup->pr_sigs    = p->p_ru.nsignals;
1285     pup->pr_vctx    = p->p_ru.nvcs;
1286     pup->pr_ictx    = p->p_ru.nivcs;
1287     pup->pr_sysc    = p->p_ru.sysc;
1288     pup->pr_ioch    = p->p_ru.ioch;

1290     prcvtsusage(pup, upup);

```

```

1292     /*
1293     * Fill one prusage struct for each active lwp.
1294     */
1295     for (ldp = p->p_lwpdir, i = 0; i < p->p_lwpdir_sz; i++, ldp++) {
1296         if (ldp->ld_entry == NULL ||
1297             (t = ldp->ld_entry->le_thread) == NULL)
1298             continue;
1299         ASSERT(!(t->t_proc_flag & TP_LWPEXIT));
1300         ASSERT(nlwp > 0);
1301         --nlwp;
1302         upup = (prusage_t *)((caddr_t)upup + LSPAN(prusage_t));
1303         prgetusage(t, pup);
1304         prcvtusage(pup, upup);
1305     }
1306     ASSERT(nlwp == 0);
1308     prunlock(pnp);
1310     error = pr_uioread(php, size, uiop);
1311     kmem_free(pup, size + sizeof(prusage_t));
1312     return (error);
1313 }
1315 static int
1316 pr_read_pagedata(prnode_t *pnp, uio_t *uiop)
1317 {
1318     proc_t *p;
1319     int error;
1321     ASSERT(pnp->pr_type == PR_PAGEDATA);
1323     if ((error = prlock(pnp, ZNO)) != 0)
1324         return (error);
1326     p = pnp->pr_common->prc_proc;
1327     if ((p->p_flag & SSYS) || p->p_as == &kas) {
1328         prunlock(pnp);
1329         return (0);
1330     }
1332     mutex_exit(&p->p_lock);
1333     error = prpdread(p, pnp->pr_hatid, uiop);
1334     mutex_enter(&p->p_lock);
1336     prunlock(pnp);
1337     return (error);
1338 }
1340 static int
1341 pr_read_opagedata(prnode_t *pnp, uio_t *uiop)
1342 {
1343     proc_t *p;
1344     struct as *as;
1345     int error;
1347     ASSERT(pnp->pr_type == PR_OPAGEDATA);
1349     if ((error = prlock(pnp, ZNO)) != 0)
1350         return (error);
1352     p = pnp->pr_common->prc_proc;
1353     as = p->p_as;
1354     if ((p->p_flag & SSYS) || as == &kas) {
1355         prunlock(pnp);
1356         return (0);

```

```

1357     }
1359     mutex_exit(&p->p_lock);
1360     error = oprpdread(as, pnp->pr_hatid, uiop);
1361     mutex_enter(&p->p_lock);
1363     prunlock(pnp);
1364     return (error);
1365 }
1367 static int
1368 pr_read_watch(prnode_t *pnp, uio_t *uiop)
1369 {
1370     proc_t *p;
1371     int error;
1372     prwatch_t *Bpwp;
1373     size_t size;
1374     prwatch_t *pwp;
1375     int nwarea;
1376     struct watched_area *pwarea;
1378     ASSERT(pnp->pr_type == PR_WATCH);
1380     if ((error = prlock(pnp, ZNO)) != 0)
1381         return (error);
1383     p = pnp->pr_common->prc_proc;
1384     nwarea = avl_numnodes(&p->p_warea);
1385     size = nwarea * sizeof(prwatch_t);
1386     if (uiop->uio_offset >= size) {
1387         prunlock(pnp);
1388         return (0);
1389     }
1391     /* drop p->p_lock to do kmem_alloc(KM_SLEEP) */
1392     mutex_exit(&p->p_lock);
1393     Bpwp = pwp = kmem_zalloc(size, KM_SLEEP);
1394     mutex_enter(&p->p_lock);
1395     /* p->p_nwarea can't change while process is locked */
1396     ASSERT(nwarea == avl_numnodes(&p->p_warea));
1398     /* gather the watched areas */
1399     for (pwarea = avl_first(&p->p_warea); pwarea != NULL;
1400         pwarea = AVL_NEXT(&p->p_warea, pwarea), pwp++) {
1401         pwp->pr_vaddr = (uintptr_t)pwarea->wa_vaddr;
1402         pwp->pr_size = pwarea->wa_eaddr - pwarea->wa_vaddr;
1403         pwp->pr_wflags = (int)pwarea->wa_flags;
1404     }
1406     prunlock(pnp);
1408     error = pr_uioread(Bpwp, size, uiop);
1409     kmem_free(Bpwp, size);
1410     return (error);
1411 }
1413 static int
1414 pr_read_lwpstatus(prnode_t *pnp, uio_t *uiop)
1415 {
1416     lwpstatus_t *sp;
1417     int error;
1419     ASSERT(pnp->pr_type == PR_LWPSTATUS);
1421     /*
1422     * We kmem_alloc() the lwpstatus structure because

```

```

1423     * it is so big it might blow the kernel stack.
1424     */
1425     sp = kmem_alloc(sizeof (*sp), KM_SLEEP);

1427     if ((error = prlock(pnp, ZNO)) != 0)
1428         goto out;

1430     if (uiop->uio_offset >= sizeof (*sp)) {
1431         prunlock(pnp);
1432         goto out;
1433     }

1435     prgetlwpstatus(pnp->pr_common->prc_thread, sp, VTOZONE(PTOV(pnp)));
1436     prunlock(pnp);

1438     error = pr_uioread(sp, sizeof (*sp), uiop);
1439 out:
1440     kmem_free(sp, sizeof (*sp));
1441     return (error);
1442 }

1444 static int
1445 pr_read_lwpsinfo(prnode_t *pnp, uio_t *uiop)
1446 {
1447     lwpsinfo_t lwpsinfo;
1448     proc_t *p;
1449     kthread_t *t;
1450     lwpent_t *lep;

1452     ASSERT(pnp->pr_type == PR_LWPSINFO);

1454     /*
1455     * We don't want the full treatment of prlock(pnp) here.
1456     * This file is world-readable and never goes invalid.
1457     * It doesn't matter if we are in the middle of an exec().
1458     */
1459     p = pr_p_lock(pnp);
1460     mutex_exit(&pr_pidlock);
1461     if (p == NULL)
1462         return (ENOENT);
1463     ASSERT(p == pnp->pr_common->prc_proc);
1464     if (pnp->pr_common->prc_tslot == -1) {
1465         prunlock(pnp);
1466         return (ENOENT);
1467     }

1469     if (uiop->uio_offset >= sizeof (lwpsinfo)) {
1470         prunlock(pnp);
1471         return (0);
1472     }

1474     if ((t = pnp->pr_common->prc_thread) != NULL)
1475         prgetlwpsinfo(t, &lwpsinfo);
1476     else {
1477         lep = p->p_lwpdir[pnp->pr_common->prc_tslot].ld_entry;
1478         bzero(&lwpsinfo, sizeof (lwpsinfo));
1479         lwpsinfo.pr_lwpid = lep->le_lwpid;
1480         lwpsinfo.pr_state = SZOMB;
1481         lwpsinfo.pr_sname = 'Z';
1482         lwpsinfo.pr_start.tv_sec = lep->le_start;
1483         lwpsinfo.pr_bindpro = PBIND_NONE;
1484         lwpsinfo.pr_bindpset = PS_NONE;
1485     }
1486     prunlock(pnp);

1488     return (pr_uioread(&lwpsinfo, sizeof (lwpsinfo), uiop));

```

```

1489 }

1491 static int
1492 pr_read_lwpusage(prnode_t *pnp, uio_t *uiop)
1493 {
1494     prhusage_t *pup;
1495     prusage_t *upup;
1496     proc_t *p;
1497     int error;

1499     ASSERT(pnp->pr_type == PR_LWPUSAGE);

1501     /* allocate now, before locking the process */
1502     pup = kmem_zalloc(sizeof (*pup), KM_SLEEP);
1503     upup = kmem_alloc(sizeof (*upup), KM_SLEEP);

1505     /*
1506     * We don't want the full treatment of prlock(pnp) here.
1507     * This file is world-readable and never goes invalid.
1508     * It doesn't matter if we are in the middle of an exec().
1509     */
1510     p = pr_p_lock(pnp);
1511     mutex_exit(&pr_pidlock);
1512     if (p == NULL) {
1513         error = ENOENT;
1514         goto out;
1515     }
1516     ASSERT(p == pnp->pr_common->prc_proc);
1517     if (pnp->pr_common->prc_thread == NULL) {
1518         prunlock(pnp);
1519         error = ENOENT;
1520         goto out;
1521     }
1522     if (uiop->uio_offset >= sizeof (prusage_t)) {
1523         prunlock(pnp);
1524         error = 0;
1525         goto out;
1526     }

1528     pup->pr_tstamp = gethrtime();
1529     prgetusage(pnp->pr_common->prc_thread, pup);

1531     prunlock(pnp);

1533     prcvtusage(pup, upup);

1535     error = pr_uioread(upup, sizeof (prusage_t), uiop);
1536 out:
1537     kmem_free(pup, sizeof (*pup));
1538     kmem_free(upup, sizeof (*upup));
1539     return (error);
1540 }

1542 /* ARGSUSED */
1543 static int
1544 pr_read_xregs(prnode_t *pnp, uio_t *uiop)
1545 {
1546     #if defined(__sparc)
1547         proc_t *p;
1548         kthread_t *t;
1549         int error;
1550         char *xreg;
1551         size_t size;

1553         ASSERT(pnp->pr_type == PR_XREGS);

```

```

1555     xreg = kmem_zalloc(sizeof (prxregset_t), KM_SLEEP);
1557     if ((error = prlock(pnp, ZNO)) != 0)
1558         goto out;
1560     p = pnp->pr_common->prc_proc;
1561     t = pnp->pr_common->prc_thread;
1563     size = prhasx(p)? prgetprxregsize(p) : 0;
1564     if (uiop->uio_offset >= size) {
1565         prunlock(pnp);
1566         goto out;
1567     }
1569     /* drop p->p_lock while (possibly) touching the stack */
1570     mutex_exit(&p->p_lock);
1571     prgetprxregs(ttolwp(t), xreg);
1572     mutex_enter(&p->p_lock);
1573     prunlock(pnp);
1575     error = pr_uioread(xreg, size, uiop);
1576 out:
1577     kmem_free(xreg, sizeof (prxregset_t));
1578     return (error);
1579 #else
1580     return (0);
1581 #endif
1582 }
1584 static int
1585 pr_read_spymaster(prnode_t *pnp, uio_t *uiop)
1586 {
1587     psinfo_t psinfo;
1588     int error;
1589     klwp_t *lwp;
1591     ASSERT(pnp->pr_type == PR_SPYMASTER);
1593     if ((error = prlock(pnp, ZNO)) != 0)
1594         return (error);
1596     lwp = pnp->pr_common->prc_thread->t_lwp;
1598     if (lwp->lwp_spymaster == NULL) {
1599         prunlock(pnp);
1600         return (0);
1601     }
1603     bcopy(lwp->lwp_spymaster, &psinfo, sizeof (psinfo_t));
1604     prunlock(pnp);
1606     return (pr_uioread(&psinfo, sizeof (psinfo), uiop));
1607 }
1609 static int
1610 pr_read_secflags(prnode_t *pnp, uio_t *uiop)
1611 {
1612     prsecflags_t ret;
1613     int error;
1614     proc_t *p;
1616     ASSERT(pnp->pr_type == PR_SECFLAGS);
1618     if ((error = prlock(pnp, ZNO)) != 0)
1619         return (error);

```

```

1621     p = pnp->pr_common->prc_proc;
1622     prgetsecflags(p, &ret);
1623     prunlock(pnp);
1625     return (pr_uioread(&ret, sizeof (ret), uiop));
1626 }
1628 #endif /* ! codereview */
1629 #if defined(__sparc)
1631 static int
1632 pr_read_gwindows(prnode_t *pnp, uio_t *uiop)
1633 {
1634     proc_t *p;
1635     kthread_t *t;
1636     gwindows_t *gwp;
1637     int error;
1638     size_t size;
1640     ASSERT(pnp->pr_type == PR_GWINDOWS);
1642     gwp = kmem_zalloc(sizeof (gwindows_t), KM_SLEEP);
1644     if ((error = prlock(pnp, ZNO)) != 0)
1645         goto out;
1647     p = pnp->pr_common->prc_proc;
1648     t = pnp->pr_common->prc_thread;
1650     /*
1651     * Drop p->p_lock while touching the stack.
1652     * The P_PR_LOCK flag prevents the lwp from
1653     * disappearing while we do this.
1654     */
1655     mutex_exit(&p->p_lock);
1656     if ((size = prnwindows(ttolwp(t))) != 0)
1657         size = sizeof (gwindows_t) -
1658             (SPARC_MAXREGWINDOW - size) * sizeof (struct rwindow);
1659     if (uiop->uio_offset >= size) {
1660         mutex_enter(&p->p_lock);
1661         prunlock(pnp);
1662         goto out;
1663     }
1664     prgetwindows(ttolwp(t), gwp);
1665     mutex_enter(&p->p_lock);
1666     prunlock(pnp);
1668     error = pr_uioread(gwp, size, uiop);
1669 out:
1670     kmem_free(gwp, sizeof (gwindows_t));
1671     return (error);
1672 }
1674 /* ARGSUSED */
1675 static int
1676 pr_read_asrs(prnode_t *pnp, uio_t *uiop)
1677 {
1678     int error;
1680     ASSERT(pnp->pr_type == PR_ASRS);
1682     /* the asrs file exists only for sparc v9_LP64 processes */
1683     if ((error = prlock(pnp, ZNO)) == 0) {
1684         proc_t *p = pnp->pr_common->prc_proc;
1685         kthread_t *t = pnp->pr_common->prc_thread;
1686         asrset_t asrset;

```

```

1688     if (p->p_model != DATAMODEL_LP64 ||
1689         uiop->uio_offset >= sizeof (asrset_t)) {
1690         prunlock(pnp);
1691         return (0);
1692     }
1693
1694     /*
1695     * Drop p->p_lock while touching the stack.
1696     * The P_PR_LOCK flag prevents the lwp from
1697     * disappearing while we do this.
1698     */
1699     mutex_exit(&p->p_lock);
1700     prgetasregs(ttolwp(t), asrset);
1701     mutex_enter(&p->p_lock);
1702     prunlock(pnp);
1703
1704     error = pr_uioread(&asrset[0], sizeof (asrset_t), uiop);
1705 }
1706
1707 return (error);
1708 }
1709
1710 #endif /* __sparc */
1711
1712 static int
1713 pr_read_pidfile(prnode_t *pnp, uio_t *uiop)
1714 {
1715     ASSERT(pnp->pr_type == PR_PIDDIR);
1716     ASSERT(pnp->pr_pidfile != NULL);
1717
1718     /* use the underlying PR_PIDFILE to read the process */
1719     pnp = VTOP(pnp->pr_pidfile);
1720     ASSERT(pnp->pr_type == PR_PIDFILE);
1721
1722     return (pr_read_pidfile(pnp, uiop));
1723 }
1724
1725 static int
1726 pr_read_pidfile(prnode_t *pnp, uio_t *uiop)
1727 {
1728     int error;
1729
1730     ASSERT(pnp->pr_type == PR_PIDFILE || pnp->pr_type == PR_LWPIDFILE);
1731
1732     if ((error = prlock(pnp, ZNO)) == 0) {
1733         proc_t *p = pnp->pr_common->prc_proc;
1734         struct as *as = p->p_as;
1735
1736         if ((p->p_flag & SSYS) || as == &kas) {
1737             /*
1738             * /proc I/O cannot be done to a system process.
1739             */
1740             error = EIO; /* old /proc semantics */
1741         } else {
1742             /*
1743             * We drop p_lock because we don't want to hold
1744             * it over an I/O operation because that could
1745             * lead to deadlock with the clock thread.
1746             * The process will not disappear and its address
1747             * space will not change because it is marked P_PR_LOCK.
1748             */
1749             mutex_exit(&p->p_lock);
1750             error = prusrrio(p, UIO_READ, uiop, 1);
1751             mutex_enter(&p->p_lock);
1752         }
1753     }

```

```

1753         prunlock(pnp);
1754     }
1755
1756     return (error);
1757 }
1758
1759 #ifdef _SYSCALL32_IMPL
1760
1761 /*
1762 * Array of ILP32 read functions, indexed by /proc file type.
1763 */
1764 static int pr_read_status_32(),
1765           pr_read_lstatus_32(), pr_read_psinfo_32(), pr_read_lpsinfo_32(),
1766           pr_read_map_32(), pr_read_rmap_32(), pr_read_xmap_32(),
1767           pr_read_sigact_32(), pr_read_auxv_32(),
1768           pr_read_usage_32(), pr_read_lusage_32(), pr_read_pagedata_32(),
1769           pr_read_watch_32(), pr_read_lwpstatus_32(), pr_read_lwpsinfo_32(),
1770           pr_read_lwpusage_32(), pr_read_spymaster_32(),
1771 #if defined(__sparc)
1772           pr_read_gwindows_32(),
1773 #endif
1774           pr_read_opagedata_32();
1775
1776 static int (*pr_read_function_32[PR_NFILES])() = {
1777     pr_read_inval, /* /proc */
1778     pr_read_inval, /* /proc/self */
1779     pr_read_pidfile, /* /proc/<pid> (old /proc read()) */
1780     pr_read_as, /* /proc/<pid>/as */
1781     pr_read_inval, /* /proc/<pid>/ctl */
1782     pr_read_status_32, /* /proc/<pid>/status */
1783     pr_read_lstatus_32, /* /proc/<pid>/lstatus */
1784     pr_read_psinfo_32, /* /proc/<pid>/psinfo */
1785     pr_read_lpsinfo_32, /* /proc/<pid>/lpsinfo */
1786     pr_read_map_32, /* /proc/<pid>/map */
1787     pr_read_rmap_32, /* /proc/<pid>/rmap */
1788     pr_read_xmap_32, /* /proc/<pid>/xmap */
1789     pr_read_cred, /* /proc/<pid>/cred */
1790     pr_read_sigact_32, /* /proc/<pid>/sigact */
1791     pr_read_auxv_32, /* /proc/<pid>/auxv */
1792 #if defined(__x86)
1793     pr_read_ldt, /* /proc/<pid>/ldt */
1794 #endif
1795     pr_read_usage_32, /* /proc/<pid>/usage */
1796     pr_read_lusage_32, /* /proc/<pid>/lusage */
1797     pr_read_pagedata_32, /* /proc/<pid>/pagedata */
1798     pr_read_watch_32, /* /proc/<pid>/watch */
1799     pr_read_inval, /* /proc/<pid>/cwd */
1800     pr_read_inval, /* /proc/<pid>/root */
1801     pr_read_inval, /* /proc/<pid>/fd */
1802     pr_read_inval, /* /proc/<pid>/fd/nn */
1803     pr_read_inval, /* /proc/<pid>/object */
1804     pr_read_inval, /* /proc/<pid>/object/xxx */
1805     pr_read_inval, /* /proc/<pid>/lwp */
1806     pr_read_inval, /* /proc/<pid>/lwp/<lwpid> */
1807     pr_read_inval, /* /proc/<pid>/lwp/<lwpid>/lwpctl */
1808     pr_read_lwpstatus_32, /* /proc/<pid>/lwp/<lwpid>/lwpstatus */
1809     pr_read_lwpsinfo_32, /* /proc/<pid>/lwp/<lwpid>/lwpsinfo */
1810     pr_read_lwpusage_32, /* /proc/<pid>/lwp/<lwpid>/lwpusage */
1811     pr_read_xregs, /* /proc/<pid>/lwp/<lwpid>/xregs */
1812     pr_read_inval, /* /proc/<pid>/lwp/<lwpid>/templates */
1813     pr_read_inval, /* /proc/<pid>/lwp/<lwpid>/templates/<id> */
1814     pr_read_spymaster_32, /* /proc/<pid>/lwp/<lwpid>/spymaster */
1815 #if defined(__sparc)
1816     pr_read_gwindows_32, /* /proc/<pid>/lwp/<lwpid>/gwindows */
1817     pr_read_asrs, /* /proc/<pid>/lwp/<lwpid>/asrs */
1818 #endif

```

```

1819     pr_read_priv,      /* /proc/<pid>/priv      */
1820     pr_read_inval,    /* /proc/<pid>/path      */
1821     pr_read_inval,    /* /proc/<pid>/path/xxx  */
1822     pr_read_inval,    /* /proc/<pid>/contracts */
1823     pr_read_inval,    /* /proc/<pid>/contracts/<ctid> */
1824     pr_read_secflags, /* /proc/<pid>/secflags  */
1825 #endif /* ! codereview */
1826     pr_read_pidfile,  /* old process file     */
1827     pr_read_pidfile,  /* old lwp file         */
1828     pr_read_opagedata_32, /* old pagedata file   */
1829 };

1831 static int
1832 pr_read_status_32(prnode_t *pnp, uio_t *uiop)
1833 {
1834     pstatus32_t *sp;
1835     proc_t *p;
1836     int error;

1838     ASSERT(pnp->pr_type == PR_STATUS);

1840     /*
1841     * We kmem_alloc() the pstatus structure because
1842     * it is so big it might blow the kernel stack.
1843     */
1844     sp = kmem_alloc(sizeof (*sp), KM_SLEEP);
1845     if ((error = prlock(pnp, ZNO)) == 0) {
1846         /*
1847         * A 32-bit process cannot get the status of a 64-bit process.
1848         * The fields for the 64-bit quantities are not large enough.
1849         */
1850         p = pnp->pr_common->prc_proc;
1851         if (PROCESS_NOT_32BIT(p)) {
1852             prunlock(pnp);
1853             error = EOVERFLOW;
1854         } else {
1855             prgetstatus32(pnp->pr_common->prc_proc, sp,
1856                 VTOZONE(PTOV(pnp)));
1857             prunlock(pnp);
1858             error = pr_uioread(sp, sizeof (*sp), uiop);
1859         }
1860     }
1861     kmem_free((caddr_t)sp, sizeof (*sp));
1862     return (error);
1863 }

1865 static int
1866 pr_read_lstatus_32(prnode_t *pnp, uio_t *uiop)
1867 {
1868     proc_t *p;
1869     kthread_t *t;
1870     lwpdir_t *ldp;
1871     size_t size;
1872     prheader32_t *php;
1873     lwpstatus32_t *sp;
1874     int error;
1875     int nlwp;
1876     int i;

1878     ASSERT(pnp->pr_type == PR_LSTATUS);

1880     if ((error = prlock(pnp, ZNO)) != 0)
1881         return (error);
1882     p = pnp->pr_common->prc_proc;
1883     /*
1884     * A 32-bit process cannot get the status of a 64-bit process.

```

```

1885     * The fields for the 64-bit quantities are not large enough.
1886     */
1887     if (PROCESS_NOT_32BIT(p)) {
1888         prunlock(pnp);
1889         return (EOVERFLOW);
1890     }
1891     nlwp = p->p_lwpcnt;
1892     size = sizeof (prheader32_t) + nlwp * LSPAN32(lwpstatus32_t);

1894     /* drop p->p_lock to do kmem_alloc(KM_SLEEP) */
1895     mutex_exit(&p->p_lock);
1896     php = kmem_zalloc(size, KM_SLEEP);
1897     mutex_enter(&p->p_lock);
1898     /* p->p_lwpcnt can't change while process is locked */
1899     ASSERT(nlwp == p->p_lwpcnt);

1901     php->pr_nent = nlwp;
1902     php->pr_entsize = LSPAN32(lwpstatus32_t);

1904     sp = (lwpstatus32_t *) (php + 1);
1905     for (ldp = p->p_lwpdir, i = 0; i < p->p_lwpdir_sz; i++, ldp++) {
1906         if (ldp->ld_entry == NULL ||
1907             (t = ldp->ld_entry->le_thread) == NULL)
1908             continue;
1909         prgetlwpstatus32(t, sp, VTOZONE(PTOV(pnp)));
1910         sp = (lwpstatus32_t *) ((caddr_t)sp + LSPAN32(lwpstatus32_t));
1911     }
1912     prunlock(pnp);

1914     error = pr_uioread(php, size, uiop);
1915     kmem_free(php, size);
1916     return (error);
1917 }

1919 static int
1920 pr_read_psinfn_32(prnode_t *pnp, uio_t *uiop)
1921 {
1922     psinfo32_t psinfo;
1923     proc_t *p;
1924     int error = 0;

1926     ASSERT(pnp->pr_type == PR_PSINFO);

1928     /*
1929     * We don't want the full treatment of prlock(pnp) here.
1930     * This file is world-readable and never goes invalid.
1931     * It doesn't matter if we are in the middle of an exec().
1932     */
1933     p = pr_p_lock(pnp);
1934     mutex_exit(&pr_pidlock);
1935     if (p == NULL)
1936         error = ENOENT;
1937     else {
1938         ASSERT(p == pnp->pr_common->prc_proc);
1939         prgetpsinfo32(p, &psinfo);
1940         prunlock(pnp);
1941         error = pr_uioread(&psinfo, sizeof (psinfo), uiop);
1942     }
1943     return (error);
1944 }

1946 static int
1947 pr_read_lpsinfo_32(prnode_t *pnp, uio_t *uiop)
1948 {
1949     proc_t *p;
1950     kthread_t *t;

```



```

1951     lwpdir_t *ldp;
1952     lwpent_t *lep;
1953     size_t size;
1954     prheader32_t *php;
1955     lwpsinfo32_t *sp;
1956     int error;
1957     int nlwp;
1958     int i;

1960     ASSERT(pnp->pr_type == PR_LPSINFO);

1962     /*
1963      * We don't want the full treatment of prlock(pnp) here.
1964      * This file is world-readable and never goes invalid.
1965      * It doesn't matter if we are in the middle of an exec().
1966      */
1967     p = pr_p_lock(pnp);
1968     mutex_exit(&pr_pidlock);
1969     if (p == NULL)
1970         return (ENOENT);
1971     ASSERT(p == pnp->pr_common->prc_proc);
1972     if ((nlwp = p->p_lwpcnt + p->p_zombcnt) == 0) {
1973         prunlock(pnp);
1974         return (ENOENT);
1975     }
1976     size = sizeof (prheader32_t) + nlwp * LSPAN32(lwpsinfo32_t);

1978     /* drop p->p_lock to do kmem_alloc(KM_SLEEP) */
1979     mutex_exit(&p->p_lock);
1980     php = kmem_zalloc(size, KM_SLEEP);
1981     mutex_enter(&p->p_lock);
1982     /* p->p_lwpcnt can't change while process is locked */
1983     ASSERT(nlwp == p->p_lwpcnt + p->p_zombcnt);

1985     php->pr_nent = nlwp;
1986     php->pr_entsize = LSPAN32(lwpsinfo32_t);

1988     sp = (lwpsinfo32_t *) (php + 1);
1989     for (ldp = p->p_lwpdir, i = 0; i < p->p_lwpdir_sz; i++, ldp++) {
1990         if ((lep = ldp->ld_entry) == NULL)
1991             continue;
1992         if ((t = lep->le_thread) != NULL)
1993             prgetlwpsinfo32(t, sp);
1994         else {
1995             bzero(sp, sizeof (*sp));
1996             sp->pr_lwpid = lep->le_lwpid;
1997             sp->pr_state = SZOMB;
1998             sp->pr_sname = 'Z';
1999             sp->pr_start.tv_sec = (time32_t) lep->le_start;
2000         }
2001         sp = (lwpsinfo32_t *) ((caddr_t) sp + LSPAN32(lwpsinfo32_t));
2002     }
2003     prunlock(pnp);

2005     error = pr_uioread(php, size, uiop);
2006     kmem_free(php, size);
2007     return (error);
2008 }

2010 static int
2011 pr_read_map_common_32(prnode_t *pnp, uio_t *uiop, prnodetype_t type)
2012 {
2013     proc_t *p;
2014     struct as *as;
2015     list_t iolhead;
2016     int error;

```

```

2018 readmap32_common:
2019     if ((error = prlock(pnp, ZNO)) != 0)
2020         return (error);

2022     p = pnp->pr_common->prc_proc;
2023     as = p->p_as;

2025     if ((p->p_flag & SSYS) || as == &kas) {
2026         prunlock(pnp);
2027         return (0);
2028     }

2030     if (PROCESS_NOT_32BIT(p)) {
2031         prunlock(pnp);
2032         return (EOVERFLOW);
2033     }

2035     if (!AS_LOCK_TRYENTER(as, RW_WRITER)) {
2036         prunlock(pnp);
2037         delay(1);
2038         goto readmap32_common;
2039     }
2040     mutex_exit(&p->p_lock);

2042     switch (type) {
2043     case PR_XMAP:
2044         error = prgetxmap32(p, &iolhead);
2045         break;
2046     case PR_RMAP:
2047         error = prgetmap32(p, 1, &iolhead);
2048         break;
2049     case PR_MAP:
2050         error = prgetmap32(p, 0, &iolhead);
2051         break;
2052     }
2053     AS_LOCK_EXIT(as);
2054     mutex_enter(&p->p_lock);
2055     prunlock(pnp);

2057     error = pr_iol_uiomove_and_free(&iolhead, uiop, error);

2059     return (error);
2060 }

2062 static int
2063 pr_read_map_32(prnode_t *pnp, uio_t *uiop)
2064 {
2065     ASSERT(pnp->pr_type == PR_MAP);
2066     return (pr_read_map_common_32(pnp, uiop, pnp->pr_type));
2067 }

2069 static int
2070 pr_read_rmap_32(prnode_t *pnp, uio_t *uiop)
2071 {
2072     ASSERT(pnp->pr_type == PR_RMAP);
2073     return (pr_read_map_common_32(pnp, uiop, pnp->pr_type));
2074 }

2076 static int
2077 pr_read_xmap_32(prnode_t *pnp, uio_t *uiop)
2078 {
2079     ASSERT(pnp->pr_type == PR_XMAP);
2080     return (pr_read_map_common_32(pnp, uiop, pnp->pr_type));
2081 }

```

```

2083 static int
2084 pr_read_sigact_32(prnode_t *pnp, uio_t *uiop)
2085 {
2086     int nsig = PROC_IS_BRANDED(curproc)? BROP(curproc)->b_nsig : NSIG;
2087     proc_t *p;
2088     struct sigaction32 *sap;
2089     int sig;
2090     int error;
2091     user_t *up;
2092
2093     ASSERT(pnp->pr_type == PR_SIGACT);
2094
2095     /*
2096      * We kmem_alloc() the sigaction32 array because
2097      * it is so big it might blow the kernel stack.
2098      */
2099     sap = kmem_alloc((nsig-1) * sizeof (struct sigaction32), KM_SLEEP);
2100
2101     if ((error = prlock(pnp, ZNO)) != 0)
2102         goto out;
2103     p = pnp->pr_common->prc_proc;
2104
2105     if (PROCESS_NOT_32BIT(p)) {
2106         prunlock(pnp);
2107         error = EOVERFLOW;
2108         goto out;
2109     }
2110
2111     if (uiop->uio_offset >= (nsig-1) * sizeof (struct sigaction32)) {
2112         prunlock(pnp);
2113         goto out;
2114     }
2115
2116     up = PTOU(p);
2117     for (sig = 1; sig < nsig; sig++)
2118         prgetaction32(p, up, sig, &sap[sig-1]);
2119     prunlock(pnp);
2120
2121     error = pr_uioread(sap, (nsig - 1) * sizeof (struct sigaction32), uiop);
2122 out:
2123     kmem_free(sap, (nsig-1) * sizeof (struct sigaction32));
2124     return (error);
2125 }
2126
2127 static int
2128 pr_read_auxv_32(prnode_t *pnp, uio_t *uiop)
2129 {
2130     auxv32_t auxv[__KERN_NAUXV_IMPL];
2131     proc_t *p;
2132     user_t *up;
2133     int error;
2134     int i;
2135
2136     ASSERT(pnp->pr_type == PR_AUXV);
2137
2138     if ((error = prlock(pnp, ZNO)) != 0)
2139         return (error);
2140     p = pnp->pr_common->prc_proc;
2141
2142     if (PROCESS_NOT_32BIT(p)) {
2143         prunlock(pnp);
2144         return (EOVERFLOW);
2145     }
2146
2147     if (uiop->uio_offset >= sizeof (auxv)) {
2148         prunlock(pnp);

```

```

2149         return (0);
2150     }
2151
2152     up = PTOU(p);
2153     for (i = 0; i < __KERN_NAUXV_IMPL; i++) {
2154         auxv[i].a_type = (int32_t)up->u_auxv[i].a_type;
2155         auxv[i].a_un.a_val = (int32_t)up->u_auxv[i].a_un.a_val;
2156     }
2157     prunlock(pnp);
2158
2159     return (pr_uioread(auxv, sizeof (auxv), uiop));
2160 }
2161
2162 static int
2163 pr_read_usage_32(prnode_t *pnp, uio_t *uiop)
2164 {
2165     prusage_t *pup;
2166     prusage32_t *upup;
2167     proc_t *p;
2168     kthread_t *t;
2169     int error;
2170
2171     ASSERT(pnp->pr_type == PR_USAGE);
2172
2173     /* allocate now, before locking the process */
2174     pup = kmem_zalloc(sizeof (*pup), KM_SLEEP);
2175     upup = kmem_alloc(sizeof (*upup), KM_SLEEP);
2176
2177     /*
2178      * We don't want the full treatment of prlock(pnp) here.
2179      * This file is world-readable and never goes invalid.
2180      * It doesn't matter if we are in the middle of an exec().
2181      */
2182     p = pr_p_lock(pnp);
2183     mutex_exit(&pr_pidlock);
2184     if (p == NULL) {
2185         error = ENOENT;
2186         goto out;
2187     }
2188     ASSERT(p == pnp->pr_common->prc_proc);
2189
2190     if (uiop->uio_offset >= sizeof (prusage32_t)) {
2191         prunlock(pnp);
2192         error = 0;
2193         goto out;
2194     }
2195
2196     pup->pr_tstamp = gethrtime();
2197
2198     pup->pr_count = p->p_defunct;
2199     pup->pr_create = p->p_mstart;
2200     pup->pr_term = p->p_mterm;
2201
2202     pup->pr_rtime = p->p_mlreal;
2203     pup->pr_utime = p->p_acct[LMS_USER];
2204     pup->pr_stime = p->p_acct[LMS_SYSTEM];
2205     pup->pr_ttime = p->p_acct[LMS_TRAP];
2206     pup->pr_tftime = p->p_acct[LMS_TFAULT];
2207     pup->pr_dftime = p->p_acct[LMS_DFAULT];
2208     pup->pr_kftime = p->p_acct[LMS_KFAULT];
2209     pup->pr_ltime = p->p_acct[LMS_USER_LOCK];
2210     pup->pr_slptime = p->p_acct[LMS_SLEEP];
2211     pup->pr_wtime = p->p_acct[LMS_WAIT_CPU];
2212     pup->pr_stoptime = p->p_acct[LMS_STOPPED];
2213
2214     pup->pr_minfl = p->p_ru.minflt;

```

```

2215     pup->pr_majf = p->p_ru.majflt;
2216     pup->pr_nswap = p->p_ru.nswap;
2217     pup->pr_inblk = p->p_ru.inblock;
2218     pup->pr_oublk = p->p_ru.oublock;
2219     pup->pr_msnd = p->p_ru.msgsnd;
2220     pup->pr_mrcv = p->p_ru.msgrcv;
2221     pup->pr_sigs = p->p_ru.nsignals;
2222     pup->pr_vctx = p->p_ru.nvcsw;
2223     pup->pr_ictx = p->p_ru.nivcsw;
2224     pup->pr_sysc = p->p_ru.sysc;
2225     pup->pr_ioch = p->p_ru.ioch;

2227     /*
2228     * Add the usage information for each active lwp.
2229     */
2230     if ((t = p->p_tlist) != NULL &&
2231         !(pnp->pr_pcommon->prc_flags & PRC_DESTROY)) {
2232         do {
2233             if (t->t_proc_flag & TP_LWPEXIT)
2234                 continue;
2235             pup->pr_count++;
2236             praddusage(t, pup);
2237         } while ((t = t->t_forw) != p->p_tlist);
2238     }

2240     prunlock(pnp);

2242     prcvtusage32(pup, upup);

2244     error = pr_uioread(upup, sizeof (prusage32_t), uiop);
2245 out:
2246     kmem_free(pup, sizeof (*pup));
2247     kmem_free(upup, sizeof (*upup));
2248     return (error);
2249 }

2251 static int
2252 pr_read_lusage_32(prnode_t *pnp, uio_t *uiop)
2253 {
2254     int nlwp;
2255     prusage_t *pup;
2256     prheader32_t *php;
2257     prusage32_t *upup;
2258     size_t size;
2259     hrtime_t curtime;
2260     proc_t *p;
2261     kthread_t *t;
2262     lwpdir_t *ldp;
2263     int error;
2264     int i;

2266     ASSERT(pnp->pr_type == PR_LUSAGE);

2268     /*
2269     * We don't want the full treatment of prlock(pnp) here.
2270     * This file is world-readable and never goes invalid.
2271     * It doesn't matter if we are in the middle of an exec().
2272     */
2273     p = pr_p_lock(pnp);
2274     mutex_exit(&pr_pidlock);
2275     if (p == NULL)
2276         return (ENOENT);
2277     ASSERT(p == pnp->pr_common->prc_proc);
2278     if ((nlwp = p->p_lwpcnt) == 0) {
2279         prunlock(pnp);
2280         return (ENOENT);

```

```

2281     }

2283     size = sizeof (prheader32_t) + (nlwp + 1) * LSPAN32(prusage32_t);
2284     if (uiop->uio_offset >= size) {
2285         prunlock(pnp);
2286         return (0);
2287     }

2289     /* drop p->p_lock to do kmem_alloc(KM_SLEEP) */
2290     mutex_exit(&p->p_lock);
2291     pup = kmem_zalloc(size + sizeof (prusage_t), KM_SLEEP);
2292     mutex_enter(&p->p_lock);
2293     /* p->p_lwpcnt can't change while process is locked */
2294     ASSERT(nlwp == p->p_lwpcnt);

2296     php = (prheader32_t *) (pup + 1);
2297     upup = (prusage32_t *) (php + 1);

2299     php->pr_nent = nlwp + 1;
2300     php->pr_entsize = LSPAN32(prusage32_t);

2302     curtime = gethrtime();

2304     /*
2305     * First the summation over defunct lwps.
2306     */
2307     pup->pr_count = p->p_defunct;
2308     pup->pr_tstamp = curtime;
2309     pup->pr_create = p->p_mstart;
2310     pup->pr_term = p->p_mterm;

2312     pup->pr_rtime = p->p_mlreal;
2313     pup->pr_utime = p->p_acct[LMS_USER];
2314     pup->pr_stime = p->p_acct[LMS_SYSTEM];
2315     pup->pr_ttime = p->p_acct[LMS_TRAP];
2316     pup->pr_tftime = p->p_acct[LMS_TFAULT];
2317     pup->pr_dftime = p->p_acct[LMS_DFAULT];
2318     pup->pr_kftime = p->p_acct[LMS_KFAULT];
2319     pup->pr_ltime = p->p_acct[LMS_USER_LOCK];
2320     pup->pr_slptime = p->p_acct[LMS_SLEEP];
2321     pup->pr_wtime = p->p_acct[LMS_WAIT_CPU];
2322     pup->pr_stoptime = p->p_acct[LMS_STOPPED];

2324     pup->pr_minf = p->p_ru.minflt;
2325     pup->pr_majf = p->p_ru.majflt;
2326     pup->pr_nswap = p->p_ru.nswap;
2327     pup->pr_inblk = p->p_ru.inblock;
2328     pup->pr_oublk = p->p_ru.oublock;
2329     pup->pr_msnd = p->p_ru.msgsnd;
2330     pup->pr_mrcv = p->p_ru.msgrcv;
2331     pup->pr_sigs = p->p_ru.nsignals;
2332     pup->pr_vctx = p->p_ru.nvcsw;
2333     pup->pr_ictx = p->p_ru.nivcsw;
2334     pup->pr_sysc = p->p_ru.sysc;
2335     pup->pr_ioch = p->p_ru.ioch;

2337     prcvtusage32(pup, upup);

2339     /*
2340     * Fill one prusage struct for each active lwp.
2341     */
2342     for (ldp = p->p_lwpdir, i = 0; i < p->p_lwpdir_sz; i++, ldp++) {
2343         if (ldp->ld_entry == NULL ||
2344             (t = ldp->ld_entry->le_thread) == NULL)
2345             continue;
2346         ASSERT(!(t->t_proc_flag & TP_LWPEXIT));

```

```

2347     ASSERT(nlwp > 0);
2348     --nlwp;
2349     upup = (prusage32_t *)
2350           ((caddr_t)upup + LSPAN32(prusage32_t));
2351     prgetusage(t, pup);
2352     prcvtusage32(pup, upup);
2353 }
2354 ASSERT(nlwp == 0);

2356 prunlock(pnp);

2358 error = pr_uioread(php, size, uiop);
2359 kmem_free(pup, size + sizeof(prusage_t));
2360 return (error);
2361 }

2363 static int
2364 pr_read_pagedata_32(prnode_t *pnp, uio_t *uiop)
2365 {
2366     proc_t *p;
2367     int error;

2369     ASSERT(pnp->pr_type == PR_PAGEDATA);

2371     if ((error = prlock(pnp, ZNO)) != 0)
2372         return (error);

2374     p = pnp->pr_common->prc_proc;
2375     if ((p->p_flag & SSYS) || p->p_as == &kas) {
2376         prunlock(pnp);
2377         return (0);
2378     }

2380     if (PROCESS_NOT_32BIT(p)) {
2381         prunlock(pnp);
2382         return (EOVERFLOW);
2383     }

2385     mutex_exit(&p->p_lock);
2386     error = prpdread32(p, pnp->pr_hatid, uiop);
2387     mutex_enter(&p->p_lock);

2389     prunlock(pnp);
2390     return (error);
2391 }

2393 static int
2394 pr_read_opagedata_32(prnode_t *pnp, uio_t *uiop)
2395 {
2396     proc_t *p;
2397     struct as *as;
2398     int error;

2400     ASSERT(pnp->pr_type == PR_OPAGEDATA);

2402     if ((error = prlock(pnp, ZNO)) != 0)
2403         return (error);

2405     p = pnp->pr_common->prc_proc;
2406     as = p->p_as;

2408     if ((p->p_flag & SSYS) || as == &kas) {
2409         prunlock(pnp);
2410         return (0);
2411     }

```

```

2413     if (PROCESS_NOT_32BIT(p)) {
2414         prunlock(pnp);
2415         return (EOVERFLOW);
2416     }

2418     mutex_exit(&p->p_lock);
2419     error = oprpdread32(as, pnp->pr_hatid, uiop);
2420     mutex_enter(&p->p_lock);

2422     prunlock(pnp);
2423     return (error);
2424 }

2426 static int
2427 pr_read_watch_32(prnode_t *pnp, uio_t *uiop)
2428 {
2429     proc_t *p;
2430     int error;
2431     prwatch32_t *Bpwp;
2432     size_t size;
2433     prwatch32_t *pwp;
2434     int nwarea;
2435     struct watched_area *pwarea;

2437     ASSERT(pnp->pr_type == PR_WATCH);

2439     if ((error = prlock(pnp, ZNO)) != 0)
2440         return (error);

2442     p = pnp->pr_common->prc_proc;
2443     if (PROCESS_NOT_32BIT(p)) {
2444         prunlock(pnp);
2445         return (EOVERFLOW);
2446     }
2447     nwarea = avl_numnodes(&p->p_warea);
2448     size = nwarea * sizeof(prwatch32_t);
2449     if (uiop->uio_offset >= size) {
2450         prunlock(pnp);
2451         return (0);
2452     }

2454     /* drop p->p_lock to do kmem_alloc(KM_SLEEP) */
2455     mutex_exit(&p->p_lock);
2456     Bpwp = pwp = kmem_zalloc(size, KM_SLEEP);
2457     mutex_enter(&p->p_lock);
2458     /* p->p_nwarea can't change while process is locked */
2459     ASSERT(nwarea == avl_numnodes(&p->p_warea));

2461     /* gather the watched areas */
2462     for (pwarea = avl_first(&p->p_warea); pwarea != NULL;
2463          pwarea = AVL_NEXT(&p->p_warea, pwarea), pwp++) {
2464         pwp->pr_vaddr = (caddr32_t)(uintptr_t)pwarea->wa_vaddr;
2465         pwp->pr_size = (size32_t)(pwarea->wa_eaddr - pwarea->wa_vaddr);
2466         pwp->pr_wflags = (int)pwarea->wa_flags;
2467     }

2469     prunlock(pnp);

2471     error = pr_uioread(Bpwp, size, uiop);
2472     kmem_free(Bpwp, size);
2473     return (error);
2474 }

2476 static int
2477 pr_read_lwpstatus_32(prnode_t *pnp, uio_t *uiop)
2478 {

```

```

2479     lwpstatus32_t *sp;
2480     proc_t *p;
2481     int error;

2483     ASSERT(pnp->pr_type == PR_LWPSTATUS);

2485     /*
2486      * We kmem_alloc() the lwpstatus structure because
2487      * it is so big it might blow the kernel stack.
2488      */
2489     sp = kmem_alloc(sizeof (*sp), KM_SLEEP);

2491     if ((error = prlock(pnp, ZNO)) != 0)
2492         goto out;

2494     /*
2495      * A 32-bit process cannot get the status of a 64-bit process.
2496      * The fields for the 64-bit quantities are not large enough.
2497      */
2498     p = pnp->pr_common->prc_proc;
2499     if (PROCESS_NOT_32BIT(p)) {
2500         prunlock(pnp);
2501         error = EOVERFLOW;
2502         goto out;
2503     }

2505     if (uiop->uio_offset >= sizeof (*sp)) {
2506         prunlock(pnp);
2507         goto out;
2508     }

2510     prgetlwpstatus32(pnp->pr_common->prc_thread, sp, VTOZONE(PTOV(pnp)));
2511     prunlock(pnp);

2513     error = pr_uioread(sp, sizeof (*sp), uiop);
2514 out:
2515     kmem_free(sp, sizeof (*sp));
2516     return (error);
2517 }

2519 static int
2520 pr_read_lwpsinfo_32(prnode_t *pnp, uio_t *uiop)
2521 {
2522     lwpsinfo32_t lwpsinfo;
2523     proc_t *p;
2524     kthread_t *t;
2525     lwpent_t *lep;

2527     ASSERT(pnp->pr_type == PR_LWPSINFO);

2529     /*
2530      * We don't want the full treatment of prlock(pnp) here.
2531      * This file is world-readable and never goes invalid.
2532      * It doesn't matter if we are in the middle of an exec().
2533      */
2534     p = pr_p_lock(pnp);
2535     mutex_exit(&pr_pidlock);
2536     if (p == NULL)
2537         return (ENOENT);
2538     ASSERT(p == pnp->pr_common->prc_proc);
2539     if (pnp->pr_common->prc_tslot == -1) {
2540         prunlock(pnp);
2541         return (ENOENT);
2542     }

2544     if (uiop->uio_offset >= sizeof (lwpsinfo)) {

```

```

2545         prunlock(pnp);
2546         return (0);
2547     }

2549     if ((t = pnp->pr_common->prc_thread) != NULL)
2550         prgetlwpsinfo32(t, &lwpsinfo);
2551     else {
2552         lep = p->p_lwpdir[pnp->pr_common->prc_tslot].ld_entry;
2553         bzero(&lwpsinfo, sizeof (lwpsinfo));
2554         lwpsinfo.pr_lwpid = lep->le_lwpid;
2555         lwpsinfo.pr_state = SZOMB;
2556         lwpsinfo.pr_sname = 'Z';
2557         lwpsinfo.pr_start.tv_sec = (time32_t)lep->le_start;
2558     }
2559     prunlock(pnp);

2561     return (pr_uioread(&lwpsinfo, sizeof (lwpsinfo), uiop));
2562 }

2564 static int
2565 pr_read_lwpusage_32(prnode_t *pnp, uio_t *uiop)
2566 {
2567     prhusage_t *pup;
2568     prusage32_t *upup;
2569     proc_t *p;
2570     int error;

2572     ASSERT(pnp->pr_type == PR_LWPUSAGE);

2574     /* allocate now, before locking the process */
2575     pup = kmem_zalloc(sizeof (*pup), KM_SLEEP);
2576     upup = kmem_alloc(sizeof (*upup), KM_SLEEP);

2578     /*
2579      * We don't want the full treatment of prlock(pnp) here.
2580      * This file is world-readable and never goes invalid.
2581      * It doesn't matter if we are in the middle of an exec().
2582      */
2583     p = pr_p_lock(pnp);
2584     mutex_exit(&pr_pidlock);
2585     if (p == NULL) {
2586         error = ENOENT;
2587         goto out;
2588     }
2589     ASSERT(p == pnp->pr_common->prc_proc);
2590     if (pnp->pr_common->prc_thread == NULL) {
2591         prunlock(pnp);
2592         error = ENOENT;
2593         goto out;
2594     }
2595     if (uiop->uio_offset >= sizeof (prusage32_t)) {
2596         prunlock(pnp);
2597         error = 0;
2598         goto out;
2599     }

2601     pup->pr_tstamp = gethrtime();
2602     prgetusage(pnp->pr_common->prc_thread, pup);

2604     prunlock(pnp);

2606     prcvtusage32(pup, upup);

2608     error = pr_uioread(upup, sizeof (prusage32_t), uiop);
2609 out:
2610     kmem_free(pup, sizeof (*pup));

```

```

2611     kmem_free(upup, sizeof (*upup));
2612     return (error);
2613 }

2615 static int
2616 pr_read_spymaster_32(prnode_t *pnp, uio_t *uiop)
2617 {
2618     psinfo32_t psinfo;
2619     int error;
2620     klpw_t *lwp;

2622     ASSERT(pnp->pr_type == PR_SPYMASTER);

2624     if ((error = prlock(pnp, ZNO)) != 0)
2625         return (error);

2627     lwp = pnp->pr_common->prc_thread->t_lwp;

2629     if (lwp->lwp_spymaster == NULL) {
2630         prunlock(pnp);
2631         return (0);
2632     }

2634     psinfo_kto32(lwp->lwp_spymaster, &psinfo);
2635     prunlock(pnp);

2637     return (pr_uioread(&psinfo, sizeof (psinfo), uiop));
2638 }

2640 #if defined(__sparc)
2641 static int
2642 pr_read_gwindows_32(prnode_t *pnp, uio_t *uiop)
2643 {
2644     proc_t *p;
2645     kthread_t *t;
2646     gwindows32_t *gwp;
2647     int error;
2648     size_t size;

2650     ASSERT(pnp->pr_type == PR_GWINDOWS);

2652     gwp = kmem_zalloc(sizeof (gwindows32_t), KM_SLEEP);

2654     if ((error = prlock(pnp, ZNO)) != 0)
2655         goto out;

2657     p = pnp->pr_common->prc_proc;
2658     t = pnp->pr_common->prc_thread;

2660     if (PROCESS_NOT_32BIT(p)) {
2661         prunlock(pnp);
2662         error = EOVERFLOW;
2663         goto out;
2664     }

2666     /*
2667     * Drop p->p_lock while touching the stack.
2668     * The P_PR_LOCK flag prevents the lwp from
2669     * disappearing while we do this.
2670     */
2671     mutex_exit(&p->p_lock);
2672     if ((size = prnwindows(ttolwp(t)) != 0)
2673         size = sizeof (gwindows32_t) -
2674             (SPARC_MAXREGWINDOW - size) * sizeof (struct rwindow32);
2675     if (uiop->uio_offset >= size) {
2676         mutex_enter(&p->p_lock);

```

```

2677         prunlock(pnp);
2678         goto out;
2679     }
2680     prgetwindows32(ttolwp(t), gwp);
2681     mutex_enter(&p->p_lock);
2682     prunlock(pnp);

2684     error = pr_uioread(gwp, size, uiop);
2685 out:
2686     kmem_free(gwp, sizeof (gwindows32_t));
2687     return (error);
2688 }
2689 #endif /* __sparc */

2691 #endif /* _SYSCALL32_IMPL */

2693 /* ARGSUSED */
2694 static int
2695 prread(vnode_t *vp, uio_t *uiop, int ioflag, cred_t *cr, caller_context_t *ct)
2696 {
2697     prnode_t *pnp = VTOP(vp);

2699     ASSERT(pnp->pr_type < PR_NFILES);

2701 #ifdef _SYSCALL32_IMPL
2702     /*
2703     * What is read from the /proc files depends on the data
2704     * model of the caller. An LP64 process will see LP64
2705     * data. An ILP32 process will see ILP32 data.
2706     */
2707     if (curproc->p_model == DATAMODEL_LP64)
2708         return (pr_read_function[pnp->pr_type](pnp, uiop));
2709     else
2710         return (pr_read_function_32[pnp->pr_type](pnp, uiop));
2711 #else
2712     return (pr_read_function[pnp->pr_type](pnp, uiop));
2713 #endif
2714 }

2716 /* ARGSUSED */
2717 static int
2718 prwrite(vnode_t *vp, uio_t *uiop, int ioflag, cred_t *cr, caller_context_t *ct)
2719 {
2720     prnode_t *pnp = VTOP(vp);
2721     int old = 0;
2722     int error;
2723     ssize_t resid;

2725     ASSERT(pnp->pr_type < PR_NFILES);

2727     /*
2728     * Only a handful of /proc files are writable, enumerate them here.
2729     */
2730     switch (pnp->pr_type) {
2731     case PR_PIDDIR: /* directory write(): visceral revulsion. */
2732         ASSERT(pnp->pr_pidfile != NULL);
2733         /* use the underlying PR_PIDFILE to write the process */
2734         vp = pnp->pr_pidfile;
2735         pnp = VTOP(vp);
2736         ASSERT(pnp->pr_type == PR_PIDFILE);
2737         /* FALLTHROUGH */
2738     case PR_PIDFILE:
2739     case PR_LWPIDFILE:
2740         old = 1;
2741         /* FALLTHROUGH */
2742     case PR_AS:

```

```

2743     if ((error = prlock(pnp, ZNO)) == 0) {
2744         proc_t *p = pnp->pr_common->prc_proc;
2745         struct as *as = p->p_as;

2747         if ((p->p_flag & SSYS) || as == &kas) {
2748             /*
2749              * /proc I/O cannot be done to a system process.
2750              */
2751             error = EIO;
2752 #ifdef _SYSCALL32_IMPL
2753         } else if (curproc->p_model == DATAMODEL_ILP32 &&
2754                 PROCESS_NOT_32BIT(p)) {
2755             error = EOVERFLOW;
2756 #endif
2757         } else {
2758             /*
2759              * See comments above (pr_read_pidfile)
2760              * about this locking dance.
2761              */
2762             mutex_exit(&p->p_lock);
2763             error = prusrrio(p, UIO_WRITE, uiop, old);
2764             mutex_enter(&p->p_lock);
2765         }
2766         prunlock(pnp);
2767     }
2768     return (error);

2770     case PR_CTL:
2771     case PR_LWPCTL:
2772         resid = uiop->uio_resid;
2773         /*
2774          * Perform the action on the control file
2775          * by passing curthreads credentials
2776          * and not target process's credentials.
2777          */
2778 #ifdef _SYSCALL32_IMPL
2779         if (curproc->p_model == DATAMODEL_ILP32)
2780             error = prwritectl32(vp, uiop, CRED());
2781         else
2782             error = prwritectl(vp, uiop, CRED());
2783 #else
2784         error = prwritectl(vp, uiop, CRED());
2785 #endif
2786         /*
2787          * This hack makes sure that the EINTR is passed
2788          * all the way back to the caller's write() call.
2789          */
2790         if (error == EINTR)
2791             uiop->uio_resid = resid;
2792         return (error);

2794     default:
2795         return ((vp->v_type == VDIR)? EISDIR : EBADF);
2796     }
2797     /* NOTREACHED */
2798 }

2800 static int
2801 prgetattr(vnode_t *vp, vattr_t *vap, int flags, cred_t *cr,
2802 caller_context_t *ct)
2803 {
2804     prnode_t *pnp = VTOP(vp);
2805     prnodetype_t type = pnp->pr_type;
2806     prcommon_t *pcp;
2807     proc_t *p;
2808     struct as *as;

```

```

2809     int error;
2810     vnode_t *rvp;
2811     timestruc_t now;
2812     extern uint_t nproc;
2813     int ngroups;
2814     int nsig;

2816     /*
2817      * This ugly bit of code allows us to keep both versions of this
2818      * function from the same source.
2819      */
2820 #ifdef _LP64
2821     int iam32bit = (curproc->p_model == DATAMODEL_ILP32);
2822 #define PR_OBJSIZE(obj32, obj64) \
2823     (iam32bit ? sizeof (obj32) : sizeof (obj64))
2824 #define PR_OBJSPAN(obj32, obj64) \
2825     (iam32bit ? LSPAN32(obj32) : LSPAN(obj64))
2826 #else
2827 #define PR_OBJSIZE(obj32, obj64) \
2828     (sizeof (obj64))
2829 #define PR_OBJSPAN(obj32, obj64) \
2830     (LSPAN(obj64))
2831 #endif

2833     /*
2834      * Return all the attributes. Should be refined
2835      * so that it returns only those asked for.
2836      * Most of this is complete fakery anyway.
2837      */

2839     /*
2840      * For files in the /proc/<pid>/object directory,
2841      * return the attributes of the underlying object.
2842      * For files in the /proc/<pid>/fd directory,
2843      * return the attributes of the underlying file, but
2844      * make it look inaccessible if it is not a regular file.
2845      * Make directories look like symlinks.
2846      */
2847     switch (type) {
2848     case PR_CURDIR:
2849     case PR_ROOTDIR:
2850         if (!(flags & ATTR_REAL))
2851             break;
2852         /* restrict full knowledge of the attributes to owner or root */
2853         if ((error = praccess(vp, 0, 0, cr, ct)) != 0)
2854             return (error);
2855         /* FALLTHROUGH */
2856     case PR_OBJECT:
2857     case PR_FD:
2858         rvp = pnp->pr_realvp;
2859         error = VOP_GETATTR(rvp, vap, flags, cr, ct);
2860         if (error)
2861             return (error);
2862         if (type == PR_FD) {
2863             if (rvp->v_type != VREG && rvp->v_type != VDIR)
2864                 vap->va_mode = 0;
2865             else
2866                 vap->va_mode &= pnp->pr_mode;
2867         }
2868         if (type == PR_OBJECT)
2869             vap->va_mode &= 07555;
2870         if (rvp->v_type == VDIR && !(flags & ATTR_REAL)) {
2871             vap->va_type = VLNK;
2872             vap->va_size = 0;
2873             vap->va_nlink = 1;
2874         }

```

```

2875         return (0);
2876 default:
2877     break;
2878 }

2880 bzero(vap, sizeof (*vap));
2881 /*
2882  * Large Files: Internally proc now uses VPROC to indicate
2883  * a proc file. Since we have been returning VREG through
2884  * VOP_GETATTR() until now, we continue to do this so as
2885  * not to break apps depending on this return value.
2886  */
2887 vap->va_type = (vp->v_type == VPROC) ? VREG : vp->v_type;
2888 vap->va_mode = pnp->pr_mode;
2889 vap->va_fsid = vp->v_vfsp->vfs_dev;
2890 vap->va_blksize = DEV_BSIZE;
2891 vap->va_rdev = 0;
2892 vap->va_seq = 0;

2894 if (type == PR_PROCDIR) {
2895     vap->va_uid = 0;
2896     vap->va_gid = 0;
2897     vap->va_nlink = nproc + 2;
2898     vap->va_nodeid = (ino64_t)PRROOTINO;
2899     gethrestime(&now);
2900     vap->va_atime = vap->va_mtime = vap->va_ctime = now;
2901     vap->va_size = (v.v_proc + 2) * PRSDSIZE;
2902     vap->va_nblocks = btod(vap->va_size);
2903     return (0);
2904 }

2906 /*
2907  * /proc/<pid>/self is a symbolic link, and has no prcommon member
2908  */
2909 if (type == PR_SELF) {
2910     vap->va_uid = crgetruid(CRED());
2911     vap->va_gid = crgetrgid(CRED());
2912     vap->va_nodeid = (ino64_t)PR_SELF;
2913     gethrestime(&now);
2914     vap->va_atime = vap->va_mtime = vap->va_ctime = now;
2915     vap->va_nlink = 1;
2916     vap->va_type = VLNK;
2917     vap->va_size = 0;
2918     return (0);
2919 }

2921 p = pr_p_lock(pnp);
2922 mutex_exit(&pr_pidlock);
2923 if (p == NULL)
2924     return (ENOENT);
2925 pcp = pnp->pr_common;

2927 mutex_enter(&p->p_crlock);
2928 vap->va_uid = crgetruid(p->p_cred);
2929 vap->va_gid = crgetrgid(p->p_cred);
2930 mutex_exit(&p->p_crlock);

2932 vap->va_nlink = 1;
2933 vap->va_nodeid = pnp->pr_ino? pnp->pr_ino :
2934     pmkino(pcp->prc_tslot, pcp->prc_slot, pnp->pr_type);
2935 if ((pcp->prc_flags & PRC_LWP) && pcp->prc_tslot != -1) {
2936     vap->va_atime.tv_sec = vap->va_mtime.tv_sec =
2937     vap->va_ctime.tv_sec =
2938     p->p_lwkdir[pcp->prc_tslot].ld_entry->le_start;
2939     vap->va_atime.tv_nsec = vap->va_mtime.tv_nsec =
2940     vap->va_ctime.tv_nsec = 0;

```

```

2941     } else {
2942         user_t *up = PTOU(p);
2943         vap->va_atime.tv_sec = vap->va_mtime.tv_sec =
2944         vap->va_ctime.tv_sec = up->u_start.tv_sec;
2945         vap->va_atime.tv_nsec = vap->va_mtime.tv_nsec =
2946         vap->va_ctime.tv_nsec = up->u_start.tv_nsec;
2947     }

2949 switch (type) {
2950 case PR_PIDDIR:
2951     /* va_nlink: count 'lwp', 'object' and 'fd' directory links */
2952     vap->va_nlink = 5;
2953     vap->va_size = sizeof (piddir);
2954     break;
2955 case PR_OBJECTDIR:
2956     if ((p->p_flag & SSYS) || (as = p->p_as) == &kas)
2957         vap->va_size = 2 * PRSDSIZE;
2958     else {
2959         mutex_exit(&p->p_lock);
2960         AS_LOCK_ENTER(as, RW_WRITER);
2961         if (as->a_updatedir)
2962             rebuild_objdir(as);
2963         vap->va_size = (as->a_sizedir + 2) * PRSDSIZE;
2964         AS_LOCK_EXIT(as);
2965         mutex_enter(&p->p_lock);
2966     }
2967     vap->va_nlink = 2;
2968     break;
2969 case PR_PATHDIR:
2970     if ((p->p_flag & SSYS) || (as = p->p_as) == &kas)
2971         vap->va_size = (P_FINFO(p)->fi_nfiles + 4) * PRSDSIZE;
2972     else {
2973         mutex_exit(&p->p_lock);
2974         AS_LOCK_ENTER(as, RW_WRITER);
2975         if (as->a_updatedir)
2976             rebuild_objdir(as);
2977         vap->va_size = (as->a_sizedir + 4 +
2978             P_FINFO(p)->fi_nfiles) * PRSDSIZE;
2979         AS_LOCK_EXIT(as);
2980         mutex_enter(&p->p_lock);
2981     }
2982     vap->va_nlink = 2;
2983     break;
2984 case PR_PATH:
2985 case PR_CURDIR:
2986 case PR_ROOTDIR:
2987 case PR_CT:
2988     vap->va_type = VLNK;
2989     vap->va_size = 0;
2990     break;
2991 case PR_FDDIR:
2992     vap->va_nlink = 2;
2993     vap->va_size = (P_FINFO(p)->fi_nfiles + 2) * PRSDSIZE;
2994     break;
2995 case PR_LWPDIR:
2996     /*
2997     * va_nlink: count each lwp as a directory link.
2998     * va_size: size of p_lwkdir + 2
2999     */
3000     vap->va_nlink = p->p_lwpcnt + p->p_zombcnt + 2;
3001     vap->va_size = (p->p_lwkdir_sz + 2) * PRSDSIZE;
3002     break;
3003 case PR_LWPIDDIR:
3004     p->va_nlink = 2;
3005     vap->va_size = sizeof (lwpiddir);
3006     break;

```



```

3007     case PR_CTDIR:
3008         vap->va_nlink = 2;
3009         vap->va_size = (avl_numnodes(&p->p_ct_held) + 2) * PRSDSIZE;
3010         break;
3011     case PR_TMPLDIR:
3012         vap->va_nlink = 2;
3013         vap->va_size = (ct_ntypes + 2) * PRSDSIZE;
3014         break;
3015     case PR_AS:
3016     case PR_PIDFILE:
3017     case PR_LWPIDFILE:
3018         if ((p->p_flag & SSYS) || (as = p->p_as) == &kas)
3019             vap->va_size = 0;
3020         else
3021             vap->va_size = as->a_resvsize;
3022         break;
3023     case PR_STATUS:
3024         vap->va_size = PR_OBJSIZE(pstatus32_t, pstatus_t);
3025         break;
3026     case PR_LSTATUS:
3027         vap->va_size = PR_OBJSIZE(prheader32_t, prheader_t) +
3028             p->p_lwpcnt * PR_OBJSPAN(lwpstatus32_t, lwpstatus_t);
3029         break;
3030     case PR_PSINFO:
3031         vap->va_size = PR_OBJSIZE(psinfo32_t, psinfo_t);
3032         break;
3033     case PR_LPSINFO:
3034         vap->va_size = PR_OBJSIZE(prheader32_t, prheader_t) +
3035             (p->p_lwpcnt + p->p_zombcnt) *
3036             PR_OBJSPAN(lwpsinfo32_t, lwpsinfo_t);
3037         break;
3038     case PR_MAP:
3039     case PR_RMAP:
3040     case PR_XMAP:
3041         if ((p->p_flag & SSYS) || (as = p->p_as) == &kas)
3042             vap->va_size = 0;
3043         else {
3044             mutex_exit(&p->p_lock);
3045             AS_LOCK_ENTER(as, RW_WRITER);
3046             if (type == PR_MAP)
3047                 vap->va_mtime = as->a_updatetime;
3048             if (type == PR_XMAP)
3049                 vap->va_size = prnsegs(as, 0) *
3050                     PR_OBJSIZE(prxmap32_t, prxmap_t);
3051             else
3052                 vap->va_size = prnsegs(as, type == PR_RMAP) *
3053                     PR_OBJSIZE(prmap32_t, prmap_t);
3054             AS_LOCK_EXIT(as);
3055             mutex_enter(&p->p_lock);
3056         }
3057         break;
3058     case PR_CRED:
3059         mutex_enter(&p->p_crlock);
3060         vap->va_size = sizeof(prcred_t);
3061         ngroups = crgetngroups(p->p_cred);
3062         if (ngroups > 1)
3063             vap->va_size += (ngroups - 1) * sizeof(gid_t);
3064         mutex_exit(&p->p_crlock);
3065         break;
3066     case PR_PRIV:
3067         vap->va_size = prgetprivsize();
3068         break;
3069     case PR_SECFLAGS:
3070         vap->va_size = sizeof(prseclags_t);
3071         break;
3072 #endif /* ! codereview */

```

```

3073     case PR_SIGACT:
3074         nsig = PROC_IS_BRANDED(curproc)? BROP(curproc)->b_nsig : NSIG;
3075         vap->va_size = (nsig-1) *
3076             PR_OBJSIZE(struct sigaction32, struct sigaction);
3077         break;
3078     case PR_AUXV:
3079         vap->va_size = __KERN_NAUXV_IMPL * PR_OBJSIZE(auxv32_t, auxv_t);
3080         break;
3081 #if defined(__x86)
3082     case PR_LDT:
3083         mutex_exit(&p->p_lock);
3084         mutex_enter(&p->p_ldtlock);
3085         vap->va_size = prnlldt(p) * sizeof(struct ssd);
3086         mutex_exit(&p->p_ldtlock);
3087         mutex_enter(&p->p_lock);
3088         break;
3089 #endif
3090     case PR_USAGE:
3091         vap->va_size = PR_OBJSIZE(prusage32_t, prusage_t);
3092         break;
3093     case PR_LUSAGE:
3094         vap->va_size = PR_OBJSIZE(prheader32_t, prheader_t) +
3095             (p->p_lwpcnt + 1) * PR_OBJSPAN(prusage32_t, prusage_t);
3096         break;
3097     case PR_PAGEDATA:
3098         if ((p->p_flag & SSYS) || (as = p->p_as) == &kas)
3099             vap->va_size = 0;
3100         else {
3101             /*
3102              * We can drop p->p_lock before grabbing the
3103              * address space lock because p->p_as will not
3104              * change while the process is marked P_PR_LOCK.
3105              */
3106             mutex_exit(&p->p_lock);
3107             AS_LOCK_ENTER(as, RW_WRITER);
3108 #ifdef _LP64
3109             vap->va_size = iam32bit?
3110                 prpdsz32(as) : prpdsz(as);
3111 #else
3112             vap->va_size = prpdsz(as);
3113 #endif
3114             AS_LOCK_EXIT(as);
3115             mutex_enter(&p->p_lock);
3116         }
3117         break;
3118     case PR_OPAGEDATA:
3119         if ((p->p_flag & SSYS) || (as = p->p_as) == &kas)
3120             vap->va_size = 0;
3121         else {
3122             mutex_exit(&p->p_lock);
3123             AS_LOCK_ENTER(as, RW_WRITER);
3124 #ifdef _LP64
3125             vap->va_size = iam32bit?
3126                 oprpdsz32(as) : oprpdsz(as);
3127 #else
3128             vap->va_size = oprpdsz(as);
3129 #endif
3130             AS_LOCK_EXIT(as);
3131             mutex_enter(&p->p_lock);
3132         }
3133         break;
3134     case PR_WATCH:
3135         vap->va_size = avl_numnodes(&p->p_warea) *
3136             PR_OBJSIZE(prwatch32_t, prwatch_t);
3137         break;
3138     case PR_LWPSTATUS:

```

```

3139         vap->va_size = PR_OBJSIZE(lwpstatus32_t, lwpstatus_t);
3140         break;
3141     case PR_LWPSINFO:
3142         vap->va_size = PR_OBJSIZE(lwpsinfo32_t, lwpsinfo_t);
3143         break;
3144     case PR_LWPUSAGE:
3145         vap->va_size = PR_OBJSIZE(prusage32_t, prusage_t);
3146         break;
3147     case PR_XREGS:
3148         if (prhasx(p))
3149             vap->va_size = prgetprxregsize(p);
3150         else
3151             vap->va_size = 0;
3152         break;
3153     case PR_SPYMASTER:
3154         if (pnp->pr_common->prc_thread->t_lwp->lwp_spymaster != NULL) {
3155             vap->va_size = PR_OBJSIZE(psinfo32_t, psinfo_t);
3156         } else {
3157             vap->va_size = 0;
3158         }
3159         break;
3160 #if defined(__sparc)
3161     case PR_GWINDOWS:
3162     {
3163         kthread_t *t;
3164         int n;
3165
3166         /*
3167          * If there is no lwp then just make the size zero.
3168          * This can happen if the lwp exits between the VOP_LOOKUP()
3169          * of the /proc/<pid>/lwp/<lwpid>/gwindows file and the
3170          * VOP_GETATTR() of the resulting vnode.
3171          */
3172         if ((t = pcp->prc_thread) == NULL) {
3173             vap->va_size = 0;
3174             break;
3175         }
3176         /*
3177          * Drop p->p_lock while touching the stack.
3178          * The P_PR_LOCK flag prevents the lwp from
3179          * disappearing while we do this.
3180          */
3181         mutex_exit(&p->p_lock);
3182         if ((n = prnwindows(ttolwp(t))) == 0)
3183             vap->va_size = 0;
3184         else
3185             vap->va_size = PR_OBJSIZE(gwindows32_t, gwindows_t) -
3186                 (SPARC_MAXREGWINDOW - n) *
3187                 PR_OBJSIZE(struct rwindow32, struct rwindow);
3188         mutex_enter(&p->p_lock);
3189         break;
3190     }
3191     case PR_ASRS:
3192 #ifdef _LP64
3193         if (p->p_model == DATAMODEL_LP64)
3194             vap->va_size = sizeof(asrset_t);
3195         else
3196 #endif
3197             vap->va_size = 0;
3198         break;
3199 #endif
3200     case PR_CTL:
3201     case PR_LWPCTL:
3202     default:
3203         vap->va_size = 0;
3204         break;

```

```

3205     }
3206
3207     prunlock(pnp);
3208     vap->va_nblocks = (fsblkcnt64_t)btod(vap->va_size);
3209     return (0);
3210 }
3211
3212 static int
3213 praccess(vnode_t *vp, int mode, int flags, cred_t *cr, caller_context_t *ct)
3214 {
3215     prnode_t *pnp = VTOP(vp);
3216     prnodetype_t type = pnp->pr_type;
3217     int vmode;
3218     vtype_t vtype;
3219     proc_t *p;
3220     int error = 0;
3221     vnode_t *rvp;
3222     vnode_t *xvp;
3223
3224     if ((mode & VWRITE) && vn_is_readonly(vp))
3225         return (EROFS);
3226
3227     switch (type) {
3228     case PR_PROCDIR:
3229         break;
3230
3231     case PR_OBJECT:
3232     case PR_FD:
3233         /*
3234          * Disallow write access to the underlying objects.
3235          * Disallow access to underlying non-regular-file fds.
3236          * Disallow access to fds with other than existing open modes.
3237          */
3238         rvp = pnp->pr_realvp;
3239         vtype = rvp->v_type;
3240         vmode = pnp->pr_mode;
3241         if ((type == PR_OBJECT && (mode & VWRITE)) ||
3242             (type == PR_FD && vtype != VREG && vtype != VDIR) ||
3243             (type == PR_FD && (vmode & mode) != mode &&
3244              secpolicy_proc_access(cr) != 0))
3245             return (EACCES);
3246         return (VOP_ACCESS(rvp, mode, flags, cr, ct));
3247
3248     case PR_PSINFO:
3249         /* these files can be read by anyone */
3250     case PR_LPSINFO:
3251     case PR_LWPSINFO:
3252     case PR_LWPDIR:
3253     case PR_LWPIDDIR:
3254     case PR_USAGE:
3255     case PR_LUSAGE:
3256     case PR_LWPUSAGE:
3257         p = pr_p_lock(pnp);
3258         mutex_exit(&pr_pidlock);
3259         if (p == NULL)
3260             return (ENOENT);
3261         prunlock(pnp);
3262         break;
3263
3264     default:
3265         /*
3266          * Except for the world-readable files above,
3267          * only /proc/pid exists if the process is a zombie.
3268          */
3269         if ((error = prlock(pnp,
3270             (type == PR_PIDDIR)? ZYES : ZNO)) != 0)
3271             return (error);

```

```

3271     p = pnp->pr_common->prc_proc;
3272     if (p != curproc)
3273         error = priv_proc_cred_perm(cr, p, NULL, mode);
3275
3276     if (error != 0 || p == curproc || (p->p_flag & SSYS) ||
3277         p->p_as == &kas || (xvp = p->p_exec) == NULL) {
3278         prunlock(pnp);
3279     } else {
3280         /*
3281          * Determine if the process's executable is readable.
3282          * We have to drop p->p_lock before the secpolicy
3283          * and VOP operation.
3284          */
3285         VN_HOLD(xvp);
3286         prunlock(pnp);
3287         if (secpolicy_proc_access(cr) != 0)
3288             error = VOP_ACCESS(xvp, VREAD, 0, cr, ct);
3289         VN_RELE(xvp);
3290     }
3291     if (error)
3292         return (error);
3293     break;
3295 }
3296
3297 if (type == PR_CURDIR || type == PR_ROOTDIR) {
3298     /*
3299      * Final access check on the underlying directory vnode.
3300      */
3301     return (VOP_ACCESS(pnp->pr_realvp, mode, flags, cr, ct));
3302 }
3303
3304 /*
3305  * Visceral revulsion: For compatibility with old /proc,
3306  * allow the /proc/<pid> directory to be opened for writing.
3307  */
3308 vmode = pnp->pr_mode;
3309 if (type == PR_PIDDIR)
3310     vmode |= VWRITE;
3311 if ((vmode & mode) != mode)
3312     error = secpolicy_proc_access(cr);
3313 return (error);
3314 }
3315
3316 /*
3317  * Array of lookup functions, indexed by /proc file type.
3318  */
3319 static vnode_t *pr_lookup_notdir(), *pr_lookup_procdir(), *pr_lookup_pidir(),
3320 *pr_lookup_objctdir(), *pr_lookup_lwpdir(), *pr_lookup_lwpidir(),
3321 *pr_lookup_fddir(), *pr_lookup_pathdir(), *pr_lookup_tmpldir(),
3322 *pr_lookup_ctdir();
3323
3324 static vnode_t *(*pr_lookup_function[PR_NFILES])() = {
3325     pr_lookup_procdir, /* /proc */
3326     pr_lookup_notdir, /* /proc/self */
3327     pr_lookup_pidir, /* /proc/<pid> */
3328     pr_lookup_notdir, /* /proc/<pid>/as */
3329     pr_lookup_notdir, /* /proc/<pid>/ctl */
3330     pr_lookup_notdir, /* /proc/<pid>/status */
3331     pr_lookup_notdir, /* /proc/<pid>/lstatus */
3332     pr_lookup_notdir, /* /proc/<pid>/psinfo */
3333     pr_lookup_notdir, /* /proc/<pid>/lpsinfo */
3334     pr_lookup_notdir, /* /proc/<pid>/map */
3335     pr_lookup_notdir, /* /proc/<pid>/rmap */
3336     pr_lookup_notdir, /* /proc/<pid>/xmap */
3337     pr_lookup_notdir, /* /proc/<pid>/cred */
3338     pr_lookup_notdir, /* /proc/<pid>/sigact */

```

```

3339     pr_lookup_notdir, /* /proc/<pid>/auxv */
3340     #if defined(__x86)
3341     pr_lookup_notdir, /* /proc/<pid>/ldt */
3342     #endif
3343     pr_lookup_notdir, /* /proc/<pid>/usage */
3344     pr_lookup_notdir, /* /proc/<pid>/lusage */
3345     pr_lookup_notdir, /* /proc/<pid>/pagedata */
3346     pr_lookup_notdir, /* /proc/<pid>/watch */
3347     pr_lookup_notdir, /* /proc/<pid>/cwd */
3348     pr_lookup_notdir, /* /proc/<pid>/root */
3349     pr_lookup_fddir, /* /proc/<pid>/fd */
3350     pr_lookup_notdir, /* /proc/<pid>/fd/n */
3351     pr_lookup_objctdir, /* /proc/<pid>/object */
3352     pr_lookup_notdir, /* /proc/<pid>/object/xxx */
3353     pr_lookup_lwpdir, /* /proc/<pid>/lwp */
3354     pr_lookup_lwpidir, /* /proc/<pid>/lwp/<lwpid> */
3355     pr_lookup_notdir, /* /proc/<pid>/lwp/<lwpid>/lwpctl */
3356     pr_lookup_notdir, /* /proc/<pid>/lwp/<lwpid>/lwpstatus */
3357     pr_lookup_notdir, /* /proc/<pid>/lwp/<lwpid>/lwpsinfo */
3358     pr_lookup_notdir, /* /proc/<pid>/lwp/<lwpid>/lwpsusage */
3359     pr_lookup_notdir, /* /proc/<pid>/lwp/<lwpid>/xregs */
3360     pr_lookup_tmpldir, /* /proc/<pid>/lwp/<lwpid>/templates */
3361     pr_lookup_notdir, /* /proc/<pid>/lwp/<lwpid>/templates/<id> */
3362     #if defined(__sparc)
3363     pr_lookup_notdir, /* /proc/<pid>/lwp/<lwpid>/gwindows */
3364     pr_lookup_notdir, /* /proc/<pid>/lwp/<lwpid>/asrs */
3365     #endif
3366     pr_lookup_notdir, /* /proc/<pid>/priv */
3367     pr_lookup_pathdir, /* /proc/<pid>/path */
3368     pr_lookup_notdir, /* /proc/<pid>/path/xxx */
3369     pr_lookup_ctdir, /* /proc/<pid>/contracts */
3370     pr_lookup_notdir, /* /proc/<pid>/contracts/<ctid> */
3371     pr_lookup_notdir, /* /proc/<pid>/secflags */
3372     #endif /* ! codereview */
3373     pr_lookup_notdir, /* old process file */
3374     pr_lookup_notdir, /* old lwp file */
3375     pr_lookup_notdir, /* old pagedata file */
3376 };
3377
3378 static int
3379 prlookup(vnode_t *dp, char *comp, vnode_t **vpp, pathname_t *pathp,
3380 int flags, vnode_t *rdir, cred_t *cr, caller_context_t *ct,
3381 int *direntflags, pathname_t *realpnp)
3382 {
3383     prnode_t *pnp = VTOP(dp);
3384     prnodetype_t type = pnp->pr_type;
3385     int error;
3386
3387     ASSERT(dp->v_type == VDIR);
3388     ASSERT(type < PR_NFILES);
3389
3390     if (type != PR_PROCDIR && strcmp(comp, "..") == 0) {
3391         VN_HOLD(pnp->pr_parent);
3392         *vpp = pnp->pr_parent;
3393         return (0);
3394     }
3395
3396     if (*comp == '\0' ||
3397         strcmp(comp, ".") == 0 || strcmp(comp, "..") == 0) {
3398         VN_HOLD(dp);
3399         *vpp = dp;
3400         return (0);
3401     }
3402
3403     switch (type) {

```

```

3403     case PR_CURDIR:
3404     case PR_ROOTDIR:
3405         /* restrict lookup permission to owner or root */
3406         if ((error = praccess(dp, VEXEC, 0, cr, ct)) != 0)
3407             return (error);
3408         /* FALLTHROUGH */
3409     case PR_FD:
3410         dp = pnp->pr_realvp;
3411         return (VOP_LOOKUP(dp, comp, vpp, pathp, flags, rdir, cr, ct,
3412             direntflags, realpnp));
3413     default:
3414         break;
3415     }

3417     if ((type == PR_OBJECTDIR || type == PR_FDDIR || type == PR_PATHDIR) &&
3418         (error = praccess(dp, VEXEC, 0, cr, ct)) != 0)
3419         return (error);

3421     /* XXX - Do we need to pass ct, direntflags, or realpnp? */
3422     *vpp = (pr_lookup_function[type](dp, comp));

3424     return ((*vpp == NULL) ? ENOENT : 0);
3425 }

3427 /* ARGSUSED */
3428 static int
3429 prcreate(vnode_t *dp, char *comp, vattr_t *vap, vexec_t excl,
3430     int mode, vnode_t **vpp, cred_t *cr, int flag, caller_context_t *ct,
3431     vsecattr_t *vsecp)
3432 {
3433     int error;

3435     if ((error = prlookup(dp, comp, vpp, NULL, 0, NULL, cr,
3436         ct, NULL, NULL)) != 0) {
3437         if (error == ENOENT) /* can't O_CREAT nonexistent files */
3438             error = EACCES; /* unwriteable directories */
3439     } else {
3440         if (excl == EXCL) /* O_EXCL */
3441             error = EEXIST;
3442         else if (vap->va_mask & AT_SIZE) { /* O_TRUNC */
3443             vnode_t *vp = *vpp;
3444             uint_t mask;

3446             if (vp->v_type == VDIR)
3447                 error = EISDIR;
3448             else if (vp->v_type != VPROC ||
3449                 VTOP(vp)->pr_type != PR_FD)
3450                 error = EACCES;
3451             else { /* /proc/<pid>/fd/<n> */
3452                 vp = VTOP(vp)->pr_realvp;
3453                 mask = vap->va_mask;
3454                 vap->va_mask = AT_SIZE;
3455                 error = VOP_SETATTR(vp, vap, 0, cr, ct);
3456                 vap->va_mask = mask;
3457             }
3458         }
3459         if (error) {
3460             VN_RELE(*vpp);
3461             *vpp = NULL;
3462         }
3463     }
3464     return (error);
3465 }

3467 /* ARGSUSED */
3468 static vnode_t *

```

```

3469 pr_lookup_notdir(vnode_t *dp, char *comp)
3470 {
3471     return (NULL);
3472 }

3474 /*
3475  * Find or construct a process vnode for the given pid.
3476  */
3477 static vnode_t *
3478 pr_lookup_procdir(vnode_t *dp, char *comp)
3479 {
3480     pid_t pid;
3481     prnode_t *pnp;
3482     prcommon_t *pcp;
3483     vnode_t *vp;
3484     proc_t *p;
3485     int c;

3487     ASSERT(VTOP(dp)->pr_type == PR_PROCDIR);

3489     if (strcmp(comp, "self") == 0) {
3490         pnp = prgetnode(dp, PR_SELF);
3491         return (PTOV(pnp));
3492     } else {
3493         pid = 0;
3494         while ((c = *comp++) != '\0') {
3495             if (c < '0' || c > '9')
3496                 return (NULL);
3497             pid = 10*pid + c - '0';
3498             if (pid > maxpid)
3499                 return (NULL);
3500         }
3501     }

3503     pnp = prgetnode(dp, PR_PIDDIR);

3505     mutex_enter(&pidlock);
3506     if ((p = prfind(pid)) == NULL || p->p_stat == SIDL) {
3507         mutex_exit(&pidlock);
3508         prfreenode(pnp);
3509         return (NULL);
3510     }
3511     ASSERT(p->p_stat != 0);

3513     /* NOTE: we're holding pidlock across the policy call. */
3514     if (secpolicy_basic_procinfo(CRED(), p, curproc) != 0) {
3515         mutex_exit(&pidlock);
3516         prfreenode(pnp);
3517         return (NULL);
3518     }

3520     mutex_enter(&p->p_lock);
3521     mutex_exit(&pidlock);

3523     /*
3524      * If a process vnode already exists and it is not invalid
3525      * and it was created by the current process and it belongs
3526      * to the same /proc mount point as our parent vnode, then
3527      * just use it and discard the newly-allocated prnode.
3528      */
3529     for (vp = p->p_trace; vp != NULL; vp = VTOP(vp)->pr_next) {
3530         if (!(VTOP(VTOP(vp)->pr_pidfile)->pr_flags & PR_INVALID) &&
3531             VTOP(vp)->pr_owner == curproc &&
3532             vp->v_vfsp == dp->v_vfsp) {
3533             ASSERT(!(VTOP(vp)->pr_flags & PR_INVALID));
3534             VN_HOLD(vp);

```

```

3535         prfreenode(pnp);
3536         mutex_exit(&p->p_lock);
3537         return (vp);
3538     }
3539 }
3540 pnp->pr_owner = curproc;

3542 /*
3543  * prgetnode() initialized most of the prnode.
3544  * Finish the job.
3545  */
3546 pcp = pnp->pr_common; /* the newly-allocated prcommon struct */
3547 if ((vp = p->p_trace) != NULL) {
3548     /* discard the new prcommon and use the existing prcommon */
3549     prfreecommon(pcp);
3550     pcp = VTOP(vp)->pr_common;
3551     mutex_enter(&pcp->prc_mutex);
3552     ASSERT(pcp->prc_refcnt > 0);
3553     pcp->prc_refcnt++;
3554     mutex_exit(&pcp->prc_mutex);
3555     pnp->pr_common = pcp;
3556 } else {
3557     /* initialize the new prcommon struct */
3558     if ((p->p_flag & SSYS) || p->p_as == &kas)
3559         pcp->prc_flags |= PRC_SYS;
3560     if (p->p_stat == SZOMB)
3561         pcp->prc_flags |= PRC_DESTROY;
3562     pcp->prc_proc = p;
3563     pcp->prc_datamodel = p->p_model;
3564     pcp->prc_pid = p->p_pid;
3565     pcp->prc_slot = p->p_slot;
3566 }
3567 pnp->pr_pcommon = pcp;
3568 pnp->pr_parent = dp;
3569 VN_HOLD(dp);
3570 /*
3571  * Link in the old, invalid directory vnode so we
3572  * can later determine the last close of the file.
3573  */
3574 pnp->pr_next = p->p_trace;
3575 p->p_trace = dp = PTOV(pnp);

3577 /*
3578  * Kludge for old /proc: initialize the PR_PIDFILE as well.
3579  */
3580 vp = pnp->pr_pidfile;
3581 pnp = VTOP(vp);
3582 pnp->pr_ino = ptoi(pcp->prc_pid);
3583 pnp->pr_common = pcp;
3584 pnp->pr_pcommon = pcp;
3585 pnp->pr_parent = dp;
3586 pnp->pr_next = p->p_plist;
3587 p->p_plist = vp;

3589     mutex_exit(&p->p_lock);
3590     return (dp);
3591 }

3593 static vnode_t *
3594 pr_lookup_piddir(vnode_t *dp, char *comp)
3595 {
3596     prnode_t *dpnp = VTOP(dp);
3597     vnode_t *vp;
3598     prnode_t *pnp;
3599     proc_t *p;
3600     user_t *up;

```

```

3601     prdirent_t *dirp;
3602     int i;
3603     enum prnodetype type;

3605     ASSERT(dpnp->pr_type == PR_PIDDIR);

3607     for (i = 0; i < NPIDDIRFILES; i++) {
3608         /* Skip "." and ".." */
3609         dirp = &spiddir[i+2];
3610         if (strcmp(comp, dirp->d_name) == 0)
3611             break;
3612     }

3614     if (i >= NPIDDIRFILES)
3615         return (NULL);

3617     type = (int)dirp->d_ino;
3618     pnp = prgetnode(dp, type);

3620     p = pr_p_lock(dpnp);
3621     mutex_exit(&pr_pidlock);
3622     if (p == NULL) {
3623         prfreenode(pnp);
3624         return (NULL);
3625     }
3626     if (dpnp->pr_pcommon->prc_flags & PRC_DESTROY) {
3627         switch (type) {
3628             case PR_PSINFO:
3629             case PR_USAGE:
3630                 break;
3631             default:
3632                 prunlock(dpnp);
3633                 prfreenode(pnp);
3634                 return (NULL);
3635         }
3636     }

3638     switch (type) {
3639     case PR_CURDIR:
3640     case PR_ROOTDIR:
3641         up = PTOU(p);
3642         vp = (type == PR_CURDIR)? up->u_cdir :
3643             (up->u_rdir? up->u_rdir : rootdir);

3645         if (vp == NULL) { /* can't happen? */
3646             prunlock(dpnp);
3647             prfreenode(pnp);
3648             return (NULL);
3649         }
3650         /*
3651          * Fill in the prnode so future references will
3652          * be able to find the underlying object's vnode.
3653          */
3654         VN_HOLD(vp);
3655         pnp->pr_realvp = vp;
3656         break;
3657     default:
3658         break;
3659     }

3661     mutex_enter(&dpnp->pr_mutex);

3663     if ((vp = dpnp->pr_files[i]) != NULL &&
3664         !(VTOP(vp)->pr_flags & PR_INVALID)) {
3665         VN_HOLD(vp);
3666         mutex_exit(&dpnp->pr_mutex);

```

```

3667         prunlock(dpnp);
3668         prfreenode(pnp);
3669         return (vp);
3670     }

3672     /*
3673     * prgetnode() initialized most of the prnode.
3674     * Finish the job.
3675     */
3676     pnp->pr_common = dpnp->pr_common;
3677     pnp->pr_pcommon = dpnp->pr_pcommon;
3678     pnp->pr_parent = dp;
3679     VN_HOLD(dp);
3680     pnp->pr_index = i;

3682     dpnp->pr_files[i] = vp = PTOV(pnp);

3684     /*
3685     * Link new vnode into list of all /proc vnodes for the process.
3686     */
3687     if (vp->v_type == VPROC) {
3688         pnp->pr_next = p->p_plist;
3689         p->p_plist = vp;
3690     }
3691     mutex_exit(&dpnp->pr_mutex);
3692     prunlock(dpnp);
3693     return (vp);
3694 }

3696 static vnode_t *
3697 pr_lookup_objctdir(vnode_t *dp, char *comp)
3698 {
3699     prnode_t *dpnp = VTOP(dp);
3700     prnode_t *pnp;
3701     proc_t *p;
3702     struct seg *seg;
3703     struct as *as;
3704     vnode_t *vp;
3705     vattr_t vattr;

3707     ASSERT(dpnp->pr_type == PR_OBJECTDIR);

3709     pnp = prgetnode(dp, PR_OBJECT);

3711     if (prlock(dpnp, ZNO) != 0) {
3712         prfreenode(pnp);
3713         return (NULL);
3714     }
3715     p = dpnp->pr_common->prc_proc;
3716     if ((p->p_flag & SSYS) || (as = p->p_as) == &kas) {
3717         prunlock(dpnp);
3718         prfreenode(pnp);
3719         return (NULL);
3720     }

3722     /*
3723     * We drop p_lock before grabbing the address space lock
3724     * in order to avoid a deadlock with the clock thread.
3725     * The process will not disappear and its address space
3726     * will not change because it is marked P_PR_LOCK.
3727     */
3728     mutex_exit(&p->p_lock);
3729     AS_LOCK_ENTER(as, RW_READER);
3730     if ((seg = AS_SEGFIRST(as)) == NULL) {
3731         vp = NULL;
3732         goto out;

```

```

3733     }
3734     if (strcmp(comp, "a.out") == 0) {
3735         vp = p->p_exec;
3736         goto out;
3737     }
3738     do {
3739         /*
3740         * Manufacture a filename for the "object" directory.
3741         */
3742         vattr.va_mask = AT_FSID|AT_NODEID;
3743         if (seg->s_ops == &segvn_ops &&
3744             SEGOP_GETVP(seg, seg->s_base, &vp) == 0 &&
3745             vp != NULL && vp->v_type == VREG &&
3746             VOP_GETATTR(vp, &vattr, 0, CRED(), NULL) == 0) {
3747             char name[64];

3749                 if (vp == p->p_exec) /* "a.out" */
3750                     continue;
3751                 pr_object_name(name, vp, &vattr);
3752                 if (strcmp(name, comp) == 0)
3753                     goto out;
3754             }
3755         } while ((seg = AS_SEGNEXT(as, seg)) != NULL);

3757     vp = NULL;
3758 out:
3759     if (vp != NULL) {
3760         VN_HOLD(vp);
3761     }
3762     AS_LOCK_EXIT(as);
3763     mutex_enter(&p->p_lock);
3764     prunlock(dpnp);

3766     if (vp == NULL)
3767         prfreenode(pnp);
3768     else {
3769         /*
3770         * Fill in the prnode so future references will
3771         * be able to find the underlying object's vnode.
3772         * Don't link this prnode into the list of all
3773         * prnodes for the process; this is a one-use node.
3774         * Its use is entirely to catch and fail opens for writing.
3775         */
3776         pnp->pr_realvp = vp;
3777         vp = PTOV(pnp);
3778     }

3780     return (vp);
3781 }

3783 /*
3784 * Find or construct an lwp vnode for the given lwpid.
3785 */
3786 static vnode_t *
3787 pr_lookup_lwpdir(vnode_t *dp, char *comp)
3788 {
3789     id_t tid; /* same type as t->t_tid */
3790     int want_agent;
3791     prnode_t *dpnp = VTOP(dp);
3792     prnode_t *pnp;
3793     prcommon_t *pcp;
3794     vnode_t *vp;
3795     proc_t *p;
3796     kthread_t *t;
3797     lwpdir_t *ldp;
3798     lwpent_t *lep;

```

```

3799     int tslot;
3800     int c;

3802     ASSERT(dnp->pr_type == PR_LWPDIR);

3804     tid = 0;
3805     if (strcmp(comp, "agent") == 0)
3806         want_agent = 1;
3807     else {
3808         want_agent = 0;
3809         while ((c = *comp++) != '\0') {
3810             id_t otid;

3812                 if (c < '0' || c > '9')
3813                     return (NULL);
3814                 otid = tid;
3815                 tid = 10*tid + c - '0';
3816                 if (tid/10 != otid) /* integer overflow */
3817                     return (NULL);
3818             }
3819         }

3821     pnp = prgetnode(dp, PR_LWPIDDIR);

3823     p = pr_p_lock(dnp);
3824     mutex_exit(&pr_pidlock);
3825     if (p == NULL) {
3826         prfreenode(pnp);
3827         return (NULL);
3828     }

3830     if (want_agent) {
3831         if ((t = p->p_agenttp) == NULL)
3832             lep = NULL;
3833         else {
3834             tid = t->t_tid;
3835             tslot = t->t_dslot;
3836             lep = p->p_lwpdir[tslot].ld_entry;
3837         }
3838     } else {
3839         if ((ldp = lwp_hash_lookup(p, tid)) == NULL)
3840             lep = NULL;
3841         else {
3842             tslot = (int)(ldp - p->p_lwpdir);
3843             lep = ldp->ld_entry;
3844         }
3845     }

3847     if (lep == NULL) {
3848         prunlock(dnp);
3849         prfreenode(pnp);
3850         return (NULL);
3851     }

3853     /*
3854     * If an lwp vnode already exists and it is not invalid
3855     * and it was created by the current process and it belongs
3856     * to the same /proc mount point as our parent vnode, then
3857     * just use it and discard the newly-allocated prnode.
3858     */
3859     for (vp = lep->le_trace; vp != NULL; vp = VTOP(vp)->pr_next) {
3860         if (!(VTOP(vp)->pr_flags & PR_INVAL) &&
3861             VTOP(vp)->pr_owner == curproc &&
3862             vp->v_vfsp == dp->v_vfsp) {
3863             VN_HOLD(vp);
3864             prunlock(dnp);

```

```

3865         prfreenode(pnp);
3866         return (vp);
3867     }
3868 }
3869 pnp->pr_owner = curproc;

3871     /*
3872     * prgetnode() initialized most of the prnode.
3873     * Finish the job.
3874     */
3875     pcp = pnp->pr_common; /* the newly-allocated prcommon struct */
3876     if ((vp = lep->le_trace) != NULL) {
3877         /* discard the new prcommon and use the existing prcommon */
3878         prfreecommon(pcp);
3879         pcp = VTOP(vp)->pr_common;
3880         mutex_enter(&pcp->prc_mutex);
3881         ASSERT(pcp->prc_refcnt > 0);
3882         pcp->prc_refcnt++;
3883         mutex_exit(&pcp->prc_mutex);
3884         pnp->pr_common = pcp;
3885     } else {
3886         /* initialize the new prcommon struct */
3887         pcp->prc_flags |= PRC_LWP;
3888         if ((p->p_flag & SSYS) || p->p_as == &kas)
3889             pcp->prc_flags |= PRC_SYS;
3890         if ((t = lep->le_thread) == NULL)
3891             pcp->prc_flags |= PRC_DESTROY;
3892         pcp->prc_proc = p;
3893         pcp->prc_datamodel = dnp->pr_pcommon->prc_datamodel;
3894         pcp->prc_pid = p->p_pid;
3895         pcp->prc_slot = p->p_slot;
3896         pcp->prc_thread = t;
3897         pcp->prc_tid = tid;
3898         pcp->prc_tslot = tslot;
3899     }

3900     pnp->pr_pcommon = dnp->pr_pcommon;
3901     pnp->pr_parent = dp;
3902     VN_HOLD(dp);
3903     /*
3904     * Link in the old, invalid directory vnode so we
3905     * can later determine the last close of the file.
3906     */
3907     pnp->pr_next = lep->le_trace;
3908     lep->le_trace = vp = PTOV(pnp);
3909     prunlock(dnp);
3910     return (vp);
3911 }

3913 static vnode_t *
3914 pr_lookup_lwpiddir(vnode_t *dp, char *comp)
3915 {
3916     prnode_t *dnp = VTOP(dp);
3917     vnode_t *vp;
3918     prnode_t *pnp;
3919     proc_t *p;
3920     prdirent_t *dirp;
3921     int i;
3922     enum prnodetype type;

3924     ASSERT(dnp->pr_type == PR_LWPIDDIR);

3926     for (i = 0; i < NLWPIDDIRFILES; i++) {
3927         /* Skip "." and ".." */
3928         dirp = &lwpiddir[i+2];
3929         if (strcmp(comp, dirp->d_name) == 0)
3930             break;

```

```

3931     }
3933     if (i >= NLWPIDDIRFILES)
3934         return (NULL);
3936     type = (int)dirp->d_ino;
3937     pnp = prgetnode(dp, type);
3939     p = pr_p_lock(dnp);
3940     mutex_exit(&pr_pidlock);
3941     if (p == NULL) {
3942         prfreenode(pnp);
3943         return (NULL);
3944     }
3945     if (dnp->pr_common->prc_flags & PRC_DESTROY) {
3946         /*
3947          * Only the lwpsinfo file is present for zombie lwps.
3948          * Nothing is present if the lwp has been reaped.
3949          */
3950         if (dnp->pr_common->prc_tslot == -1 ||
3951             type != PR_LWPSINFO) {
3952             prunlock(dnp);
3953             prfreenode(pnp);
3954             return (NULL);
3955         }
3956     }
3958 #if defined(__sparc)
3959     /* the asrs file exists only for sparc v9_LP64 processes */
3960     if (type == PR_ASRS && p->p_model != DATAMODEL_LP64) {
3961         prunlock(dnp);
3962         prfreenode(pnp);
3963         return (NULL);
3964     }
3965 #endif
3967     mutex_enter(&dnp->pr_mutex);
3969     if ((vp = dnp->pr_files[i]) != NULL &&
3970         !(VTOP(vp)->pr_flags & PR_INVALID)) {
3971         VN_HOLD(vp);
3972         mutex_exit(&dnp->pr_mutex);
3973         prunlock(dnp);
3974         prfreenode(pnp);
3975         return (vp);
3976     }
3978     /*
3979     * prgetnode() initialized most of the prnode.
3980     * Finish the job.
3981     */
3982     pnp->pr_common = dnp->pr_common;
3983     pnp->pr_pcommon = dnp->pr_pcommon;
3984     pnp->pr_parent = dp;
3985     VN_HOLD(dp);
3986     pnp->pr_index = i;
3988     dnp->pr_files[i] = vp = PTOV(pnp);
3990     /*
3991     * Link new vnode into list of all /proc vnodes for the process.
3992     */
3993     if (vp->v_type == VPROC) {
3994         pnp->pr_next = p->p_plist;
3995         p->p_plist = vp;
3996     }

```

```

3997         mutex_exit(&dnp->pr_mutex);
3998         prunlock(dnp);
3999         return (vp);
4000     }
4002     /*
4003     * Lookup one of the process's open files.
4004     */
4005     static vnode_t *
4006     pr_lookup_fddir(vnode_t *dp, char *comp)
4007     {
4008         prnode_t *dnp = VTOP(dp);
4009         prnode_t *pnp;
4010         vnode_t *vp = NULL;
4011         proc_t *p;
4012         file_t *fp;
4013         uint_t fd;
4014         int c;
4015         uf_entry_t *ufp;
4016         uf_info_t *fip;
4018         ASSERT(dnp->pr_type == PR_FDDIR);
4020         fd = 0;
4021         while ((c = *comp++) != '\0') {
4022             int ofd;
4023             if (c < '0' || c > '9')
4024                 return (NULL);
4025             ofd = fd;
4026             fd = 10*ofd + c - '0';
4027             if (fd/10 != ofd) /* integer overflow */
4028                 return (NULL);
4029         }
4031         pnp = prgetnode(dp, PR_FD);
4033         if (prlock(dnp, ZNO) != 0) {
4034             prfreenode(pnp);
4035             return (NULL);
4036         }
4037         p = dnp->pr_common->prc_proc;
4038         if ((p->p_flag & SSYS) || p->p_as == &kas) {
4039             prunlock(dnp);
4040             prfreenode(pnp);
4041             return (NULL);
4042         }
4044         fip = P_FINFO(p);
4045         mutex_exit(&p->p_lock);
4046         mutex_enter(&fip->fi_lock);
4047         if (fd < fip->fi_nfiles) {
4048             UF_ENTER(ufp, fip, fd);
4049             if ((fp = ufp->uf_file) != NULL) {
4050                 pnp->pr_mode = 0711;
4051                 if (fp->f_flag & FREAD)
4052                     pnp->pr_mode |= 0444;
4053                 if (fp->f_flag & FWRITE)
4054                     pnp->pr_mode |= 0222;
4055                 vp = fp->f_vnode;
4056                 VN_HOLD(vp);
4057             }
4058             UF_EXIT(ufp);
4059         }
4060         mutex_exit(&fip->fi_lock);
4061         mutex_enter(&p->p_lock);
4062         prunlock(dnp);

```



```

4064     if (vp == NULL)
4065         prfreenode(pnp);
4066     else {
4067         /*
4068          * Fill in the prnode so future references will
4069          * be able to find the underlying object's vnode.
4070          * Don't link this prnode into the list of all
4071          * prnodes for the process; this is a one-use node.
4072          */
4073         pnp->pr_realvp = vp;
4074         pnp->pr_parent = dp;          /* needed for prlookup */
4075         VN_HOLD(dp);
4076         vp = PTOV(pnp);
4077         if (pnp->pr_realvp->v_type == VDIR)
4078             vp->v_type = VDIR;
4079     }
4081     return (vp);
4082 }

4084 static vnode_t *
4085 pr_lookup_pathdir(vnode_t *dp, char *comp)
4086 {
4087     prnode_t *dpnp = VTOP(dp);
4088     prnode_t *pnp;
4089     vnode_t *vp = NULL;
4090     proc_t *p;
4091     uint_t fd, flags = 0;
4092     int c;
4093     uf_entry_t *ufp;
4094     uf_info_t *fip;
4095     enum { NAME_FD, NAME_OBJECT, NAME_ROOT, NAME_CWD, NAME_UNKNOWN } type;
4096     char *tmp;
4097     int idx;
4098     struct seg *seg;
4099     struct as *as = NULL;
4100     vattr_t vattr;

4102     ASSERT(dpnp->pr_type == PR_PATHDIR);

4104     /*
4105      * First, check if this is a numeric entry, in which case we have a
4106      * file descriptor.
4107      */
4108     fd = 0;
4109     type = NAME_FD;
4110     tmp = comp;
4111     while ((c = *tmp++) != '\0') {
4112         int ofd;
4113         if (c < '0' || c > '9') {
4114             type = NAME_UNKNOWN;
4115             break;
4116         }
4117         ofd = fd;
4118         fd = 10*ofd + c - '0';
4119         if (fd/10 != ofd) { /* integer overflow */
4120             type = NAME_UNKNOWN;
4121             break;
4122         }
4123     }

4125     /*
4126      * Next, see if it is one of the special values {root, cwd}.
4127      */
4128     if (type == NAME_UNKNOWN) {

```

```

4129         if (strcmp(comp, "root") == 0)
4130             type = NAME_ROOT;
4131         else if (strcmp(comp, "cwd") == 0)
4132             type = NAME_CWD;
4133     }

4135     /*
4136      * Grab the necessary data from the process
4137      */
4138     if (prlock(dpnp, ZNO) != 0)
4139         return (NULL);
4140     p = dpnp->pr_common->prc_proc;

4142     fip = P_FINFO(p);

4144     switch (type) {
4145     case NAME_ROOT:
4146         if ((vp = PTOU(p)->u_rdir) == NULL)
4147             vp = p->p_zone->zone_rootvp;
4148         VN_HOLD(vp);
4149         break;
4150     case NAME_CWD:
4151         vp = PTOU(p)->u_cdir;
4152         VN_HOLD(vp);
4153         break;
4154     default:
4155         if ((p->p_flag & SSYS) || (as = p->p_as) == &kas) {
4156             prunlock(dpnp);
4157             return (NULL);
4158         }
4159     }
4160     mutex_exit(&p->p_lock);

4162     /*
4163      * Determine if this is an object entry
4164      */
4165     if (type == NAME_UNKNOWN) {
4166         /*
4167          * Start with the inode index immediately after the number of
4168          * files.
4169          */
4170         mutex_enter(&fip->fi_lock);
4171         idx = fip->fi_nfiles + 4;
4172         mutex_exit(&fip->fi_lock);

4174         if (strcmp(comp, "a.out") == 0) {
4175             if (p->p_execdir != NULL) {
4176                 vp = p->p_execdir;
4177                 VN_HOLD(vp);
4178                 type = NAME_OBJECT;
4179                 flags |= PR_AOUT;
4180             } else {
4181                 vp = p->p_exec;
4182                 VN_HOLD(vp);
4183                 type = NAME_OBJECT;
4184             }
4185         } else {
4186             AS_LOCK_ENTER(as, RW_READER);
4187             if ((seg = AS_SEGFIRST(as)) != NULL) {
4188                 do {
4189                     /*
4190                      * Manufacture a filename for the
4191                      * "object" directory.
4192                      */
4193                     vattr.va_mask = AT_FSID|AT_NODEID;
4194                     if (seg->s_ops == &segvn_ops &&

```

```

4195 SEGOP_GETVP(seg, seg->s_base, &vp)
4196 == 0 &&
4197 vp != NULL && vp->v_type == VREG &&
4198 VOP_GETATTR(vp, &vattr, 0, CRED(),
4199 NULL) == 0) {
4200     char name[64];
4201
4202     if (vp == p->p_exec)
4203         continue;
4204     idx++;
4205     pr_object_name(name, vp,
4206 &vattr);
4207     if (strcmp(name, comp) == 0)
4208         break;
4209     }
4210     } while ((seg = AS_SEGNEXT(as, seg)) != NULL);
4211     }
4212
4213     if (seg == NULL) {
4214         vp = NULL;
4215     } else {
4216         VN_HOLD(vp);
4217         type = NAME_OBJECT;
4218     }
4219
4220     AS_LOCK_EXIT(as);
4221 }
4222
4223
4224 switch (type) {
4225 case NAME_FD:
4226     mutex_enter(&fip->fi_lock);
4227     if (fd < fip->fi_nfiles) {
4228         UF_ENTER(ufp, fip, fd);
4229         if (ufp->uf_file != NULL) {
4230             vp = ufp->uf_file->f_vnode;
4231             VN_HOLD(vp);
4232         }
4233         UF_EXIT(ufp);
4234     }
4235     mutex_exit(&fip->fi_lock);
4236     idx = fd + 4;
4237     break;
4238 case NAME_ROOT:
4239     idx = 2;
4240     break;
4241 case NAME_CWD:
4242     idx = 3;
4243     break;
4244 case NAME_OBJECT:
4245 case NAME_UNKNOWN:
4246     /* Nothing to do */
4247     break;
4248 }
4249
4250 mutex_enter(&p->p_lock);
4251 prunlock(dpnp);
4252
4253 if (vp != NULL) {
4254     pnp = prgetnode(dp, PR_PATH);
4255
4256     pnp->pr_flags |= flags;
4257     pnp->pr_common = dpnp->pr_common;
4258     pnp->pr_pcommon = dpnp->pr_pcommon;
4259     pnp->pr_realvp = vp;

```

```

4261     pnp->pr_parent = dp; /* needed for prlookup */
4262     pnp->pr_ino = pmkino(idx, dpnp->pr_common->prc_slot, PR_PATH);
4263     VN_HOLD(dp);
4264     vp = PTOV(pnp);
4265     vp->v_type = VLNK;
4266     }
4267
4268     return (vp);
4269 }
4270
4271 /*
4272 * Look up one of the process's active templates.
4273 */
4274 static vnode_t *
4275 pr_lookup_tmpldir(vnode_t *dp, char *comp)
4276 {
4277     prnode_t *dpnp = VTOP(dp);
4278     prnode_t *pnp;
4279     vnode_t *vp = NULL;
4280     proc_t *p;
4281     int i;
4282
4283     ASSERT(dpnp->pr_type == PR_TMPLDIR);
4284
4285     for (i = 0; i < ct_ntypes; i++)
4286         if (strcmp(comp, ct_types[i]->ct_type_name) == 0)
4287             break;
4288     if (i == ct_ntypes)
4289         return (NULL);
4290
4291     pnp = prgetnode(dp, PR_TMPL);
4292
4293     if (prlock(dpnp, ZNO) != 0) {
4294         prfreenode(pnp);
4295         return (NULL);
4296     }
4297     p = dpnp->pr_common->prc_proc;
4298     if ((p->p_flag & SSYS) || p->p_as == &kas ||
4299         (dpnp->pr_common->prc_flags & (PRC_DESTROY | PRC_LWP)) != PRC_LWP) {
4300         prunlock(dpnp);
4301         prfreenode(pnp);
4302         return (NULL);
4303     }
4304     if (ttolwp(dpnp->pr_common->prc_thread)->lwp_ct_active[i] != NULL) {
4305         pnp->pr_common = dpnp->pr_common;
4306         pnp->pr_pcommon = dpnp->pr_pcommon;
4307         pnp->pr_parent = dp;
4308         pnp->pr_cttype = i;
4309         VN_HOLD(dp);
4310         vp = PTOV(pnp);
4311     } else {
4312         prfreenode(pnp);
4313     }
4314     prunlock(dpnp);
4315
4316     return (vp);
4317 }
4318
4319 /*
4320 * Look up one of the contracts owned by the process.
4321 */
4322 static vnode_t *
4323 pr_lookup_ctdir(vnode_t *dp, char *comp)
4324 {
4325     prnode_t *dpnp = VTOP(dp);
4326     prnode_t *pnp;

```

```

4327     vnode_t *vp = NULL;
4328     proc_t *p;
4329     id_t id = 0;
4330     contract_t *ct;
4331     int c;

4333     ASSERT(dpn->pr_type == PR_CTDIR);

4335     while ((c = *comp++) != '\0') {
4336         id_t oid;
4337         if (c < '0' || c > '9')
4338             return (NULL);
4339         oid = id;
4340         id = 10 * id + c - '0';
4341         if (id / 10 != oid) /* integer overflow */
4342             return (NULL);
4343     }

4345     /*
4346      * Search all contracts; we'll filter below.
4347      */
4348     ct = contract_ptr(id, GLOBAL_ZONEUNIQID);
4349     if (ct == NULL)
4350         return (NULL);

4352     pnp = prgetnode(dp, PR_CT);

4354     if (prlock(dpn, ZNO) != 0) {
4355         prfreenode(pnp);
4356         contract_rele(ct);
4357         return (NULL);
4358     }
4359     p = dpn->pr_common->prc_proc;
4360     /*
4361      * We only allow lookups of contracts owned by this process, or,
4362      * if we are zsched and this is a zone's procfs, contracts on
4363      * stuff in the zone which are held by processes or contracts
4364      * outside the zone. (see logic in contract_status_common)
4365      */
4366     if ((ct->ct_owner != p) &&
4367         !(p == VTOZONE(dp)->zone_zsched && ct->ct_state < CTS_ORPHAN &&
4368           VTOZONE(dp)->zone_uniqid == contract_getzuniqid(ct) &&
4369           VTOZONE(dp)->zone_uniqid != GLOBAL_ZONEUNIQID &&
4370           ct->ct_czuniqid == GLOBAL_ZONEUNIQID)) {
4371         prunlock(dpn);
4372         prfreenode(pnp);
4373         contract_rele(ct);
4374         return (NULL);
4375     }
4376     pnp->pr_common = dpn->pr_common;
4377     pnp->pr_pcommon = dpn->pr_pcommon;
4378     pnp->pr_contract = ct;
4379     pnp->pr_parent = dp;
4380     pnp->pr_ino = pmkino(id, pnp->pr_common->prc_slot, PR_CT);
4381     VN_HOLD(dp);
4382     prunlock(dpn);
4383     vp = PTOV(pnp);

4385     return (vp);
4386 }

4388 /*
4389  * Construct an lwp vnode for the old /proc interface.
4390  * We stand on our head to make the /proc plumbing correct.
4391  */
4392     vnode_t *

```

```

4393     prlwpnode(prnode_t *pnp, uint_t tid)
4394     {
4395         char comp[12];
4396         vnode_t *dp;
4397         vnode_t *vp;
4398         prcommon_t *pcp;
4399         proc_t *p;

4401         /*
4402          * Lookup the /proc/<pid>/lwp/<lwpid> directory vnode.
4403          */
4404         if (pnp->pr_type == PR_PIDFILE) {
4405             dp = pnp->pr_parent; /* /proc/<pid> */
4406             VN_HOLD(dp);
4407             vp = pr_lookup_piddir(dp, "lwp");
4408             VN_RELE(dp);
4409             if ((dp = vp) == NULL) /* /proc/<pid>/lwp */
4410                 return (NULL);
4411         } else if (pnp->pr_type == PR_LWPIDFILE) {
4412             dp = pnp->pr_parent; /* /proc/<pid>/lwp/<lwpid> */
4413             dp = VTOP(dp)->pr_parent; /* /proc/<pid>/lwp */
4414             VN_HOLD(dp);
4415         } else {
4416             return (NULL);
4417         }

4419         (void) pr_u32tos(tid, comp, sizeof (comp));
4420         vp = pr_lookup_lwpdir(dp, comp);
4421         VN_RELE(dp);
4422         if ((dp = vp) == NULL)
4423             return (NULL);

4425         pnp = prgetnode(dp, PR_LWPIDFILE);
4426         vp = PTOV(pnp);

4428         /*
4429          * prgetnode() initialized most of the prnode.
4430          * Finish the job.
4431          */
4432         pcp = VTOP(dp)->pr_common;
4433         pnp->pr_ino = ptoi(pcp->prc_pid);
4434         pnp->pr_common = pcp;
4435         pnp->pr_pcommon = VTOP(dp)->pr_pcommon;
4436         pnp->pr_parent = dp;
4437         /*
4438          * Link new vnode into list of all /proc vnodes for the process.
4439          */
4440         p = pr_p_lock(pnp);
4441         mutex_exit(&pr_pidlock);
4442         if (p == NULL) {
4443             VN_RELE(dp);
4444             prfreenode(pnp);
4445             vp = NULL;
4446         } else if (pcp->prc_thread == NULL) {
4447             prunlock(pnp);
4448             VN_RELE(dp);
4449             prfreenode(pnp);
4450             vp = NULL;
4451         } else {
4452             pnp->pr_next = p->p_plist;
4453             p->p_plist = vp;
4454             prunlock(pnp);
4455         }

4457         return (vp);
4458     }

```

```

4460 #if defined(DEBUG)
4462 static uint32_t nprnode;
4463 static uint32_t nprcommon;
4465 #define INCREMENT(x)    atomic_inc_32(&x);
4466 #define DECREMENT(x)   atomic_dec_32(&x);
4468 #else
4470 #define INCREMENT(x)
4471 #define DECREMENT(x)
4473 #endif /* DEBUG */
4475 /*
4476  * New /proc vnode required; allocate it and fill in most of the fields.
4477  */
4478 prnode_t *
4479 prgetnode(vnode_t *dp, prnodetype_t type)
4480 {
4481     prnode_t *pnp;
4482     prcommon_t *pcp;
4483     vnode_t *vp;
4484     ulong_t nfiles;
4486     INCREMENT(nprnode);
4487     pnp = kmem_zalloc(sizeof (prnode_t), KM_SLEEP);
4489     mutex_init(&pnp->pr_mutex, NULL, MUTEX_DEFAULT, NULL);
4490     pnp->pr_type = type;
4492     pnp->pr_vnode = vn_alloc(KM_SLEEP);
4494     vp = PTOV(pnp);
4495     vp->v_flag = VNOCACHE|VNOMAP|VNOSWAP|VNOMOUNT;
4496     vn_setops(vp, prvnodeops);
4497     vp->v_vfsp = dp->v_vfsp;
4498     vp->v_type = VPROC;
4499     vp->v_data = (caddr_t)pnp;
4501     switch (type) {
4502     case PR_PIDDIR:
4503     case PR_LWPIDDIR:
4504         /*
4505          * We need a prcommon and a files array for each of these.
4506          */
4507         INCREMENT(nprcommon);
4509         pcp = kmem_zalloc(sizeof (prcommon_t), KM_SLEEP);
4510         pcp->prc_refcnt = 1;
4511         pnp->pr_common = pcp;
4512         mutex_init(&pcp->prc_mutex, NULL, MUTEX_DEFAULT, NULL);
4513         cv_init(&pcp->prc_wait, NULL, CV_DEFAULT, NULL);
4515         nfiles = (type == PR_PIDDIR)? NPIDDIRFILES : NLWPIDDIRFILES;
4516         pnp->pr_files =
4517             kmem_zalloc(nfiles * sizeof (vnode_t *), KM_SLEEP);
4519         vp->v_type = VDIR;
4520         /*
4521          * Mode should be read-search by all, but we cannot so long
4522          * as we must support compatibility mode with old /proc.
4523          * Make /proc/<pid> be read by owner only, search by all.
4524          * Make /proc/<pid>/lwp/<lwpid> read-search by all. Also,

```

```

4525         * set VDIROPEN on /proc/<pid> so it can be opened for writing.
4526         */
4527         if (type == PR_PIDDIR) {
4528             /* kludge for old /proc interface */
4529             prnode_t *xpnp = prgetnode(dp, PR_PIDFILE);
4530             pnp->pr_pidfile = PTOV(xpnp);
4531             pnp->pr_mode = 0511;
4532             vp->v_flag |= VDIROPEN;
4533         } else {
4534             pnp->pr_mode = 0555;
4535         }
4537         break;
4539     case PR_CURDIR:
4540     case PR_ROOTDIR:
4541     case PR_FDDIR:
4542     case PR_OBJECTDIR:
4543     case PR_PATHDIR:
4544     case PR_CTDIR:
4545     case PR_TPLDIR:
4546         vp->v_type = VDIR;
4547         pnp->pr_mode = 0500; /* read-search by owner only */
4548         break;
4550     case PR_CT:
4551         vp->v_type = VLNK;
4552         pnp->pr_mode = 0500; /* read-search by owner only */
4553         break;
4555     case PR_PATH:
4556     case PR_SELF:
4557         vp->v_type = VLNK;
4558         pnp->pr_mode = 0777;
4559         break;
4561     case PR_LWPDIR:
4562         vp->v_type = VDIR;
4563         pnp->pr_mode = 0555; /* read-search by all */
4564         break;
4566     case PR_AS:
4567     case PR_TPL:
4568         pnp->pr_mode = 0600; /* read-write by owner only */
4569         break;
4571     case PR_CTL:
4572     case PR_LWPCTL:
4573         pnp->pr_mode = 0200; /* write-only by owner only */
4574         break;
4576     case PR_PIDFILE:
4577     case PR_LWPIDFILE:
4578         pnp->pr_mode = 0600; /* read-write by owner only */
4579         break;
4581     case PR_PSINFO:
4582     case PR_LPSINFO:
4583     case PR_LWPSINFO:
4584     case PR_USAGE:
4585     case PR_LUSAGE:
4586     case PR_LWPUSAGE:
4587         pnp->pr_mode = 0444; /* read-only by all */
4588         break;
4590     default:

```

```

4591         pnp->pr_mode = 0400; /* read-only by owner only */
4592         break;
4593     }
4594     vn_exists(vp);
4595     return (pnp);
4596 }

4598 /*
4599  * Free the storage obtained from prgetnode().
4600  */
4601 void
4602 prfreenode(prnode_t *pnp)
4603 {
4604     vnode_t *vp;
4605     ulong_t nfiles;

4607     vn_invalid(PTOV(pnp));
4608     vn_free(PTOV(pnp));
4609     mutex_destroy(&pnp->pr_mutex);

4611     switch (pnp->pr_type) {
4612     case PR_PIDDIR:
4613         /* kludge for old /proc interface */
4614         if (pnp->pr_pidfile != NULL) {
4615             prfreenode(VTOP(pnp->pr_pidfile));
4616             pnp->pr_pidfile = NULL;
4617         }
4618         /* FALLTHROUGH */
4619     case PR_LWPIDDIR:
4620         /*
4621          * We allocated a prcommon and a files array for each of these.
4622          */
4623         prfreecommon(pnp->pr_common);
4624         nfiles = (pnp->pr_type == PR_PIDDIR)?
4625             NPIDDIRFILES: NLWPIDDIRFILES;
4626         kmem_free(pnp->pr_files, nfiles * sizeof (vnode_t *));
4627         break;
4628     default:
4629         break;
4630     }
4631     /*
4632      * If there is an underlying vnode, be sure
4633      * to release it after freeing the prnode.
4634      */
4635     vp = pnp->pr_realvp;
4636     kmem_free(pnp, sizeof (*pnp));
4637     DECREMENT(nprnode);
4638     if (vp != NULL) {
4639         VN_RELE(vp);
4640     }
4641 }

4643 /*
4644  * Free a prcommon structure, if the reference count reaches zero.
4645  */
4646 static void
4647 prfreecommon(prcommon_t *pcp)
4648 {
4649     mutex_enter(&pcp->prc_mutex);
4650     ASSERT(pcp->prc_refcnt > 0);
4651     if (--pcp->prc_refcnt != 0)
4652         mutex_exit(&pcp->prc_mutex);
4653     else {
4654         mutex_exit(&pcp->prc_mutex);
4655         ASSERT(pcp->prc_pollhead.ph_list == NULL);
4656         ASSERT(pcp->prc_refcnt == 0);

```

```

4657         ASSERT(pcp->prc_selfopens == 0 && pcp->prc_writers == 0);
4658         mutex_destroy(&pcp->prc_mutex);
4659         cv_destroy(&pcp->prc_wait);
4660         kmem_free(pcp, sizeof (prcommon_t));
4661         DECREMENT(nprcommon);
4662     }
4663 }

4665 /*
4666  * Array of readdir functions, indexed by /proc file type.
4667  */
4668 static int pr_readdir_notdir(), pr_readdir_procdir(), pr_readdir_piddir(),
4669     pr_readdir_objctdir(), pr_readdir_lwpdir(), pr_readdir_lwpiddir(),
4670     pr_readdir_fddir(), pr_readdir_pathdir(), pr_readdir_tmpldir(),
4671     pr_readdir_ctdir();

4673 static int (*pr_readdir_function[PR_NFILES])() = {
4674     pr_readdir_procdir, /* /proc */
4675     pr_readdir_notdir, /* /proc/self */
4676     pr_readdir_piddir, /* /proc/<pid> */
4677     pr_readdir_notdir, /* /proc/<pid>/as */
4678     pr_readdir_notdir, /* /proc/<pid>/ctl */
4679     pr_readdir_notdir, /* /proc/<pid>/status */
4680     pr_readdir_notdir, /* /proc/<pid>/lstatus */
4681     pr_readdir_notdir, /* /proc/<pid>/psinfo */
4682     pr_readdir_notdir, /* /proc/<pid>/lpsinfo */
4683     pr_readdir_notdir, /* /proc/<pid>/map */
4684     pr_readdir_notdir, /* /proc/<pid>/rmap */
4685     pr_readdir_notdir, /* /proc/<pid>/xmap */
4686     pr_readdir_notdir, /* /proc/<pid>/cred */
4687     pr_readdir_notdir, /* /proc/<pid>/sigact */
4688     pr_readdir_notdir, /* /proc/<pid>/auxv */
4689     #if defined(__x86)
4690     pr_readdir_notdir, /* /proc/<pid>/ldt */
4691     #endif
4692     pr_readdir_notdir, /* /proc/<pid>/usage */
4693     pr_readdir_notdir, /* /proc/<pid>/lusage */
4694     pr_readdir_notdir, /* /proc/<pid>/pagedata */
4695     pr_readdir_notdir, /* /proc/<pid>/watch */
4696     pr_readdir_notdir, /* /proc/<pid>/cwd */
4697     pr_readdir_notdir, /* /proc/<pid>/root */
4698     pr_readdir_fddir, /* /proc/<pid>/fd */
4699     pr_readdir_notdir, /* /proc/<pid>/fd/nn */
4700     pr_readdir_objctdir, /* /proc/<pid>/object */
4701     pr_readdir_notdir, /* /proc/<pid>/object/xxx */
4702     pr_readdir_lwpdir, /* /proc/<pid>/lwp */
4703     pr_readdir_lwpiddir, /* /proc/<pid>/lwp/<lwpid> */
4704     pr_readdir_notdir, /* /proc/<pid>/lwp/<lwpid>/lwpctl */
4705     pr_readdir_notdir, /* /proc/<pid>/lwp/<lwpid>/lwpstatus */
4706     pr_readdir_notdir, /* /proc/<pid>/lwp/<lwpid>/lwpsinfo */
4707     pr_readdir_notdir, /* /proc/<pid>/lwp/<lwpid>/lwpsusage */
4708     pr_readdir_notdir, /* /proc/<pid>/lwp/<lwpid>/xregs */
4709     pr_readdir_tmpldir, /* /proc/<pid>/lwp/<lwpid>/templates */
4710     pr_readdir_notdir, /* /proc/<pid>/lwp/<lwpid>/templates/<id> */
4711     pr_readdir_notdir, /* /proc/<pid>/lwp/<lwpid>/spymaster */
4712     #if defined(__sparc)
4713     pr_readdir_notdir, /* /proc/<pid>/lwp/<lwpid>/gwindows */
4714     pr_readdir_notdir, /* /proc/<pid>/lwp/<lwpid>/asrs */
4715     #endif
4716     pr_readdir_notdir, /* /proc/<pid>/priv */
4717     pr_readdir_pathdir, /* /proc/<pid>/path */
4718     pr_readdir_notdir, /* /proc/<pid>/path/xxx */
4719     pr_readdir_ctdir, /* /proc/<pid>/contracts */
4720     pr_readdir_notdir, /* /proc/<pid>/contracts/<ctid> */
4721     pr_readdir_notdir, /* /proc/<pid>/secflags */
4722     #endif /* ! codereview */

```

```

4723     pr_readdir_notdir,      /* old process file */
4724     pr_readdir_notdir,      /* old lwp file */
4725     pr_readdir_notdir,      /* old pagedata file */
4726 };

4728 /* ARGSUSED */
4729 static int
4730 pr_readdir(vnode_t *vp, uio_t *uiop, cred_t *cr, int *eofp,
4731 caller_context_t *ct, int flags)
4732 {
4733     prnode_t *pnp = VTOP(vp);

4735     ASSERT(pnp->pr_type < PR_NFILES);

4737     /* XXX - Do we need to pass ct and flags? */
4738     return (pr_readdir_function[pnp->pr_type](pnp, uiop, eofp));
4739 }

4741 /* ARGSUSED */
4742 static int
4743 pr_readdir_notdir(prnode_t *pnp, uio_t *uiop, int *eofp)
4744 {
4745     return (ENOTDIR);
4746 }

4748 /* ARGSUSED */
4749 static int
4750 pr_readdir_procdir(prnode_t *pnp, uio_t *uiop, int *eofp)
4751 {
4752     zoneid_t zoneid;
4753     gfs_readdir_state_t gstate;
4754     int error, eof = 0;
4755     offset_t n;

4757     ASSERT(pnp->pr_type == PR_PROCDIR);

4759     zoneid = VTOZONE(PTOV(pnp))->zone_id;

4761     if ((error = gfs_readdir_init(&gstate, PNSIZ, PRSDSIZE, uiop,
4762 PRROOTINO, PRROOTINO, 0)) != 0)
4763         return (error);

4765     /*
4766      * Loop until user's request is satisfied or until all processes
4767      * have been examined.
4768      */
4769     while ((error = gfs_readdir_pred(&gstate, uiop, &n)) == 0) {
4770         uint_t pid;
4771         int pslot;
4772         proc_t *p;

4774         /*
4775          * Find next entry. Skip processes not visible where
4776          * this /proc was mounted.
4777          */
4778         mutex_enter(&pidlock);
4779         while (n < v.v_proc &&
4780 ((p = pid_entry(n)) == NULL || p->p_stat == SIDL ||
4781 (zoneid != GLOBAL_ZONEID && p->p_zone->zone_id != zoneid) ||
4782 secpolicy_basic_procinfo(CRED(), p, curproc) != 0))
4783             n++;

4785         /*
4786          * Stop when entire proc table has been examined.
4787          */
4788         if (n >= v.v_proc) {

```

```

4789         mutex_exit(&pidlock);
4790         eof = 1;
4791         break;
4792     }

4794     ASSERT(p->p_stat != 0);
4795     pid = p->p_pid;
4796     pslot = p->p_slot;
4797     mutex_exit(&pidlock);
4798     error = gfs_readdir_emitn(&gstate, uiop, n,
4799 pmkino(0, pslot, PR_PIDDIR), pid);
4800     if (error)
4801         break;
4802 }

4804     return (gfs_readdir_fini(&gstate, error, eofp, eof));
4805 }

4807 /* ARGSUSED */
4808 static int
4809 pr_readdir_piddir(prnode_t *pnp, uio_t *uiop, int *eofp)
4810 {
4811     int zombie = ((pnp->pr_pcommon->prc_flags & PRC_DESTROY) != 0);
4812     prdirent_t dirent;
4813     prdirent_t *dirp;
4814     offset_t off;
4815     int error;

4817     ASSERT(pnp->pr_type == PR_PIDDIR);

4819     if (uiop->uio_offset < 0 ||
4820 uiop->uio_offset % sizeof (prdirent_t) != 0 ||
4821 uiop->uio_resid < sizeof (prdirent_t))
4822         return (EINVAL);
4823     if (pnp->pr_pcommon->prc_proc == NULL)
4824         return (ENOENT);
4825     if (uiop->uio_offset >= sizeof (piddir))
4826         goto out;

4828     /*
4829      * Loop until user's request is satisfied, omitting some
4830      * files along the way if the process is a zombie.
4831      */
4832     for (dirp = &piddir[uiop->uio_offset / sizeof (prdirent_t)];
4833 uiop->uio_resid >= sizeof (prdirent_t) &&
4834 dirp < &piddir[NPIDDIRFILES+2];
4835 uiop->uio_offset = off + sizeof (prdirent_t), dirp++) {
4836         off = uiop->uio_offset;
4837         if (zombie) {
4838             switch (dirp->d_ino) {
4839                 case PR_PIDDIR:
4840                 case PR_PROCDIR:
4841                 case PR_PSINFO:
4842                 case PR_USAGE:
4843                     break;
4844                 default:
4845                     continue;
4846             }
4847         }
4848         bcopy(dirp, &dirent, sizeof (prdirent_t));
4849         if (dirent.d_ino == PR_PROCDIR)
4850             dirent.d_ino = PRROOTINO;
4851         else
4852             dirent.d_ino = pmkino(0, pnp->pr_pcommon->prc_slot,
4853 dirent.d_ino);
4854         if ((error = uiomove((caddr_t)&dirent, sizeof (prdirent_t),

```

```

4855         UIO_READ, uiop)) != 0)
4856             return (error);
4857     }
4858 out:
4859     if (eofp)
4860         *eofp = (uiop->uio_offset >= sizeof (piddir));
4861     return (0);
4862 }

4864 static void
4865 rebuild_objdir(struct as *as)
4866 {
4867     struct seg *seg;
4868     vnode_t *vp;
4869     vattn_t vattn;
4870     vnode_t **dir;
4871     ulong_t nalloc;
4872     ulong_t nentries;
4873     int i, j;
4874     ulong_t nold, nnew;

4876     ASSERT(AS_WRITE_HELD(as));

4878     if (as->a_updatedir == 0 && as->a_objectdir != NULL)
4879         return;
4880     as->a_updatedir = 0;

4882     if ((nalloc = avl_numnodes(&as->a_segtree)) == 0 ||
4883         (seg = AS_SEGFIRST(as)) == NULL) /* can't happen? */
4884         return;

4886     /*
4887      * Allocate space for the new object directory.
4888      * (This is usually about two times too many entries.)
4889      */
4890     nalloc = (nalloc + 0xf) & ~0xf; /* multiple of 16 */
4891     dir = kmem_zalloc(nalloc * sizeof (vnode_t *), KM_SLEEP);

4893     /* fill in the new directory with desired entries */
4894     nentries = 0;
4895     do {
4896         vattn.va_mask = AT_FSID|AT_NODEID;
4897         if (seg->s_ops == &segvn_ops &&
4898             SEGOP_GETVP(seg, seg->s_base, &vp) == 0 &&
4899             vp != NULL && vp->v_type == VREG &&
4900             VOP_GETATTR(vp, &vattn, 0, CRED(), NULL) == 0) {
4901             for (i = 0; i < nentries; i++)
4902                 if (vp == dir[i])
4903                     break;
4904             if (i == nentries) {
4905                 ASSERT(nentries < nalloc);
4906                 dir[nentries++] = vp;
4907             }
4908         }
4909     } while ((seg = AS_SEGNEXT(as, seg)) != NULL);

4911     if (as->a_objectdir == NULL) { /* first time */
4912         as->a_objectdir = dir;
4913         as->a_sizedir = nalloc;
4914         return;
4915     }

4917     /*
4918      * Null out all of the defunct entries in the old directory.
4919      */
4920     nold = 0;

```

```

4921     nnew = nentries;
4922     for (i = 0; i < as->a_sizedir; i++) {
4923         if ((vp = as->a_objectdir[i]) != NULL) {
4924             for (j = 0; j < nentries; j++) {
4925                 if (vp == dir[j]) {
4926                     dir[j] = NULL;
4927                     nnew--;
4928                     break;
4929                 }
4930             }
4931             if (j == nentries)
4932                 as->a_objectdir[i] = NULL;
4933             else
4934                 nold++;
4935         }
4936     }

4938     if (nold + nnew > as->a_sizedir) {
4939         /*
4940          * Reallocate the old directory to have enough
4941          * space for the old and new entries combined.
4942          * Round up to the next multiple of 16.
4943          */
4944         ulong_t newsize = (nold + nnew + 0xf) & ~0xf;
4945         vnode_t **newdir = kmem_zalloc(newsize * sizeof (vnode_t *),
4946             KM_SLEEP);
4947         bcopy(as->a_objectdir, newdir,
4948             as->a_sizedir * sizeof (vnode_t *));
4949         kmem_free(as->a_objectdir, as->a_sizedir * sizeof (vnode_t *));
4950         as->a_objectdir = newdir;
4951         as->a_sizedir = newsize;
4952     }

4954     /*
4955      * Move all new entries to the old directory and
4956      * deallocate the space used by the new directory.
4957      */
4958     if (nnew) {
4959         for (i = 0, j = 0; i < nentries; i++) {
4960             if ((vp = dir[i]) == NULL)
4961                 continue;
4962             for (; j < as->a_sizedir; j++) {
4963                 if (as->a_objectdir[j] != NULL)
4964                     continue;
4965                 as->a_objectdir[j++] = vp;
4966                 break;
4967             }
4968         }
4969     }
4970     kmem_free(dir, nalloc * sizeof (vnode_t *));
4971 }

4973 /*
4974  * Return the vnode from a slot in the process's object directory.
4975  * The caller must have locked the process's address space.
4976  * The only caller is below, in pr_readdir_objdir().
4977  */
4978 static vnode_t *
4979 obj_entry(struct as *as, int slot)
4980 {
4981     ASSERT(AS_LOCK_HELD(as));
4982     if (as->a_objectdir == NULL)
4983         return (NULL);
4984     ASSERT(slot < as->a_sizedir);
4985     return (as->a_objectdir[slot]);
4986 }

```

```

4988 /* ARGSUSED */
4989 static int
4990 pr_readdir_objectdir(prnode_t *pnp, uio_t *uiop, int *eofp)
4991 {
4992     gfs_readdir_state_t gstate;
4993     int error, eof = 0;
4994     offset_t n;
4995     int pslot;
4996     size_t objdirsize;
4997     proc_t *p;
4998     struct as *as;
4999     vnode_t *vp;
5001     ASSERT(pnp->pr_type == PR_OBJECTDIR);
5003     if ((error = prlock(pnp, ZNO)) != 0)
5004         return (error);
5005     p = pnp->pr_common->prc_proc;
5006     pslot = p->p_slot;
5008     /*
5009     * We drop p_lock before grabbing the address space lock
5010     * in order to avoid a deadlock with the clock thread.
5011     * The process will not disappear and its address space
5012     * will not change because it is marked P_PR_LOCK.
5013     */
5014     mutex_exit(&p->p_lock);
5016     if ((error = gfs_readdir_init(&gstate, 64, PRSDSIZE, uiop,
5017         pmkino(0, pslot, PR_PIDDIR),
5018         pmkino(0, pslot, PR_OBJECTDIR), 0)) != 0) {
5019         mutex_enter(&p->p_lock);
5020         prunlock(pnp);
5021         return (error);
5022     }
5024     if ((p->p_flag & SSYS) || (as = p->p_as) == &kas) {
5025         as = NULL;
5026         objdirsize = 0;
5027     }
5029     /*
5030     * Loop until user's request is satisfied or until
5031     * all mapped objects have been examined. Cannot hold
5032     * the address space lock for the following call as
5033     * gfs_readdir_pred() ultimately causes a call to uiomove().
5034     */
5035     while ((error = gfs_readdir_pred(&gstate, uiop, &n)) == 0) {
5036         vattr_t vattr;
5037         char str[64];
5039         /*
5040         * Set the correct size of the directory just
5041         * in case the process has changed it's address
5042         * space via mmap/munmap calls.
5043         */
5044         if (as != NULL) {
5045             AS_LOCK_ENTER(as, RW_WRITER);
5046             if (as->a_updatedir)
5047                 rebuild_objdir(as);
5048             objdirsize = as->a_sizedir;
5049         }
5051         /*
5052         * Find next object.

```

```

5053         /*
5054         vattr.va_mask = AT_FSID | AT_NODEID;
5055         while (n < objdirsize && (((vp = obj_entry(as, n)) == NULL) ||
5056             (VOP_GETATTR(vp, &vattr, 0, CRED(), NULL)
5057                 != 0))) {
5058             vattr.va_mask = AT_FSID | AT_NODEID;
5059             n++;
5060         }
5062         if (as != NULL)
5063             AS_LOCK_EXIT(as);
5065         /*
5066         * Stop when all objects have been reported.
5067         */
5068         if (n >= objdirsize) {
5069             eof = 1;
5070             break;
5071         }
5073         if (vp == p->p_exec)
5074             (void) strcpy(str, "a.out");
5075         else
5076             pr_object_name(str, vp, &vattr);
5078         error = gfs_readdir_emit(&gstate, uiop, n, vattr.va_nodeid,
5079             str, 0);
5081         if (error)
5082             break;
5083     }
5085     mutex_enter(&p->p_lock);
5086     prunlock(pnp);
5088     return (gfs_readdir_fini(&gstate, error, eofp, eof));
5089 }
5091 /* ARGSUSED */
5092 static int
5093 pr_readdir_lwpdir(prnode_t *pnp, uio_t *uiop, int *eofp)
5094 {
5095     gfs_readdir_state_t gstate;
5096     int error, eof = 0;
5097     offset_t tslot;
5098     proc_t *p;
5099     int pslot;
5100     lwpdir_t *lwpdir;
5101     int lwpdirsize;
5103     ASSERT(pnp->pr_type == PR_LWPDIR);
5105     p = pr_p_lock(pnp);
5106     mutex_exit(&pr_pidlock);
5107     if (p == NULL)
5108         return (ENOENT);
5109     ASSERT(p == pnp->pr_common->prc_proc);
5110     pslot = p->p_slot;
5111     lwpdir = p->p_lwpdir;
5112     lwpdirsize = p->p_lwpdir_sz;
5114     /*
5115     * Drop p->p_lock so we can safely do uiomove().
5116     * The lwp directory will not change because
5117     * we have the process locked with P_PR_LOCK.
5118     */

```



```

5119     mutex_exit(&p->p_lock);

5122     if ((error = gfs_readdir_init(&gstate, PLNSIZ, PRSDSIZE, uiop,
5123     pmkino(0, pslot, PR_PIDDIR),
5124     pmkino(0, pslot, PR_LWPDIR), 0)) != 0) {
5125         mutex_enter(&p->p_lock);
5126         prunlock(pnp);
5127         return (error);
5128     }

5130     /*
5131     * Loop until user's request is satisfied or until all lwps
5132     * have been examined.
5133     */
5134     while ((error = gfs_readdir_pred(&gstate, uiop, &tslot)) == 0) {
5135         lwpent_t *lep;
5136         uint_t tid;

5138         /*
5139         * Find next LWP.
5140         */
5141         while (tslot < lwpdirsize &&
5142             ((lep = lwpdir[tslot].ld_entry) == NULL))
5143             tslot++;
5144         /*
5145         * Stop when all lwps have been reported.
5146         */
5147         if (tslot >= lwpdirsize) {
5148             eof = 1;
5149             break;
5150         }

5152         tid = lep->le_lwpid;
5153         error = gfs_readdir_emitn(&gstate, uiop, tslot,
5154             pmkino(tslot, pslot, PR_LWPDIR), tid);
5155         if (error)
5156             break;
5157     }

5159     mutex_enter(&p->p_lock);
5160     prunlock(pnp);

5162     return (gfs_readdir_fini(&gstate, error, eofp, eof));
5163 }

5165 /* ARGSUSED */
5166 static int
5167 pr_readdir_lwppid(prnode_t *pnp, uio_t *uiop, int *eofp)
5168 {
5169     prcommon_t *pcp = pnp->pr_common;
5170     int zombie = ((pcp->prc_flags & PRC_DESTROY) != 0);
5171     prdirent_t dirent;
5172     prdirent_t *dirp;
5173     offset_t off;
5174     int error;
5175     int pslot;
5176     int tslot;

5178     ASSERT(pnp->pr_type == PR_LWPDIR);

5180     if (uiop->uio_offset < 0 ||
5181         uiop->uio_offset % sizeof (prdirent_t) != 0 ||
5182         uiop->uio_resid < sizeof (prdirent_t))
5183         return (EINVAL);
5184     if (pcp->prc_proc == NULL || pcp->prc_tslot == -1)

```

```

5185         return (ENOENT);
5186     if (uiop->uio_offset >= sizeof (lwppid))
5187         goto out;

5189     /*
5190     * Loop until user's request is satisfied, omitting some files
5191     * along the way if the lwp is a zombie and also depending
5192     * on the data model of the process.
5193     */
5194     pslot = pcp->prc_slot;
5195     tslot = pcp->prc_tslot;
5196     for (dirp = &lwppid[uiop->uio_offset / sizeof (prdirent_t)];
5197         uiop->uio_resid >= sizeof (prdirent_t) &&
5198         dirp < &lwppid[NLWPPIDDIRFILES+2];
5199         uiop->uio_offset = off + sizeof (prdirent_t), dirp++) {
5200         off = uiop->uio_offset;
5201         if (zombie) {
5202             switch (dirp->d_ino) {
5203             case PR_LWPPIDDIR:
5204             case PR_LWPDIR:
5205             case PR_LWPSINFO:
5206                 break;
5207             default:
5208                 continue;
5209             }
5210         }
5211 #if defined(__sparc)
5212         /* the asrs file exists only for sparc v9_LP64 processes */
5213         if (dirp->d_ino == PR_ASRS &&
5214             pcp->prc_datamodel != DATAMODEL_LP64)
5215             continue;
5216 #endif
5217         bcopy(dirp, &dirent, sizeof (prdirent_t));
5218         if (dirent.d_ino == PR_LWPDIR)
5219             dirent.d_ino = pmkino(0, pslot, dirp->d_ino);
5220         else
5221             dirent.d_ino = pmkino(tslot, pslot, dirp->d_ino);
5222         if ((error = uiomove((caddr_t)&dirent, sizeof (prdirent_t),
5223             UIO_READ, uiop)) != 0)
5224             return (error);
5225     }
5226 out:
5227     if (eofp)
5228         *eofp = (uiop->uio_offset >= sizeof (lwppid));
5229     return (0);
5230 }

5232 /* ARGSUSED */
5233 static int
5234 pr_readdir_fddir(prnode_t *pnp, uio_t *uiop, int *eofp)
5235 {
5236     gfs_readdir_state_t gstate;
5237     int error, eof = 0;
5238     offset_t n;
5239     proc_t *p;
5240     int pslot;
5241     int fddirsize;
5242     uf_info_t *fip;

5244     ASSERT(pnp->pr_type == PR_FDDIR);

5246     if ((error = prlock(pnp, ZNO)) != 0)
5247         return (error);
5248     p = pnp->pr_common->prc_proc;
5249     pslot = p->p_slot;
5250     fip = P_FINFO(p);

```

```

5251     mutex_exit(&p->p_lock);

5253     if ((error = gfs_readdir_init(&gstate, PLNSIZ, PRSDSIZE, uiop,
5254         pmkino(0, pslot, PR_PIDDIR), pmkino(0, pslot, PR_FDDIR), 0) != 0) {
5255         mutex_enter(&p->p_lock);
5256         prunlock(pnp);
5257         return (error);
5258     }

5260     mutex_enter(&fip->fi_lock);
5261     if ((p->p_flag & SSYS) || p->p_as == &kas)
5262         fddirsize = 0;
5263     else
5264         fddirsize = fip->fi_nfiles;

5266     /*
5267     * Loop until user's request is satisfied or until
5268     * all file descriptors have been examined.
5269     */
5270     while ((error = gfs_readdir_pred(&gstate, uiop, &n) == 0) {
5271         /*
5272         * Find next fd.
5273         */
5274         while (n < fddirsize && fip->fi_list[n].uf_file == NULL)
5275             n++;
5276         /*
5277         * Stop when all fds have been reported.
5278         */
5279         if (n >= fddirsize) {
5280             eof = 1;
5281             break;
5282         }

5284         error = gfs_readdir_emitn(&gstate, uiop, n,
5285             pmkino(n, pslot, PR_FD), n);
5286         if (error)
5287             break;
5288     }

5290     mutex_exit(&fip->fi_lock);
5291     mutex_enter(&p->p_lock);
5292     prunlock(pnp);

5294     return (gfs_readdir_fini(&gstate, error, eofp, eof));
5295 }

5297 /* ARGSUSED */
5298 static int
5299 pr_readdir_pathdir(prnode_t *pnp, uio_t *uiop, int *eofp)
5300 {
5301     longlong_t bp[DIRENT64_RECLEN(64) / sizeof(longlong_t)];
5302     dirent64_t *dirent = (dirent64_t *)bp;
5303     int reclen;
5304     ssize_t oresid;
5305     offset_t off, idx;
5306     int error = 0;
5307     proc_t *p;
5308     int fd, obj;
5309     int pslot;
5310     int fddirsize;
5311     uf_info_t *fip;
5312     struct as *as = NULL;
5313     size_t objdirsize;
5314     vattr_t vattr;
5315     vnode_t *vp;

```

```

5317     ASSERT(pnp->pr_type == PR_PATHDIR);

5319     if (uiop->uio_offset < 0 ||
5320         uiop->uio_resid <= 0 ||
5321         (uiop->uio_offset % PRSDSIZE) != 0)
5322         return (EINVAL);
5323     oresid = uiop->uio_resid;
5324     bzero(bp, sizeof(bp));

5326     if ((error = prlock(pnp, ZNO)) != 0)
5327         return (error);
5328     p = pnp->pr_common->prc_proc;
5329     fip = P_FINFO(p);
5330     pslot = p->p_slot;
5331     mutex_exit(&p->p_lock);

5333     if ((p->p_flag & SSYS) || (as = p->p_as) == &kas) {
5334         as = NULL;
5335         objdirsize = 0;
5336     } else {
5337         AS_LOCK_ENTER(as, RW_WRITER);
5338         if (as->a_updatedir)
5339             rebuild_objdir(as);
5340         objdirsize = as->a_sizedir;
5341         AS_LOCK_EXIT(as);
5342         as = NULL;
5343     }

5345     mutex_enter(&fip->fi_lock);
5346     if ((p->p_flag & SSYS) || p->p_as == &kas)
5347         fddirsize = 0;
5348     else
5349         fddirsize = fip->fi_nfiles;

5351     for (; uiop->uio_resid > 0; uiop->uio_offset = off + PRSDSIZE) {
5352         /*
5353         * There are 4 special files in the path directory: ".", "..",
5354         * "root", and "cwd". We handle those specially here.
5355         */
5356         off = uiop->uio_offset;
5357         idx = off / PRSDSIZE;
5358         if (off == 0) {
5359             /* "." */
5360             dirent->d_ino = pmkino(0, pslot, PR_PATHDIR);
5361             dirent->d_name[0] = '.';
5362             dirent->d_name[1] = '\0';
5363             reclen = DIRENT64_RECLEN(1);
5364         } else if (idx == 1) {
5365             /* ".." */
5366             dirent->d_ino = pmkino(0, pslot, PR_PIDDIR);
5367             dirent->d_name[0] = '.';
5368             dirent->d_name[1] = '.';
5369             dirent->d_name[2] = '\0';
5370             reclen = DIRENT64_RECLEN(2);
5371         } else if (idx == 2) {
5372             /* "root" */
5373             dirent->d_ino = pmkino(idx, pslot, PR_PATH);
5374             (void) strcpy(dirent->d_name, "root");
5375             reclen = DIRENT64_RECLEN(4);
5376         } else if (idx == 3) {
5377             /* "cwd" */
5378             dirent->d_ino = pmkino(idx, pslot, PR_PATH);
5379             (void) strcpy(dirent->d_name, "cwd");
5380             reclen = DIRENT64_RECLEN(3);
5381         } else if (idx < 4 + fddirsize) {
5382             /*
5383             * In this case, we have one of the file descriptors.
5384             */
5385             fd = idx - 4;
5386             if (fip->fi_list[fd].uf_file == NULL)

```

```

5383         continue;
5384         dirent->d_ino = pmkino(idx, pslot, PR_PATH);
5385         (void) pr_u32tos(fd, dirent->d_name, PLNSIZ+1);
5386         reclen = DIRENT64_RECLEN(PLNSIZ);
5387     } else if (idx < 4 + fddirsize + objdirsize) {
5388         if (fip != NULL) {
5389             mutex_exit(&fip->fi_lock);
5390             fip = NULL;
5391         }
5392     }
5393     /*
5394     * We drop p_lock before grabbing the address space lock
5395     * in order to avoid a deadlock with the clock thread.
5396     * The process will not disappear and its address space
5397     * will not change because it is marked P_PR_LOCK.
5398     */
5399     if (as == NULL) {
5400         as = p->p_as;
5401         AS_LOCK_ENTER(as, RW_WRITER);
5402     }
5403
5404     if (as->a_updatedir) {
5405         rebuild_objdir(as);
5406         objdirsize = as->a_sizedir;
5407     }
5408
5409     obj = idx - 4 - fddirsize;
5410     if ((vp = obj_entry(as, obj)) == NULL)
5411         continue;
5412     vattr.va_mask = AT_FSID|AT_NODEID;
5413     if (VOP_GETATTR(vp, &vattr, 0, CRED(), NULL) != 0)
5414         continue;
5415     if (vp == p->p_exec)
5416         (void) strcpy(dirent->d_name, "a.out");
5417     else
5418         pr_object_name(dirent->d_name, vp, &vattr);
5419     dirent->d_ino = pmkino(idx, pslot, PR_PATH);
5420     reclen = DIRENT64_RECLEN(strlen(dirent->d_name));
5421 } else {
5422     break;
5423 }
5424
5425 dirent->d_off = uiop->uio_offset + PRSDSIZE;
5426 dirent->d_reclen = (ushort_t)reclen;
5427 if (reclen > uiop->uio_resid) {
5428     /*
5429     * Error if no entries have been returned yet.
5430     */
5431     if (uiop->uio_resid == oresid)
5432         error = EINVAL;
5433     break;
5434 }
5435 /*
5436 * Drop the address space lock to do the uiomove().
5437 */
5438 if (as != NULL)
5439     AS_LOCK_EXIT(as);
5440
5441 error = uiomove((caddr_t)dirent, reclen, UIO_READ, uiop);
5442 if (as != NULL)
5443     AS_LOCK_ENTER(as, RW_WRITER);
5444
5445 if (error)
5446     break;
5447 }

```

```

5449     if (error == 0 && eofp)
5450         *eofp = (uiop->uio_offset >= (fddirsize + 2) * PRSDSIZE);
5451
5452     if (fip != NULL)
5453         mutex_exit(&fip->fi_lock);
5454     if (as != NULL)
5455         AS_LOCK_EXIT(as);
5456     mutex_enter(&p->p_lock);
5457     prunlock(pnp);
5458     return (error);
5459 }
5460
5461 static int
5462 pr_readdir_tmpldir(prnode_t *pnp, uiop_t *uiop, int *eofp)
5463 {
5464     proc_t *p;
5465     int pslot, tslot;
5466     gfs_readdir_state_t gstate;
5467     int error, eof = 0;
5468     offset_t n;
5469
5470     ASSERT(pnp->pr_type == PR_TMPLDIR);
5471
5472     if ((error = prlock(pnp, ZNO)) != 0)
5473         return (error);
5474     p = pnp->pr_common->prc_proc;
5475     pslot = pnp->pr_common->prc_slot;
5476     tslot = pnp->pr_common->prc_tslot;
5477     mutex_exit(&p->p_lock);
5478
5479     if ((error = gfs_readdir_init(&gstate, PRDIRSIZE, PRSDSIZE, uiop,
5480 pmkino(tslot, pslot, PR_LWPDIR),
5481 pmkino(tslot, pslot, PR_TMPLDIR), 0)) != 0) {
5482         mutex_enter(&p->p_lock);
5483         prunlock(pnp);
5484         return (error);
5485     }
5486
5487     while ((error = gfs_readdir_pred(&gstate, uiop, &n)) == 0) {
5488         /*
5489         * Check for an active template. Reading a directory's
5490         * contents is already racy, so we don't bother taking
5491         * any locks.
5492         */
5493         while (n < ct_ntypes &&
5494 pnp->pr_common->prc_thread->t_lwp->lwp_ct_active[n] == NULL)
5495             n++;
5496         /*
5497         * Stop when all types have been reported.
5498         */
5499         if (n >= ct_ntypes) {
5500             eof = 1;
5501             break;
5502         }
5503         /*
5504         * The pmkino invocation below will need to be updated
5505         * when we create our fifth contract type.
5506         */
5507         ASSERT(ct_ntypes <= 4);
5508         error = gfs_readdir_emit(&gstate, uiop, n,
5509 pmkino((tslot << 2) | n, pslot, PR_TMPL),
5510 ct_types[n]->ct_type_name, 0);
5511         if (error)
5512             break;
5513     }

```

```

5515     mutex_enter(&p->p_lock);
5516     prunlock(pnp);

5518     return (gfs_readdir_fini(&gstate, error, eofp, eof));
5519 }

5521 static int
5522 pr_readdir_ctdir(prnode_t *pnp, uio_t *uiop, int *eofp)
5523 {
5524     proc_t *p;
5525     int pslot;
5526     gfs_readdir_state_t gstate;
5527     int error, eof = 0;
5528     offset_t n;
5529     uint64_t zid;

5531     ASSERT(pnp->pr_type == PR_CTDIR);

5533     if ((error = prlock(pnp, ZNO)) != 0)
5534         return (error);
5535     p = pnp->pr_common->prc_proc;
5536     pslot = p->p_slot;
5537     mutex_exit(&p->p_lock);

5539     if ((error = gfs_readdir_init(&gstate, PRDIRSIZE, PRSDSIZE, uiop,
5540 pmkino(0, pslot, PR_PIDDIR), pmkino(0, pslot, PR_CTDIR), 0)) != 0) {
5541         mutex_enter(&p->p_lock);
5542         prunlock(pnp);
5543         return (error);
5544     }

5546     zid = VTOZONE(pnp->pr_vnode)->zone_uniqid;
5547     while ((error = gfs_readdir_pred(&gstate, uiop, &n)) == 0) {
5548         id_t next = contract_plookup(p, n, zid);
5549         if (next == -1) {
5550             eof = 1;
5551             break;
5552         }
5553         error = gfs_readdir_emitn(&gstate, uiop, next,
5554 pmkino(next, pslot, PR_CT), next);
5555         if (error)
5556             break;
5557     }

5559     mutex_enter(&p->p_lock);
5560     prunlock(pnp);

5562     return (gfs_readdir_fini(&gstate, error, eofp, eof));
5563 }

5565 /* ARGSUSED */
5566 static int
5567 prfsync(vnode_t *vp, int syncflag, cred_t *cr, caller_context_t *ct)
5568 {
5569     return (0);
5570 }

5572 /*
5573  * Utility: remove a /proc vnode from a linked list, threaded through pr_next.
5574  */
5575 static void
5576 pr_list_unlink(vnode_t *pvp, vnode_t **listp)
5577 {
5578     vnode_t *vp;
5579     prnode_t *pnp;

```

```

5581     while ((vp = *listp) != NULL) {
5582         pnp = VTOP(vp);
5583         if (vp == pvp) {
5584             *listp = pnp->pr_next;
5585             pnp->pr_next = NULL;
5586             break;
5587         }
5588         listp = &pnp->pr_next;
5589     }
5590 }

5592 /* ARGSUSED */
5593 static void
5594 prinactive(vnode_t *vp, cred_t *cr, caller_context_t *ct)
5595 {
5596     prnode_t *pnp = VTOP(vp);
5597     prnodetype_t type = pnp->pr_type;
5598     proc_t *p;
5599     vnode_t *dp;
5600     vnode_t *ovp = NULL;
5601     prnode_t *opnp = NULL;

5603     switch (type) {
5604     case PR_OBJECT:
5605     case PR_FD:
5606     case PR_SELF:
5607     case PR_PATH:
5608         /* These are not linked into the usual lists */
5609         ASSERT(vp->v_count == 1);
5610         if ((dp = pnp->pr_parent) != NULL)
5611             VN_RELE(dp);
5612         prfreenode(pnp);
5613         return;
5614     default:
5615         break;
5616     }

5618     mutex_enter(&pr_pidlock);
5619     if (pnp->pr_pcommon == NULL)
5620         p = NULL;
5621     else if ((p = pnp->pr_pcommon->prc_proc) != NULL)
5622         mutex_enter(&p->p_lock);
5623     mutex_enter(&vp->v_lock);

5625     if (type == PR_PROCDIR || vp->v_count > 1) {
5626         vp->v_count--;
5627         mutex_exit(&vp->v_lock);
5628         if (p != NULL)
5629             mutex_exit(&p->p_lock);
5630         mutex_exit(&pr_pidlock);
5631         return;
5632     }

5634     if ((dp = pnp->pr_parent) != NULL) {
5635         prnode_t *dpnp;

5637         switch (type) {
5638         case PR_PIDFILE:
5639         case PR_LWPIDFILE:
5640         case PR_OPAGEDATA:
5641             break;
5642         default:
5643             dpnp = VTOP(dp);
5644             mutex_enter(&dpnp->pr_mutex);
5645             if (dpnp->pr_files != NULL &&
5646                 dpnp->pr_files[pnp->pr_index] == vp)

```

```

5647         dpnp->pr_files[pnp->pr_index] = NULL;
5648         mutex_exit(&dpnp->pr_mutex);
5649         break;
5650     }
5651     pnp->pr_parent = NULL;
5652 }
5653
5654 ASSERT(vp->v_count == 1);
5655
5656 /*
5657  * If we allocated an old /proc/pid node, free it too.
5658  */
5659 if (pnp->pr_pidfile != NULL) {
5660     ASSERT(type == PR_PIDDIR);
5661     ovp = pnp->pr_pidfile;
5662     opnp = VTOP(ovp);
5663     ASSERT(opnp->pr_type == PR_PIDFILE);
5664     pnp->pr_pidfile = NULL;
5665 }
5666
5667 mutex_exit(&pr_pidlock);
5668
5669 if (p != NULL) {
5670     /*
5671      * Remove the vnodes from the lists of
5672      * /proc vnodes for the process.
5673      */
5674     int slot;
5675
5676     switch (type) {
5677     case PR_PIDDIR:
5678         pr_list_unlink(vp, &p->p_trace);
5679         break;
5680     case PR_LWPIDDIR:
5681         if ((slot = pnp->pr_common->prc_tslot) != -1) {
5682             lwpent_t *lep = p->p_lwpdir[slot].ld_entry;
5683             pr_list_unlink(vp, &lep->le_trace);
5684         }
5685         break;
5686     default:
5687         pr_list_unlink(vp, &p->p_plist);
5688         break;
5689     }
5690     if (ovp != NULL)
5691         pr_list_unlink(ovp, &p->p_plist);
5692     mutex_exit(&p->p_lock);
5693 }
5694
5695 mutex_exit(&vp->v_lock);
5696
5697 if (type == PR_CT && pnp->pr_contract != NULL) {
5698     contract_rele(pnp->pr_contract);
5699     pnp->pr_contract = NULL;
5700 }
5701
5702 if (opnp != NULL)
5703     prfreenode(opnp);
5704 prfreenode(pnp);
5705 if (dp != NULL) {
5706     VN_RELE(dp);
5707 }
5708 }
5709
5710 /* ARGSUSED */
5711 static int
5712 prseek(vnode_t *vp, offset_t ooff, offset_t *noffp, caller_context_t *ct)

```

```

5713 {
5714     return (0);
5715 }
5716
5717 /*
5718  * We use the p_execdir member of proc_t to expand the %d token in core file
5719  * paths (the directory path for the executable that dumped core; see
5720  * coreadm(1M) for details). We'd like gcore(1) to be able to expand %d in
5721  * the same way as core dumping from the kernel, but there's no convenient
5722  * and comprehensible way to export the path name for p_execdir. To solve
5723  * this, we try to find the actual path to the executable that was used. In
5724  * pr_lookup_pathdir(), we mark the a.out path name vnode with the PR_AOUT
5725  * flag, and use that here to indicate that more work is needed beyond the
5726  * call to vnodetopath().
5727  */
5728 static int
5729 prreadlink_lookup(prnode_t *pnp, char *buf, size_t size, cred_t *cr)
5730 {
5731     proc_t *p;
5732     vnode_t *vp, *execvp, *vrootp;
5733     int ret;
5734     size_t len;
5735     dirent64_t *dp;
5736     size_t dlen = DIRENT64_RECLEN(MAXPATHLEN);
5737     char *dbuf;
5738
5739     p = curproc;
5740     mutex_enter(&p->p_lock);
5741     if ((vrootp = PTOU(p)->u_rdir) == NULL)
5742         vrootp = rootdir;
5743     VN_HOLD(vrootp);
5744     mutex_exit(&p->p_lock);
5745
5746     ret = vnodetopath(vrootp, pnp->pr_realvp, buf, size, cr);
5747
5748     /*
5749      * If PR_AOUT isn't set, then we looked up the path for the vnode;
5750      * otherwise, we looked up the path for (what we believe to be) the
5751      * containing directory.
5752      */
5753     if ((pnp->pr_flags & PR_AOUT) == 0) {
5754         VN_RELE(vrootp);
5755         return (ret);
5756     }
5757
5758     /*
5759      * Fail if there's a problem locking the process. This will only
5760      * occur if the process is changing so the information we would
5761      * report would already be invalid.
5762      */
5763     if (prlock(pnp, ZNO) != 0) {
5764         VN_RELE(vrootp);
5765         return (EIO);
5766     }
5767
5768     p = pnp->pr_common->prc_proc;
5769     mutex_exit(&p->p_lock);
5770
5771     execvp = p->p_exec;
5772     VN_HOLD(execvp);
5773
5774     /*
5775      * If our initial lookup of the directory failed, fall back to
5776      * the path name information for p_exec.
5777      */
5778     if (ret != 0) {

```

```

5779         mutex_enter(&p->p_lock);
5780         prunlock(pnp);
5781         ret = vnodetopath(vrootp, execvp, buf, size, cr);
5782         VN_RELE(execvp);
5783         VN_RELE(vrootp);
5784         return (ret);
5785     }

5787     len = strlen(buf);

5789     /*
5790     * We use u_comm as a guess for the last component of the full
5791     * executable path name. If there isn't going to be enough space
5792     * we fall back to using the p_exec so that we can have _an_
5793     * answer even if it's not perfect.
5794     */
5795     if (strlen(PTOU(p)->u_comm) + len + 1 < size) {
5796         buf[len] = '/';
5797         (void) strcpy(buf + len + 1, PTOU(p)->u_comm);
5798         mutex_enter(&p->p_lock);
5799         prunlock(pnp);

5801         /*
5802         * Do a forward lookup of our u_comm guess.
5803         */
5804         if (lookupnameat(buf + len + 1, UIO_SYSSPACE, FOLLOW, NULLVPP,
5805             &vp, pnp->pr_realvp) == 0) {
5806             if (vn_compare(vp, execvp)) {
5807                 VN_RELE(vp);
5808                 VN_RELE(execvp);
5809                 VN_RELE(vrootp);
5810                 return (0);
5811             }

5813             VN_RELE(vp);
5814         }
5815     } else {
5816         mutex_enter(&p->p_lock);
5817         prunlock(pnp);
5818     }

5820     dbuf = kmem_alloc(dlen, KM_SLEEP);

5822     /*
5823     * Try to find a matching vnode by iterating through the directory's
5824     * entries. If that fails, fall back to the path information for
5825     * p_exec.
5826     */
5827     if ((ret = dirfindvp(vrootp, pnp->pr_realvp, execvp, cr, dbuf,
5828         dlen, &dp) == 0 && strlen(dp->d_name) + len + 1 < size) {
5829         buf[len] = '/';
5830         (void) strcpy(buf + len + 1, dp->d_name);
5831     } else {
5832         ret = vnodetopath(vrootp, execvp, buf, size, cr);
5833     }

5835     kmem_free(dbuf, dlen);
5836     VN_RELE(execvp);
5837     VN_RELE(vrootp);

5839     return (ret);
5840 }

5842 /* ARGSUSED */
5843 static int
5844 prreadlink(vnode_t *vp, uio_t *uiop, cred_t *cr, caller_context_t *ctp)

```

```

5845 {
5846     prnode_t *pnp = VTOP(vp);
5847     char *buf;
5848     int ret = EINVAL;
5849     char idbuf[16];
5850     int length, rlength;
5851     contract_t *ct;

5853     switch (pnp->pr_type) {
5854     case PR_SELF:
5855         (void) snprintf(idbuf, sizeof(idbuf), "%d", curproc->p_pid);
5856         ret = uiomove(idbuf, strlen(idbuf), UIO_READ, uiop);
5857         break;
5858     case PR_OBJECT:
5859     case PR_FD:
5860     case PR_CURDIR:
5861     case PR_ROOTDIR:
5862         if (pnp->pr_realvp->v_type == VDIR)
5863             ret = 0;
5864         break;
5865     case PR_PATH:
5866         buf = kmem_alloc(MAXPATHLEN, KM_SLEEP);

5868         if ((ret = prreadlink_lookup(pnp, buf, MAXPATHLEN, cr)) == 0)
5869             ret = uiomove(buf, strlen(buf), UIO_READ, uiop);

5871         kmem_free(buf, MAXPATHLEN);
5872         break;
5873     case PR_CT:
5874         ASSERT(pnp->pr_contract != NULL);
5875         ct = pnp->pr_contract;
5876         length = sizeof(CTFS_ROOT "/") + sizeof(idbuf) +
5877             strlen(ct->ct_type->ct_type_name);
5878         buf = kmem_alloc(length, KM_SLEEP);
5879         rlength = snprintf(buf, length, CTFS_ROOT "%s/%d",
5880             ct->ct_type->ct_type_name, ct->ct_id);
5881         ASSERT(rlength < length);
5882         ret = uiomove(buf, rlength, UIO_READ, uiop);
5883         kmem_free(buf, length);
5884         break;
5885     default:
5886         break;
5887     }

5889     return (ret);
5890 }

5892 /*ARGSUSED2*/
5893 static int
5894 prcmp(vnode_t *vp1, vnode_t *vp2, caller_context_t *ctp)
5895 {
5896     prnode_t *pp1, *pp2;

5898     if (vp1 == vp2)
5899         return (1);

5901     if (!vn_matchchops(vp1, prvnops) || !vn_matchchops(vp2, prvnops))
5902         return (0);

5904     pp1 = VTOP(vp1);
5905     pp2 = VTOP(vp2);

5907     if (pp1->pr_type != pp2->pr_type)
5908         return (0);
5909     if (pp1->pr_type == PR_PROCDIR)
5910         return (1);

```

```

5911     if (pp1->pr_ino || pp2->pr_ino)
5912         return (pp2->pr_ino == pp1->pr_ino);

5914     if (pp1->pr_common == NULL || pp2->pr_common == NULL)
5915         return (0);

5917     return (pp1->pr_common->prc_slot == pp2->pr_common->prc_slot &&
5918         pp1->pr_common->prc_tslot == pp2->pr_common->prc_tslot);
5919 }

5921 static int
5922 prrealvp(vnode_t *vp, vnode_t **vpp, caller_context_t *ct)
5923 {
5924     vnode_t *rvp;

5926     if ((rvp = VTOP(vp)->pr_realvp) != NULL) {
5927         vp = rvp;
5928         if (VOP_REALVP(vp, &rvp, ct) == 0)
5929             vp = rvp;
5930     }

5932     *vpp = vp;
5933     return (0);
5934 }

5936 /*
5937  * Return the answer requested to poll().
5938  * POLLIN, POLLRDNORM, and POLLOUT are recognized as in fs_poll().
5939  * In addition, these have special meaning for /proc files:
5940  *   POLLPRI      process or lwp stopped on an event of interest
5941  *   POLLERR      /proc file descriptor is invalid
5942  *   POLLHUP      process or lwp has terminated
5943  */
5944 /*ARGSUSED5*/
5945 static int
5946 prpoll(vnode_t *vp, short events, int anyyet, short *reventsp,
5947     pollhead_t **phpp, caller_context_t *ct)
5948 {
5949     prnode_t *pnp = VTOP(vp);
5950     prcommon_t *pcp = pnp->pr_common;
5951     pollhead_t *php = &pcp->prc_pollhead;
5952     proc_t *p;
5953     short revents;
5954     int error;
5955     int lockstate;

5957     ASSERT(pnp->pr_type < PR_NFILES);

5959     /*
5960      * Support for old /proc interface.
5961      */
5962     if (pnp->pr_pidfile != NULL) {
5963         vp = pnp->pr_pidfile;
5964         pnp = VTOP(vp);
5965         ASSERT(pnp->pr_type == PR_PIDFILE);
5966         ASSERT(pnp->pr_common == pcp);
5967     }

5969     *reventsp = revents = 0;
5970     *phpp = (pollhead_t *)NULL;

5972     if (vp->v_type == VDIR) {
5973         *reventsp |= POLLNVAL;
5974         return (0);
5975     }

```

```

5977     /* avoid deadlock with prnotify() */
5978     if (pollunlock(&lockstate) != 0) {
5979         *reventsp = POLLNVAL;
5980         return (0);
5981     }

5983     if ((error = prlock(pnp, ZNO)) != 0) {
5984         pollrelock(lockstate);
5985         switch (error) {
5986             case ENOENT: /* process or lwp died */
5987                 *reventsp = POLLHUP;
5988                 error = 0;
5989                 break;
5990             case EAGAIN: /* invalidated */
5991                 *reventsp = POLLERR;
5992                 error = 0;
5993                 break;
5994         }
5995         return (error);
5996     }

5998     /*
5999      * We have the process marked locked (P_PR_LOCK) and we are holding
6000      * its p->p_lock. We want to unmark the process but retain
6001      * exclusive control w.r.t. other /proc controlling processes
6002      * before reacquiring the polling locks.
6003      *
6004      * prunmark() does this for us. It unmarks the process
6005      * but retains p->p_lock so we still have exclusive control.
6006      * We will drop p->p_lock at the end to relinquish control.
6007      *
6008      * We cannot call prunlock() at the end to relinquish control
6009      * because prunlock(), like prunmark(), may drop and reacquire
6010      * p->p_lock and that would lead to a lock order violation
6011      * w.r.t. the polling locks we are about to reacquire.
6012      */
6013     p = pcp->prc_proc;
6014     ASSERT(p != NULL);
6015     prunmark(p);

6017     pollrelock(lockstate); /* reacquire dropped poll locks */

6019     if ((p->p_flag & SSYS) || p->p_as == &kas)
6020         revents = POLLNVAL;
6021     else {
6022         short ev;

6024         if ((ev = (events & (POLLIN|POLLRDNORM))) != 0)
6025             revents |= ev;
6026         /*
6027          * POLLWRNORM (same as POLLOUT) really should not be
6028          * used to indicate that the process or lwp stopped.
6029          * However, USL chose to use POLLWRNORM rather than
6030          * POLLPRI to indicate this, so we just accept either
6031          * requested event to indicate stopped. (grr...)
6032          */
6033         if ((ev = (events & (POLLPRI|POLLOUT|POLLWRNORM))) != 0) {
6034             kthread_t *t;

6036             if (pcp->prc_flags & PRC_LWP) {
6037                 t = pcp->prc_thread;
6038                 ASSERT(t != NULL);
6039                 thread_lock(t);
6040             } else {
6041                 t = prchoose(p); /* returns locked t */
6042                 ASSERT(t != NULL);

```

```

6043     }
6044     if (ISTOPPED(t) || VSTOPPED(t))
6045         revents |= ev;
6046     thread_unlock(t);
6047 }
6048 }
6049
6051 *reventsp = revents;
6052 if ((!anyyet && revents == 0) || (events & POLLET)) {
6053     /*
6054     * Arrange to wake up the polling lwp when
6055     * the target process/lwp stops or terminates
6056     * or when the file descriptor becomes invalid.
6057     */
6058     pcp->prc_flags |= PRC_POLL;
6059     *phpp = php;
6060 }
6061 mutex_exit(&p->p_lock);
6062 return (0);
6063 }
6065 /* in priocntl.c */
6066 extern int priocntl(vnode_t *, int, intptr_t, int, cred_t *, int *,
6067     caller_context_t *);
6069 /*
6070 * /proc vnode operations vector
6071 */
6072 const fs_operation_def_t pr_vnodeops_template[] = {
6073     VOPNAME_OPEN,      { .vop_open = propen },
6074     VOPNAME_CLOSE,    { .vop_close = prclose },
6075     VOPNAME_READ,     { .vop_read = prread },
6076     VOPNAME_WRITE,    { .vop_write = prwrite },
6077     VOPNAME_IOCTL,    { .vop_ioctl = priocntl },
6078     VOPNAME_GETATTR,  { .vop_getattr = prgetattr },
6079     VOPNAME_ACCESS,   { .vop_access = praccess },
6080     VOPNAME_LOOKUP,   { .vop_lookup = prlookup },
6081     VOPNAME_CREATE,   { .vop_create = prcreate },
6082     VOPNAME_READDIR,  { .vop_readdir = preaddir },
6083     VOPNAME_READLINK, { .vop_readlink = prreadlink },
6084     VOPNAME_FSYNC,    { .vop_fsync = prfsync },
6085     VOPNAME_INACTIVE, { .vop_inactive = prinactive },
6086     VOPNAME_SEEK,     { .vop_seek = prseek },
6087     VOPNAME_CMP,      { .vop_cmp = prcmp },
6088     VOPNAME_FRLOCK,   { .error = fs_error },
6089     VOPNAME_REALVP,   { .vop_realvp = prrealvp },
6090     VOPNAME_POLL,     { .vop_poll = prpoll },
6091     VOPNAME_DISPOSE,  { .error = fs_error },
6092     VOPNAME_SHRLOCK,  { .error = fs_error },
6093     NULL,             NULL
6094 };

```



```

*****
32731 Thu Jun 30 22:03:25 2016
new/usr/src/uts/common/os/cred.c
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
_____unchanged_portion_omitted_____

162 /*
163  * Initialize credentials data structures.
164  */

166 void
167 cred_init(void)
168 {
169     priv_init();

171     crsize = sizeof (cred_t);

173     if (get_c2audit_load() > 0) {
174 #ifdef _LP64
175         /* assure audit context is 64-bit aligned */
176         audoff = (crsize +
177                 sizeof (int64_t) - 1) & ~(sizeof (int64_t) - 1);
178 #else /* _LP64 */
179         audoff = crsize;
180 #endif /* _LP64 */
181         crsize = audoff + sizeof (auditinfo_addr_t);
182         crsize = (crsize + sizeof (int) - 1) & ~(sizeof (int) - 1);
183     }

185     cred_cache = kmem_cache_create("cred_cache", crsize, 0,
186     NULL, NULL, NULL, NULL, NULL, 0);

188     /*
189     * dummycr is used to copy initial state for creds.
190     */
191     dummycr = cralloc();
192     bzero(dummycr, crsize);
193     dummycr->cr_ref = 1;
194     dummycr->cr_uid = (uid_t)-1;
195     dummycr->cr_gid = (gid_t)-1;
196     dummycr->cr_ruid = (uid_t)-1;
197     dummycr->cr_rgid = (gid_t)-1;
198     dummycr->cr_suid = (uid_t)-1;
199     dummycr->cr_sgid = (gid_t)-1;

202     /*
203     * kcred is used by anything that needs all privileges; it's
204     * also the template used for crget as it has all the compatible
205     * sets filled in.
206     */
207     kcred = cralloc();

209     bzero(kcred, crsize);
210     kcred->cr_ref = 1;

212     /* kcred is never freed, so we don't need zone_cred_hold here */
213     kcred->cr_zone = &zone0;

215     priv_fillset(&CR_LPRIV(kcred));

```

```

216     CR_IPRIV(kcred) = *priv_basic;

218     priv_addset(&CR_IPRIV(kcred), PRIV_PROC_SECFLAGS);

220 #endif /* ! codereview */
221 /* Not a basic privilege, if chown is not restricted add it to IO */
222 if (!rstchown)
223     priv_addset(&CR_IPRIV(kcred), PRIV_FILE_CHOWN_SELF);

225     /* Basic privilege, if link is restricted remove it from IO */
226     if (rstlink)
227         priv_delset(&CR_IPRIV(kcred), PRIV_FILE_LINK_ANY);

229     CR_EPRIV(kcred) = CR_PPRIV(kcred) = CR_IPRIV(kcred);

231     CR_FLAGS(kcred) = NET_MAC_AWARE;

233     /*
234     * Set up credentials of p0.
235     */
236     ttproc(curthread)->p_cred = kcred;
237     curthread->t_cred = kcred;

239     ucredsize = UCRED_SIZE;

241     mutex_init(&ephemeral_zone_mutex, NULL, MUTEX_DEFAULT, NULL);
242     zone_key_create(&ephemeral_zone_key, NULL, NULL, destroy_ephemeral_zsd);
243 }

245 /*
246  * Allocate (nearly) uninitialized cred_t.
247  */
248 static cred_t *
249 cralloc_flags(int flgs)
250 {
251     cred_t *cr = kmem_cache_alloc(cred_cache, flgs);

253     if (cr == NULL)
254         return (NULL);

256     cr->cr_ref = 1;          /* So we can crfree() */
257     cr->cr_zone = NULL;
258     cr->cr_label = NULL;
259     cr->cr_ksid = NULL;
260     cr->cr_klpd = NULL;
261     cr->cr_grps = NULL;
262     return (cr);
263 }

265 cred_t *
266 cralloc(void)
267 {
268     return (cralloc_flags(KM_SLEEP));
269 }

271 /*
272  * As cralloc but prepared for ksid change (if appropriate).
273  */
274 cred_t *
275 cralloc_ksid(void)
276 {
277     cred_t *cr = cralloc();
278     if (hasephids)
279         cr->cr_ksid = kcrsid_alloc();
280     return (cr);
281 }

```

```

283 /*
284 * Allocate a initialized cred structure and crhold() it.
285 * Initialized means: all ids 0, group count 0, L=Full, E=P=I=IO
286 */
287 cred_t *
288 crget(void)
289 {
290     cred_t *cr = kmem_cache_alloc(cred_cache, KM_SLEEP);

292     bcopy(kcred, cr, crsize);
293     cr->cr_ref = 1;
294     zone_cred_hold(cr->cr_zone);
295     if (cr->cr_label)
296         label_hold(cr->cr_label);
297     ASSERT(cr->cr_klpd == NULL);
298     ASSERT(cr->cr_grps == NULL);
299     return (cr);
300 }

302 /*
303 * Broadcast the cred to all the threads in the process.
304 * The current thread's credentials can be set right away, but other
305 * threads must wait until the start of the next system call or trap.
306 * This avoids changing the cred in the middle of a system call.
307 *
308 * The cred has already been held for the process and the thread (2 holds),
309 * and p->p_cred set.
310 *
311 * p->p_crlock shouldn't be held here, since p_lock must be acquired.
312 */
313 void
314 crset(proc_t *p, cred_t *cr)
315 {
316     kthread_id_t    t;
317     kthread_id_t    first;
318     cred_t *oldcr;

320     ASSERT(p == curproc); /* assumes p_lwpcnt can't change */

322     /*
323      * DTrace accesses t_cred in probe context. t_cred must always be
324      * either NULL, or point to a valid, allocated cred structure.
325      */
326     t = curthread;
327     oldcr = t->t_cred;
328     t->t_cred = cr; /* the cred is held by caller for this thread */
329     crfree(oldcr); /* free the old cred for the thread */

331     /*
332      * Broadcast to other threads, if any.
333      */
334     if (p->p_lwpcnt > 1) {
335         mutex_enter(&p->p_lock); /* to keep thread list safe */
336         first = curthread;
337         for (t = first->t_forw; t != first; t = t->t_forw)
338             t->t_pre_sys = 1; /* so syscall will get new cred */
339         mutex_exit(&p->p_lock);
340     }
341 }

343 /*
344 * Put a hold on a cred structure.
345 */
346 void
347 crhold(cred_t *cr)

```

```

348 {
349     ASSERT(cr->cr_ref != 0xdeadbeef && cr->cr_ref != 0);
350     atomic_inc_32(&cr->cr_ref);
351 }

353 /*
354 * Release previous hold on a cred structure. Free it if refcnt == 0.
355 * If cred uses label different from zone label, free it.
356 */
357 void
358 crfree(cred_t *cr)
359 {
360     ASSERT(cr->cr_ref != 0xdeadbeef && cr->cr_ref != 0);
361     if (atomic_dec_32_nv(&cr->cr_ref) == 0) {
362         ASSERT(cr != kcred);
363         if (cr->cr_label)
364             label_rele(cr->cr_label);
365         if (cr->cr_klpd)
366             crklpd_rele(cr->cr_klpd);
367         if (cr->cr_zone)
368             zone_cred_rele(cr->cr_zone);
369         if (cr->cr_ksid)
370             kcrsid_rele(cr->cr_ksid);
371         if (cr->cr_grps)
372             crgrprele(cr->cr_grps);

374         kmem_cache_free(cred_cache, cr);
375     }
376 }

378 /*
379 * Copy a cred structure to a new one and free the old one.
380 * The new cred will have two references. One for the calling process,
381 * and one for the thread.
382 */
383 cred_t *
384 crcopy(cred_t *cr)
385 {
386     cred_t *newcr;

388     newcr = cralloc();
389     bcopy(cr, newcr, crsize);
390     if (newcr->cr_zone)
391         zone_cred_hold(newcr->cr_zone);
392     if (newcr->cr_label)
393         label_hold(newcr->cr_label);
394     if (newcr->cr_ksid)
395         kcrsid_hold(newcr->cr_ksid);
396     if (newcr->cr_klpd)
397         crklpd_hold(newcr->cr_klpd);
398     if (newcr->cr_grps)
399         crgrphold(newcr->cr_grps);
400     crfree(cr);
401     newcr->cr_ref = 2; /* caller gets two references */
402     return (newcr);
403 }

405 /*
406 * Copy a cred structure to a new one and free the old one.
407 * The new cred will have two references. One for the calling process,
408 * and one for the thread.
409 * This variation on crcopy uses a pre-allocated structure for the
410 * "new" cred.
411 */
412 void
413 crcopy_to(cred_t *oldcr, cred_t *newcr)

```

```

414 {
415     credsid_t *nkcr = newcr->cr_ksid;

417     bcopy(oldcr, newcr, crsize);
418     if (newcr->cr_zone)
419         zone_cred_hold(newcr->cr_zone);
420     if (newcr->cr_label)
421         label_hold(newcr->cr_label);
422     if (newcr->cr_klpd)
423         crklpd_hold(newcr->cr_klpd);
424     if (newcr->cr_grps)
425         crgrphold(newcr->cr_grps);
426     if (nkcr) {
427         newcr->cr_ksid = nkcr;
428         kcrsidcopy_to(oldcr->cr_ksid, newcr->cr_ksid);
429     } else if (newcr->cr_ksid)
430         kcrsid_hold(newcr->cr_ksid);
431     crfree(oldcr);
432     newcr->cr_ref = 2;          /* caller gets two references */
433 }

435 /*
436  * Dup a cred struct to a new held one.
437  * The old cred is not freed.
438  */
439 static cred_t *
440 crdup_flags(const cred_t *cr, int flgs)
441 {
442     cred_t *newcr;

444     newcr = cralloc_flags(flgs);

446     if (newcr == NULL)
447         return (NULL);

449     bcopy(cr, newcr, crsize);
450     if (newcr->cr_zone)
451         zone_cred_hold(newcr->cr_zone);
452     if (newcr->cr_label)
453         label_hold(newcr->cr_label);
454     if (newcr->cr_klpd)
455         crklpd_hold(newcr->cr_klpd);
456     if (newcr->cr_ksid)
457         kcrsid_hold(newcr->cr_ksid);
458     if (newcr->cr_grps)
459         crgrphold(newcr->cr_grps);
460     newcr->cr_ref = 1;
461     return (newcr);
462 }

464 cred_t *
465 crdup(cred_t *cr)
466 {
467     return (crdup_flags(cr, KM_SLEEP));
468 }

470 /*
471  * Dup a cred struct to a new held one.
472  * The old cred is not freed.
473  * This variation on crdup uses a pre-allocated structure for the
474  * "new" cred.
475  */
476 void
477 crdup_to(cred_t *oldcr, cred_t *newcr)
478 {
479     credsid_t *nkcr = newcr->cr_ksid;

```

```

481     bcopy(oldcr, newcr, crsize);
482     if (newcr->cr_zone)
483         zone_cred_hold(newcr->cr_zone);
484     if (newcr->cr_label)
485         label_hold(newcr->cr_label);
486     if (newcr->cr_klpd)
487         crklpd_hold(newcr->cr_klpd);
488     if (newcr->cr_grps)
489         crgrphold(newcr->cr_grps);
490     if (nkcr) {
491         newcr->cr_ksid = nkcr;
492         kcrsidcopy_to(oldcr->cr_ksid, newcr->cr_ksid);
493     } else if (newcr->cr_ksid)
494         kcrsid_hold(newcr->cr_ksid);
495     newcr->cr_ref = 1;
496 }

498 /*
499  * Return the (held) credentials for the current running process.
500  */
501 cred_t *
502 crgetcred(void)
503 {
504     cred_t *cr;
505     proc_t *p;

507     p = ttoproc(curthread);
508     mutex_enter(&p->p_crlock);
509     crhold(cr = p->p_cred);
510     mutex_exit(&p->p_crlock);
511     return (cr);
512 }

514 /*
515  * Backward compatibility check for suser().
516  * Accounting flag is now set in the policy functions; auditing is
517  * done through use of privilege in the audit trail.
518  */
519 int
520 suser(cred_t *cr)
521 {
522     return (PRIV_POLICY(cr, PRIV_SYS_USUSER_COMPAT, B_FALSE, EPERM, NULL)
523         == 0);
524 }

526 /*
527  * Determine whether the supplied group id is a member of the group
528  * described by the supplied credentials.
529  */
530 int
531 groupmember(gid_t gid, const cred_t *cr)
532 {
533     if (gid == cr->cr_gid)
534         return (1);
535     return (supgroupmember(gid, cr));
536 }

538 /*
539  * As groupmember but only check against the supplemental groups.
540  */
541 int
542 supgroupmember(gid_t gid, const cred_t *cr)
543 {
544     int hi, lo;
545     credgrp_t *grps = cr->cr_grps;

```

```

546     const gid_t *gp, *endgp;
547
548     if (grps == NULL)
549         return (0);
550
551     /* For a small number of groups, use sequential search. */
552     if (grps->crq_ngroups <= BIN_GROUP_SEARCH_CUTOFF) {
553         endgp = &grps->crq_groups[grps->crq_ngroups];
554         for (gp = grps->crq_groups; gp < endgp; gp++)
555             if (*gp == gid)
556                 return (1);
557         return (0);
558     }
559
560     /* We use binary search when we have many groups. */
561     lo = 0;
562     hi = grps->crq_ngroups - 1;
563     gp = grps->crq_groups;
564
565     do {
566         int m = (lo + hi) / 2;
567
568         if (gid > gp[m])
569             lo = m + 1;
570         else if (gid < gp[m])
571             hi = m - 1;
572         else
573             return (1);
574     } while (lo <= hi);
575
576     return (0);
577 }
578
579 /*
580 * This function is called to check whether the credentials set
581 * "scrp" has permission to act on credentials set "tcrp". It enforces the
582 * permission requirements needed to send a signal to a process.
583 * The same requirements are imposed by other system calls, however.
584 *
585 * The rules are:
586 * (1) if the credentials are the same, the check succeeds
587 * (2) if the zone ids don't match, and scrp is not in the global zone or
588 * does not have the PRIV_PROC_ZONE privilege, the check fails
589 * (3) if the real or effective user id of scrp matches the real or saved
590 * user id of tcrp or scrp has the PRIV_PROC_OWNER privilege, the check
591 * succeeds
592 * (4) otherwise, the check fails
593 */
594 int
595 hasprocperm(const cred_t *tcrp, const cred_t *scrp)
596 {
597     if (scrp == tcrp)
598         return (1);
599     if (scrp->cr_zone != tcrp->cr_zone &&
600         (scrp->cr_zone != global_zone ||
601          secpolicy_proc_zone(scrp) != 0))
602         return (0);
603     if (scrp->cr_uid == tcrp->cr_ruid ||
604         scrp->cr_ruid == tcrp->cr_ruid ||
605         scrp->cr_uid == tcrp->cr_suid ||
606         scrp->cr_ruid == tcrp->cr_suid ||
607         !PRIV_POLICY(scrp, PRIV_PROC_OWNER, B_FALSE, EPERM, "hasprocperm"))
608         return (1);
609     return (0);
610 }

```

```

612 /*
613 * This interface replaces hasprocperm; it works like hasprocperm but
614 * additionally returns success if the proc_t's match
615 * It is the preferred interface for most uses.
616 * And it will acquire p_crlock itself, so it asserts that it shouldn't
617 * be held.
618 */
619 int
620 prochasprocperm(proc_t *tp, proc_t *sp, const cred_t *scrp)
621 {
622     int rets;
623     cred_t *tcrp;
624
625     ASSERT(MUTEX_NOT_HELD(&tp->p_crlock));
626
627     if (tp == sp)
628         return (1);
629
630     if (tp->p_sessp != sp->p_sessp && secpolicy_basic_proc(scrp) != 0)
631         return (0);
632
633     mutex_enter(&tp->p_crlock);
634     crhold(tcrp = tp->p_cred);
635     mutex_exit(&tp->p_crlock);
636     rets = hasprocperm(tcrp, scrp);
637     crfree(tcrp);
638
639     return (rets);
640 }
641
642 /*
643 * This routine is used to compare two credentials to determine if
644 * they refer to the same "user". If the pointers are equal, then
645 * they must refer to the same user. Otherwise, the contents of
646 * the credentials are compared to see whether they are equivalent.
647 *
648 * This routine returns 0 if the credentials refer to the same user,
649 * 1 if they do not.
650 */
651 int
652 crcmp(const cred_t *cr1, const cred_t *cr2)
653 {
654     credgrp_t *grp1, *grp2;
655
656     if (cr1 == cr2)
657         return (0);
658
659     if (cr1->cr_uid == cr2->cr_uid &&
660         cr1->cr_gid == cr2->cr_gid &&
661         cr1->cr_ruid == cr2->cr_ruid &&
662         cr1->cr_rgid == cr2->cr_rgid &&
663         cr1->cr_zone == cr2->cr_zone &&
664         ((grp1 = cr1->cr_grps) == (grp2 = cr2->cr_grps) ||
665          (grp1 != NULL && grp2 != NULL &&
666           grp1->crq_ngroups == grp2->crq_ngroups &&
667           bcmp(grp1->crq_groups, grp2->crq_groups,
668                grp1->crq_ngroups * sizeof (gid_t)) == 0)) {
669         return (!priv_isequalset(&CR_OEPRIV(cr1), &CR_OEPRIV(cr2)));
670     }
671     return (1);
672 }
673
674 /*
675 * Read access functions to cred_t.
676 */
677 uid_t

```

```

678 crgetuid(const cred_t *cr)
679 {
680     return (cr->cr_uid);
681 }

683 uid_t
684 crgetruid(const cred_t *cr)
685 {
686     return (cr->cr_ruid);
687 }

689 uid_t
690 crgetsuid(const cred_t *cr)
691 {
692     return (cr->cr_suid);
693 }

695 gid_t
696 crgetgid(const cred_t *cr)
697 {
698     return (cr->cr_gid);
699 }

701 gid_t
702 crgetrgid(const cred_t *cr)
703 {
704     return (cr->cr_rgid);
705 }

707 gid_t
708 crgetsgid(const cred_t *cr)
709 {
710     return (cr->cr_sgid);
711 }

713 const auditinfo_addr_t *
714 crgetauinfo(const cred_t *cr)
715 {
716     return ((const auditinfo_addr_t *)CR_AUINFO(cr));
717 }

719 auditinfo_addr_t *
720 crgetauinfo_modifiable(cred_t *cr)
721 {
722     return (CR_AUINFO(cr));
723 }

725 zoneid_t
726 crgetzoneid(const cred_t *cr)
727 {
728     return (cr->cr_zone == NULL ?
729         (cr->cr_uid == -1 ? (zoneid_t)-1 : GLOBAL_ZONEID) :
730         cr->cr_zone->zone_id);
731 }

733 projid_t
734 crgetprojid(const cred_t *cr)
735 {
736     return (cr->cr_projid);
737 }

739 zone_t *
740 crgetzone(const cred_t *cr)
741 {
742     return (cr->cr_zone);
743 }

```

```

745 struct ts_label_s *
746 crgetlabel(const cred_t *cr)
747 {
748     return (cr->cr_label ?
749         cr->cr_label :
750         (cr->cr_zone ? cr->cr_zone->zone_slab : NULL));
751 }

753 boolean_t
754 crisremote(const cred_t *cr)
755 {
756     return (REMOTE_PEER_CRED(cr));
757 }

759 #define BADUID(x, zn) ((x) != -1 && !VALID_UID((x), (zn)))
760 #define BADGID(x, zn) ((x) != -1 && !VALID_GID((x), (zn)))

762 int
763 crsetresuid(cred_t *cr, uid_t r, uid_t e, uid_t s)
764 {
765     zone_t *zone = crgetzone(cr);
767     ASSERT(cr->cr_ref <= 2);
769     if (BADUID(r, zone) || BADUID(e, zone) || BADUID(s, zone))
770         return (-1);
772     if (r != -1)
773         cr->cr_ruid = r;
774     if (e != -1)
775         cr->cr_uid = e;
776     if (s != -1)
777         cr->cr_suid = s;
779     return (0);
780 }

782 int
783 crsetresgid(cred_t *cr, gid_t r, gid_t e, gid_t s)
784 {
785     zone_t *zone = crgetzone(cr);
787     ASSERT(cr->cr_ref <= 2);
789     if (BADGID(r, zone) || BADGID(e, zone) || BADGID(s, zone))
790         return (-1);
792     if (r != -1)
793         cr->cr_rgid = r;
794     if (e != -1)
795         cr->cr_gid = e;
796     if (s != -1)
797         cr->cr_sgid = s;
799     return (0);
800 }

802 int
803 crsetugid(cred_t *cr, uid_t uid, gid_t gid)
804 {
805     zone_t *zone = crgetzone(cr);
807     ASSERT(cr->cr_ref <= 2);
809     if (!VALID_UID(uid, zone) || !VALID_GID(gid, zone))

```

```

810         return (-1);
812     cr->cr_uid = cr->cr_ruid = cr->cr_suid = uid;
813     cr->cr_gid = cr->cr_rgid = cr->cr_sgid = gid;
815     return (0);
816 }

818 static int
819 gidcmp(const void *v1, const void *v2)
820 {
821     gid_t g1 = *(gid_t *)v1;
822     gid_t g2 = *(gid_t *)v2;

824     if (g1 < g2)
825         return (-1);
826     else if (g1 > g2)
827         return (1);
828     else
829         return (0);
830 }

832 int
833 crsetgroups(cred_t *cr, int n, gid_t *grp)
834 {
835     ASSERT(cr->cr_ref <= 2);

837     if (n > ngroups_max || n < 0)
838         return (-1);

840     if (cr->cr_grps != NULL)
841         crgrprele(cr->cr_grps);

843     if (n > 0) {
844         cr->cr_grps = kmem_alloc(CREDGRPSZ(n), KM_SLEEP);
845         bcopy(grp, cr->cr_grps->crg_groups, n * sizeof (gid_t));
846         cr->cr_grps->crg_ref = 1;
847         cr->cr_grps->crg_ngroups = n;
848         qsort(cr->cr_grps->crg_groups, n, sizeof (gid_t), gidcmp);
849     } else {
850         cr->cr_grps = NULL;
851     }

853     return (0);
854 }

856 void
857 crsetprojid(cred_t *cr, projid_t projid)
858 {
859     ASSERT(projid >= 0 && projid <= MAXPROJID);
860     cr->cr_projid = projid;
861 }

863 /*
864  * This routine returns the pointer to the first element of the crg_groups
865  * array. It can move around in an implementation defined way.
866  * Note that when we have no grouplist, we return one element but the
867  * caller should never reference it.
868  */
869 const gid_t *
870 crgetgroups(const cred_t *cr)
871 {
872     return (cr->cr_grps == NULL ? &cr->cr_gid : cr->cr_grps->crg_groups);
873 }

875 int

```

```

876 crgetngroups(const cred_t *cr)
877 {
878     return (cr->cr_grps == NULL ? 0 : cr->cr_grps->crg_ngroups);
879 }

881 void
882 cred2prcred(const cred_t *cr, prcred_t *pccr)
883 {
884     pccr->pr_euid = cr->cr_uid;
885     pccr->pr_ruid = cr->cr_ruid;
886     pccr->pr_suid = cr->cr_suid;
887     pccr->pr_egid = cr->cr_gid;
888     pccr->pr_rgid = cr->cr_rgid;
889     pccr->pr_sgid = cr->cr_sgid;
890     pccr->pr_groups[0] = 0; /* in case ngroups == 0 */
891     pccr->pr_ngroups = cr->cr_grps == NULL ? 0 : cr->cr_grps->crg_ngroups;

893     if (pccr->pr_ngroups != 0)
894         bcopy(cr->cr_grps->crg_groups, pccr->pr_groups,
895             sizeof (gid_t) * pccr->pr_ngroups);
896 }

898 static int
899 cred2ucaud(const cred_t *cr, auditinfo64_addr_t *ainfo, const cred_t *rcr)
900 {
901     auditinfo_addr_t *ai;
902     au_tid_addr_t tid;

904     if (secpolicy_audit_getattr(rcr, B_TRUE) != 0)
905         return (-1);

907     ai = CR_AUINFO(cr); /* caller makes sure this is non-NULL */
908     tid = ai->ai_termid;

910     ainfo->ai_auid = ai->ai_auid;
911     ainfo->ai_mask = ai->ai_mask;
912     ainfo->ai_asid = ai->ai_asid;

914     ainfo->ai_termid.at_type = tid.at_type;
915     bcopy(&tid.at_addr, &ainfo->ai_termid.at_addr, 4 * sizeof (uint_t));

917     ainfo->ai_termid.at_port.at_major = (uint32_t)getmajor(tid.at_port);
918     ainfo->ai_termid.at_port.at_minor = (uint32_t)getminor(tid.at_port);

920     return (0);
921 }

923 void
924 cred2uclabel(const cred_t *cr, bslabel_t *labelp)
925 {
926     ts_label_t *tslp;

928     if ((tslp = crgetlabel(cr)) != NULL)
929         bcopy(&tslp->tsl_label, labelp, sizeof (bslabel_t));
930 }

932 /*
933  * Convert a credential into a "ucred". Allow the caller to specify
934  * an aligned buffer, e.g., in an mblk, so we don't have to allocate
935  * memory and copy it twice.
936  */
937 * This function may call cred2ucaud(), which calls CRED(). Since this
938 * can be called from an interrupt thread, receiver's cred (rcr) is needed
939 * to determine whether audit info should be included.
940 */
941 struct ucred_s *

```

```

942 cred2ucred(const cred_t *cr, pid_t pid, void *buf, const cred_t *rcr)
943 {
944     struct ucred_s *uc;
945     uint32_t realsz = ucredminsize(cr);
946     ts_label_t *tslp = is_system_labeled() ? crgetlabel(cr) : NULL;
947
948     /* The structure isn't always completely filled in, so zero it */
949     if (buf == NULL) {
950         uc = kmem_zalloc(realsz, KM_SLEEP);
951     } else {
952         bzero(buf, realsz);
953         uc = buf;
954     }
955     uc->uc_size = realsz;
956     uc->uc_pid = pid;
957     uc->uc_projid = cr->cr_projid;
958     uc->uc_zoneid = crgetzoneid(cr);
959
960     if (REMOTE_PEER_CRED(cr)) {
961         /*
962          * Other than label, the rest of cred info about a
963          * remote peer isn't available. Copy the label directly
964          * after the header where we generally copy the pcred.
965          * That's why we use sizeof (struct ucred_s). The other
966          * offset fields are initialized to 0.
967          */
968         uc->uc_labeloff = tslp == NULL ? 0 : sizeof (struct ucred_s);
969     } else {
970         uc->uc_credoff = UCRED_CRED_OFF;
971         uc->uc_privoff = UCRED_PRIV_OFF;
972         uc->uc_audoff = UCRED_AUD_OFF;
973         uc->uc_labeloff = tslp == NULL ? 0 : UCRED_LABEL_OFF;
974
975         cred2prcred(cr, UCRED(uc));
976         cred2prpriv(cr, UCPRIV(uc));
977
978         if (audoff == 0 || cred2ucaud(cr, UCAUD(uc), rcr) != 0)
979             uc->uc_audoff = 0;
980     }
981     if (tslp != NULL)
982         bcopy(&tslp->tsl_label, UCLABEL(uc), sizeof (bslabel_t));
983
984     return (uc);
985 }
986
987 /*
988 * Don't allocate the non-needed group entries. Note: this function
989 * must match the code in cred2ucred; they must agree about the
990 * minimal size of the ucred.
991 */
992 uint32_t
993 ucredminsize(const cred_t *cr)
994 {
995     int ndiff;
996
997     if (cr == NULL)
998         return (ucredsize);
999
1000     if (REMOTE_PEER_CRED(cr)) {
1001         if (is_system_labeled())
1002             return (sizeof (struct ucred_s) + sizeof (bslabel_t));
1003         else
1004             return (sizeof (struct ucred_s));
1005     }
1006
1007     if (cr->cr_grps == NULL)

```

```

1008         ndiff = ngroups_max - 1; /* Needs one for pcred_t */
1009     else
1010         ndiff = ngroups_max - cr->cr_grps->crg_ngroups;
1011
1012     return (ucredsize - ndiff * sizeof (gid_t));
1013 }
1014
1015 /*
1016 * Get the "ucred" of a process.
1017 */
1018 struct ucred_s *
1019 pgetucred(proc_t *p)
1020 {
1021     cred_t *cr;
1022     struct ucred_s *uc;
1023
1024     mutex_enter(&p->p_crlock);
1025     cr = p->p_cred;
1026     crhold(cr);
1027     mutex_exit(&p->p_crlock);
1028
1029     uc = cred2ucred(cr, p->p_pid, NULL, CRED());
1030     crfree(cr);
1031
1032     return (uc);
1033 }
1034
1035 /*
1036 * If the reply status is NFSERR_EACCES, it may be because we are
1037 * root (no root net access). Check the real uid, if it isn't root
1038 * make that the uid instead and retry the call.
1039 * Private interface for NFS.
1040 */
1041 cred_t *
1042 crnetadjust(cred_t *cr)
1043 {
1044     if (cr->cr_uid == 0 && cr->cr_ruid != 0) {
1045         cr = crdup(cr);
1046         cr->cr_uid = cr->cr_ruid;
1047         return (cr);
1048     }
1049     return (NULL);
1050 }
1051
1052 /*
1053 * The reference count is of interest when you want to check
1054 * whether it is ok to modify the credential in place.
1055 */
1056 uint_t
1057 crgetref(const cred_t *cr)
1058 {
1059     return (cr->cr_ref);
1060 }
1061
1062 static int
1063 get_c2audit_load(void)
1064 {
1065     static int gotit = 0;
1066     static int c2audit_load;
1067
1068     if (gotit)
1069         return (c2audit_load);
1070     c2audit_load = 1; /* set default value once */
1071     if (mod_sysctl(SYS_CHECK_EXCLUDE, "c2audit") != 0)
1072         c2audit_load = 0;
1073     gotit++;

```

```

1075     return (c2audit_load);
1076 }

1078 int
1079 get_audit_ucrsize(void)
1080 {
1081     return (get_c2audit_load() ? sizeof (auditinfo64_addr_t) : 0);
1082 }

1084 /*
1085  * Set zone pointer in credential to indicated value. First adds a
1086  * hold for the new zone, then drops the hold on previous zone (if any).
1087  * This is done in this order in case the old and new zones are the
1088  * same.
1089  */
1090 void
1091 crsetzone(cred_t *cr, zone_t *zptr)
1092 {
1093     zone_t *oldzptr = cr->cr_zone;

1095     ASSERT(cr != kcred);
1096     ASSERT(cr->cr_ref <= 2);
1097     cr->cr_zone = zptr;
1098     zone_cred_hold(zptr);
1099     if (oldzptr)
1100         zone_cred_rele(oldzptr);
1101 }

1103 /*
1104  * Create a new cred based on the supplied label
1105  */
1106 cred_t *
1107 newcred_from_bslabel(bslabel_t *blabel, uint32_t doi, int flags)
1108 {
1109     ts_label_t *lbl = labelalloc(blabel, doi, flags);
1110     cred_t *cr = NULL;

1112     if (lbl != NULL) {
1113         if ((cr = crdup_flags(dummycr, flags)) != NULL) {
1114             cr->cr_label = lbl;
1115         } else {
1116             label_rele(lbl);
1117         }
1118     }

1120     return (cr);
1121 }

1123 /*
1124  * Derive a new cred from the existing cred, but with a different label.
1125  * To be used when a cred is being shared, but the label needs to be changed
1126  * by a caller without affecting other users
1127  */
1128 cred_t *
1129 copycred_from_tslabel(const cred_t *cr, ts_label_t *label, int flags)
1130 {
1131     cred_t *newcr = NULL;

1133     if ((newcr = crdup_flags(cr, flags)) != NULL) {
1134         if (newcr->cr_label != NULL)
1135             label_rele(newcr->cr_label);
1136         label_hold(label);
1137         newcr->cr_label = label;
1138     }

```

```

1140     return (newcr);
1141 }

1143 /*
1144  * Derive a new cred from the existing cred, but with a different label.
1145  */
1146 cred_t *
1147 copycred_from_bslabel(const cred_t *cr, bslabel_t *blabel,
1148     uint32_t doi, int flags)
1149 {
1150     ts_label_t *lbl = labelalloc(blabel, doi, flags);
1151     cred_t *newcr = NULL;

1153     if (lbl != NULL) {
1154         newcr = copycred_from_tslabel(cr, lbl, flags);
1155         label_rele(lbl);
1156     }

1158     return (newcr);
1159 }

1161 /*
1162  * This function returns a pointer to the kcred-equivalent in the current zone.
1163  */
1164 cred_t *
1165 zone_kcred(void)
1166 {
1167     zone_t *zone;

1169     if ((zone = CRED()->cr_zone) != NULL)
1170         return (zone->zone_kcred);
1171     else
1172         return (kcred);
1173 }

1175 boolean_t
1176 valid_ephemeral_uid(zone_t *zone, uid_t id)
1177 {
1178     ephemeral_zsd_t *eph_zsd;
1179     if (id <= IDMAP_WK_MAX_UID)
1180         return (B_TRUE);

1182     eph_zsd = get_ephemeral_zsd(zone);
1183     ASSERT(eph_zsd != NULL);
1184     membar_consumer();
1185     return (id > eph_zsd->min_uid && id <= eph_zsd->last_uid);
1186 }

1188 boolean_t
1189 valid_ephemeral_gid(zone_t *zone, gid_t id)
1190 {
1191     ephemeral_zsd_t *eph_zsd;
1192     if (id <= IDMAP_WK_MAX_GID)
1193         return (B_TRUE);

1195     eph_zsd = get_ephemeral_zsd(zone);
1196     ASSERT(eph_zsd != NULL);
1197     membar_consumer();
1198     return (id > eph_zsd->min_gid && id <= eph_zsd->last_gid);
1199 }

1201 int
1202 eph_uid_alloc(zone_t *zone, int flags, uid_t *start, int count)
1203 {
1204     ephemeral_zsd_t *eph_zsd = get_ephemeral_zsd(zone);

```



```

1206     ASSERT(eph_zsd != NULL);
1208     mutex_enter(&eph_zsd->eph_lock);
1210     /* Test for unsigned integer wrap around */
1211     if (eph_zsd->last_uid + count < eph_zsd->last_uid) {
1212         mutex_exit(&eph_zsd->eph_lock);
1213         return (-1);
1214     }
1216     /* first call or idmap crashed and state corrupted */
1217     if (flags != 0)
1218         eph_zsd->min_uid = eph_zsd->last_uid;
1220     hasephids = B_TRUE;
1221     *start = eph_zsd->last_uid + 1;
1222     atomic_add_32(&eph_zsd->last_uid, count);
1223     mutex_exit(&eph_zsd->eph_lock);
1224     return (0);
1225 }
1227 int
1228 eph_gid_alloc(zone_t *zone, int flags, gid_t *start, int count)
1229 {
1230     ephemeral_zsd_t *eph_zsd = get_ephemeral_zsd(zone);
1232     ASSERT(eph_zsd != NULL);
1234     mutex_enter(&eph_zsd->eph_lock);
1236     /* Test for unsigned integer wrap around */
1237     if (eph_zsd->last_gid + count < eph_zsd->last_gid) {
1238         mutex_exit(&eph_zsd->eph_lock);
1239         return (-1);
1240     }
1242     /* first call or idmap crashed and state corrupted */
1243     if (flags != 0)
1244         eph_zsd->min_gid = eph_zsd->last_gid;
1246     hasephids = B_TRUE;
1247     *start = eph_zsd->last_gid + 1;
1248     atomic_add_32(&eph_zsd->last_gid, count);
1249     mutex_exit(&eph_zsd->eph_lock);
1250     return (0);
1251 }
1253 /*
1254  * IMPORTANT.The two functions get_ephemeral_data() and set_ephemeral_data()
1255  * are project private functions that are for use of the test system only and
1256  * are not to be used for other purposes.
1257  */
1259 void
1260 get_ephemeral_data(zone_t *zone, uid_t *min_uid, uid_t *last_uid,
1261                  gid_t *min_gid, gid_t *last_gid)
1262 {
1263     ephemeral_zsd_t *eph_zsd = get_ephemeral_zsd(zone);
1265     ASSERT(eph_zsd != NULL);
1267     mutex_enter(&eph_zsd->eph_lock);
1269     *min_uid = eph_zsd->min_uid;
1270     *last_uid = eph_zsd->last_uid;
1271     *min_gid = eph_zsd->min_gid;

```

```

1272     *last_gid = eph_zsd->last_gid;
1274     mutex_exit(&eph_zsd->eph_lock);
1275 }
1278 void
1279 set_ephemeral_data(zone_t *zone, uid_t min_uid, uid_t last_uid,
1280                  gid_t min_gid, gid_t last_gid)
1281 {
1282     ephemeral_zsd_t *eph_zsd = get_ephemeral_zsd(zone);
1284     ASSERT(eph_zsd != NULL);
1286     mutex_enter(&eph_zsd->eph_lock);
1288     if (min_uid != 0)
1289         eph_zsd->min_uid = min_uid;
1290     if (last_uid != 0)
1291         eph_zsd->last_uid = last_uid;
1292     if (min_gid != 0)
1293         eph_zsd->min_gid = min_gid;
1294     if (last_gid != 0)
1295         eph_zsd->last_gid = last_gid;
1297     mutex_exit(&eph_zsd->eph_lock);
1298 }
1300 /*
1301  * If the credential user SID or group SID is mapped to an ephemeral
1302  * ID, map the credential to nobody.
1303  */
1304 cred_t *
1305 crgetmapped(const cred_t *cr)
1306 {
1307     ephemeral_zsd_t *eph_zsd;
1308     /*
1309      * Someone incorrectly passed a NULL cred to a vnode operation
1310      * either on purpose or by calling CRED() in interrupt context.
1311      */
1312     if (cr == NULL)
1313         return (NULL);
1315     if (cr->cr_ksid != NULL) {
1316         if (cr->cr_ksid->kr_sidx[KSID_USER].ks_id > MAXUID) {
1317             eph_zsd = get_ephemeral_zsd(crgetzone(cr));
1318             return (eph_zsd->eph_nobody);
1319         }
1321         if (cr->cr_ksid->kr_sidx[KSID_GROUP].ks_id > MAXUID) {
1322             eph_zsd = get_ephemeral_zsd(crgetzone(cr));
1323             return (eph_zsd->eph_nobody);
1324         }
1325     }
1327     return ((cred_t *)cr);
1328 }
1330 /* index should be in range for a ksidindex_t */
1331 void
1332 crsetssid(cred_t *cr, ksid_t *ksp, int index)
1333 {
1334     ASSERT(cr->cr_ref <= 2);
1335     ASSERT(index >= 0 && index < KSID_COUNT);
1336     if (cr->cr_ksid == NULL && ksp == NULL)
1337         return;

```

```

1338     cr->cr_ksid = kcrsid_setsid(cr->cr_ksid, ksp, index);
1339 }

1341 void
1342 crsetsidlist(cred_t *cr, ksidlist_t *ksl)
1343 {
1344     ASSERT(cr->cr_ref <= 2);
1345     if (cr->cr_ksid == NULL && ksl == NULL)
1346         return;
1347     cr->cr_ksid = kcrsid_setsidlist(cr->cr_ksid, ksl);
1348 }

1350 ksid_t *
1351 crgetsid(const cred_t *cr, int i)
1352 {
1353     ASSERT(i >= 0 && i < KSID_COUNT);
1354     if (cr->cr_ksid != NULL && cr->cr_ksid->kr_sidx[i].ks_domain)
1355         return ((ksid_t *)&cr->cr_ksid->kr_sidx[i]);
1356     return (NULL);
1357 }

1359 ksidlist_t *
1360 crgetsidlist(const cred_t *cr)
1361 {
1362     if (cr->cr_ksid != NULL)
1363         return (cr->cr_ksid->kr_sidlist);
1364     return (NULL);
1365 }

1367 /*
1368  * Interface to set the effective and permitted privileges for
1369  * a credential; this interface does no security checks and is
1370  * intended for kernel (file)servers creating credentials with
1371  * specific privileges.
1372  */
1373 int
1374 crsetpriv(cred_t *cr, ...)
1375 {
1376     va_list ap;
1377     const char *privnm;

1379     ASSERT(cr->cr_ref <= 2);

1381     priv_set_PA(cr);

1383     va_start(ap, cr);

1385     while ((privnm = va_arg(ap, const char *)) != NULL) {
1386         int priv = priv_getbyname(privnm, 0);
1387         if (priv < 0)
1388             return (-1);

1390         priv_addset(&CR_PPRIV(cr), priv);
1391         priv_addset(&CR_EPRIV(cr), priv);
1392     }
1393     priv_adjust_PA(cr);
1394     va_end(ap);
1395     return (0);
1396 }

1398 /*
1399  * Interface to effectively set the PRIV_ALL for
1400  * a credential; this interface does no security checks and is
1401  * intended for kernel (file)servers to extend the user credentials
1402  * to be ALL, like either kcred or zcred.
1403  */

```

```

1404 void
1405 crset_zone_privall(cred_t *cr)
1406 {
1407     zone_t *zone = crgetzone(cr);

1409     priv_fillset(&CR_LPRIV(cr));
1410     CR_EPRIV(cr) = CR_PPRIV(cr) = CR_IPRIV(cr) = CR_LPRIV(cr);
1411     priv_intersect(zone->zone_privset, &CR_LPRIV(cr));
1412     priv_intersect(zone->zone_privset, &CR_EPRIV(cr));
1413     priv_intersect(zone->zone_privset, &CR_IPRIV(cr));
1414     priv_intersect(zone->zone_privset, &CR_PPRIV(cr));
1415 }

1417 struct credklpd *
1418 crgetcrkklpd(const cred_t *cr)
1419 {
1420     return (cr->cr_klpd);
1421 }

1423 void
1424 crsetcrkklpd(cred_t *cr, struct credklpd *crkklpd)
1425 {
1426     ASSERT(cr->cr_ref <= 2);

1428     if (cr->cr_klpd != NULL)
1429         crkklpd_rele(cr->cr_klpd);
1430     cr->cr_klpd = crkklpd;
1431 }

1433 credgrp_t *
1434 crgrpcopyin(int n, gid_t *gidset)
1435 {
1436     credgrp_t *mem;
1437     size_t sz = CREDGRPSZ(n);

1439     ASSERT(n > 0);

1441     mem = kmem_alloc(sz, KM_SLEEP);

1443     if (copyin(gidset, mem->crg_groups, sizeof (gid_t) * n)) {
1444         kmem_free(mem, sz);
1445         return (NULL);
1446     }
1447     mem->crg_ref = 1;
1448     mem->crg_ngroups = n;
1449     qsort(mem->crg_groups, n, sizeof (gid_t), gidcmp);
1450     return (mem);
1451 }

1453 const gid_t *
1454 crgetggroups(const credgrp_t *grps)
1455 {
1456     return (grps->crg_groups);
1457 }

1459 void
1460 crsetcredgrp(cred_t *cr, credgrp_t *grps)
1461 {
1462     ASSERT(cr->cr_ref <= 2);

1464     if (cr->cr_grps != NULL)
1465         crgrprele(cr->cr_grps);

1467     cr->cr_grps = grps;
1468 }

```

```
1470 void
1471 crgrprele(credgrp_t *grps)
1472 {
1473     if (atomic_dec_32_nv(&grps->crg_ref) == 0)
1474         kmem_free(grps, CREDGRPSZ(grps->crg_ngroups));
1475 }

1477 static void
1478 crgrphold(credgrp_t *grps)
1479 {
1480     atomic_inc_32(&grps->crg_ref);
1481 }
```

new/usr/src/uts/common/os/exec.c

1

```
*****
54319 Thu Jun 30 22:03:26 2016
new/usr/src/uts/common/os/exec.c
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
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 /*
23  * Copyright (c) 1988, 2010, Oracle and/or its affiliates. All rights reserved.
24 */
25
26 /*      Copyright (c) 1988 AT&T */
27 /*      All Rights Reserved      */
28 /*
29  * Copyright 2014, Joyent, Inc.  All rights reserved.
30 */
31
32 #include <sys/types.h>
33 #include <sys/param.h>
34 #include <sys/sysmacros.h>
35 #include <sys/system.h>
36 #include <sys/signal.h>
37 #include <sys/cred_impl.h>
38 #include <sys/policy.h>
39 #include <sys/user.h>
40 #include <sys/errno.h>
41 #include <sys/file.h>
42 #include <sys/vfs.h>
43 #include <sys/vnode.h>
44 #include <sys/mman.h>
45 #include <sys/acct.h>
46 #include <sys/cpuvar.h>
47 #include <sys/proc.h>
48 #include <sys/cmn_err.h>
49 #include <sys/debug.h>
50 #include <sys/pathname.h>
51 #include <sys/vm.h>
52 #include <sys/lgrp.h>
53 #include <sys/vtrace.h>
54 #include <sys/exec.h>
55 #include <sys/exechdr.h>
56 #include <sys/kmem.h>
57 #include <sys/prsystem.h>
58 #include <sys/modctl.h>
```

new/usr/src/uts/common/os/exec.c

2

```
59 #include <sys/vmparam.h>
60 #include <sys/door.h>
61 #include <sys/schedctl.h>
62 #include <sys/utrap.h>
63 #include <sys/systeminfo.h>
64 #include <sys/stack.h>
65 #include <sys/rctl.h>
66 #include <sys/dtrace.h>
67 #include <sys/lwpchan_impl.h>
68 #include <sys/pool.h>
69 #include <sys/sdt.h>
70 #include <sys/brand.h>
71 #include <sys/klpd.h>
72 #include <sys/random.h>
73 #endif /* ! codereview */
74
75 #include <c2/audit.h>
76
77 #include <vm/hat.h>
78 #include <vm/anon.h>
79 #include <vm/as.h>
80 #include <vm/seg.h>
81 #include <vm/seg_vn.h>
82
83 #define PRIV_RESET          0x01    /* needs to reset privs */
84 #define PRIV_SETID         0x02    /* needs to change uids */
85 #define PRIV_SETUGID       0x04    /* is setuid/setgid/forced privs */
86 #define PRIV_INCREASE      0x08    /* child runs with more privs */
87 #define MAC_FLAGS          0x10    /* need to adjust MAC flags */
88 #define PRIV_FORCED        0x20    /* has forced privileges */
89
90 static int execsetid(struct vnode *, struct vattr *, uid_t *, uid_t *,
91     priv_set_t *, cred_t *, const char *);
92 static int hold_execlw(struct execlw *);
93
94 uint_t auxv_hwcaps = 0; /* auxv AT_SUN_HWCAP value; determined on the fly */
95 uint_t auxv_hwcaps_2 = 0; /* AT_SUN_HWCAP2 */
96 #if defined(_SYSCALL32_IMPL)
97 uint_t auxv_hwcaps32 = 0; /* 32-bit version of auxv_hwcaps */
98 uint_t auxv_hwcaps32_2 = 0; /* 32-bit version of auxv_hwcaps */
99 #endif
100
101 #define PSUIDFLAGS          (SNOCD|SUGID)
102
103 /*
104  * These are consumed within the specific exec modules, but are defined here
105  * because
106  *
107  * 1) The exec modules are unloadable, which would make this near useless.
108  *
109  * 2) We want them to be common across all of them, should more than ELF come
110  *    to support them.
111  *
112  * All must be powers of 2.
113  */
114 size_t aslr_max_brk_skew = 16 * 1024 * 1024; /* 16MB */
115 #pragma weak exec_stackgap = aslr_max_stack_skew /* Old, compatible name */
116 size_t aslr_max_stack_skew = 64 * 1024; /* 64KB */
117
118 /*
119 #endif /* ! codereview */
120 * exece() - system call wrapper around exec_common()
121 */
122 int
123 exece(const char *fname, const char **argv, const char **envp)
124 {
```

```

125     int error;

127     error = exec_common(fname, argp, envp, EBA_NONE);
128     return (error ? (set_errno(error)) : 0);
129 }

131 int
132 exec_common(const char *fname, const char **argp, const char **envp,
133            int brand_action)
134 {
135     vnode_t *vp = NULL, *dir = NULL, *tmpvp = NULL;
136     proc_t *p = ttoproc(curthread);
137     klpw_t *lwp = ttolwp(curthread);
138     struct user *up = PTOU(p);
139     long execsz;          /* temporary count of exec size */
140     int i;
141     int error;
142     char exec_file[MAXCOMLEN+1];
143     struct pathname pn;
144     struct pathname resolvepn;
145     struct uarg args;
146     struct execa ua;
147     k_sigset_t savedmask;
148     lwpdir_t *lwpdir = NULL;
149     tidhash_t *tidhash;
150     lwpdir_t *old_lwpdir = NULL;
151     uint_t old_lwpdir_sz;
152     tidhash_t *old_tidhash;
153     uint_t old_tidhash_sz;
154     ret_tidhash_t *ret_tidhash;
155     lwpent_t *lep;
156     boolean_t brandme = B_FALSE;

158     /*
159     * exec() is not supported for the /proc agent lwp.
160     */
161     if (curthread == p->p_agenttpp)
162         return (ENOTSUP);

164     if (brand_action != EBA_NONE) {
165         /*
166         * Brand actions are not supported for processes that are not
167         * running in a branded zone.
168         */
169         if (!ZONE_IS_BRANDED(p->p_zone))
170             return (ENOTSUP);

172         if (brand_action == EBA_NATIVE) {
173             /* Only branded processes can be unbranded */
174             if (!PROC_IS_BRANDED(p))
175                 return (ENOTSUP);
176         } else {
177             /* Only unbranded processes can be branded */
178             if (PROC_IS_BRANDED(p))
179                 return (ENOTSUP);
180             brandme = B_TRUE;
181         }
182     } else {
183         /*
184         * If this is a native zone, or if the process is already
185         * branded, then we don't need to do anything.  If this is
186         * a native process in a branded zone, we need to brand the
187         * process as it exec()'s the new binary.
188         */
189         if (ZONE_IS_BRANDED(p->p_zone) && !PROC_IS_BRANDED(p))
190             brandme = B_TRUE;

```

```

191     }

193     /*
194     * Inform /proc that an exec() has started.
195     * Hold signals that are ignored by default so that we will
196     * not be interrupted by a signal that will be ignored after
197     * successful completion of gexec().
198     */
199     mutex_enter(&p->p_lock);
200     prexecstart();
201     schedctl_finish_sigblock(curthread);
202     savedmask = curthread->t_hold;
203     sigorset(&curthread->t_hold, &ignoredefault);
204     mutex_exit(&p->p_lock);

206     /*
207     * Look up path name and remember last component for later.
208     * To help coreadm expand its %d token, we attempt to save
209     * the directory containing the executable in p_execdir.  The
210     * first call to lookuppn() may fail and return EINVAL because
211     * dirvpp is non-NULL.  In that case, we make a second call to
212     * lookuppn() with dirvpp set to NULL; p_execdir will be NULL,
213     * but coreadm is allowed to expand %d to the empty string and
214     * there are other cases in which that failure may occur.
215     */
216     if ((error = pn_get((char *)fname, UIO_USERSPACE, &pn)) != 0)
217         goto out;
218     pn_alloc(&resolvepn);
219     if ((error = lookuppn(&pn, &resolvepn, FOLLOW, &dir, &vp)) != 0) {
220         pn_free(&resolvepn);
221         pn_free(&pn);
222         if (error != EINVAL)
223             goto out;

225         dir = NULL;
226         if ((error = pn_get((char *)fname, UIO_USERSPACE, &pn)) != 0)
227             goto out;
228         pn_alloc(&resolvepn);
229         if ((error = lookuppn(&pn, &resolvepn, FOLLOW, NULLVPP,
230                               &vp)) != 0) {
231             pn_free(&resolvepn);
232             pn_free(&pn);
233             goto out;
234         }
235     }
236     if (vp == NULL) {
237         if (dir != NULL)
238             VN_RELE(dir);
239         error = ENOENT;
240         pn_free(&resolvepn);
241         pn_free(&pn);
242         goto out;
243     }

245     if ((error = secpolicy_basic_exec(CRED(), vp)) != 0) {
246         if (dir != NULL)
247             VN_RELE(dir);
248         pn_free(&resolvepn);
249         pn_free(&pn);
250         VN_RELE(vp);
251         goto out;
252     }

254     /*
255     * We do not allow executing files in attribute directories.
256     * We test this by determining whether the resolved path

```

```

257  * contains a "/" when we're in an attribute directory;
258  * only if the pathname does not contain a "/" the resolved path
259  * points to a file in the current working (attribute) directory.
260  */
261  if ((p->p_user.u_cdir->v_flag & V_XATTRDIR) != 0 &&
262      strchr(resolvepn.pn_path, '/') == NULL) {
263      if (dir != NULL)
264          VN_RELE(dir);
265      error = EACCES;
266      pn_free(&resolvepn);
267      pn_free(&pn);
268      VN_RELE(vp);
269      goto out;
270  }

272  bzero(exec_file, MAXCOMLEN+1);
273  (void) strncpy(exec_file, pn.pn_path, MAXCOMLEN);
274  bzero(&args, sizeof (args));
275  args.pathname = resolvepn.pn_path;
276  /* don't free resolvepn until we are done with args */
277  pn_free(&pn);

279  /*
280   * If we're running in a profile shell, then call pfexecd.
281   */
282  if ((CR_FLAGS(p->p_cred) & PRIV_PFEEXEC) != 0) {
283      error = pfexec_call(p->p_cred, &resolvepn, &args.pfcred,
284                          &args.scrubenv);

286      /* Returning errno in case we're not allowed to execute. */
287      if (error > 0) {
288          if (dir != NULL)
289              VN_RELE(dir);
290          pn_free(&resolvepn);
291          VN_RELE(vp);
292          goto out;
293      }

295      /* Don't change the credentials when using old ptrace. */
296      if (args.pfcred != NULL &&
297          (p->p_proc_flag & P_PR_PTRACE) != 0) {
298          crfree(args.pfcred);
299          args.pfcred = NULL;
300          args.scrubenv = E_FALSE;
301      }
302  }

304  /*
305   * Specific exec handlers, or policies determined via
306   * /etc/system may override the historical default.
307   */
308  args.stk_prot = PROT_ZFOD;
309  args.dat_prot = PROT_ZFOD;

311  CPU_STATS_ADD_K(sys, sysexec, 1);
312  DTRACE_PROCL(exec, char *, args.pathname);

314  ua.fname = fname;
315  ua.argp = argp;
316  ua.envp = envp;

318  /* If necessary, brand this process before we start the exec. */
319  if (brandme)
320      brand_setbrand(p);

322  if ((error = gexec(&vp, &ua, &args, NULL, 0, &execsz,

```

```

323      exec_file, p->p_cred, brand_action)) != 0) {
324      if (brandme)
325          brand_clearbrand(p, B_FALSE);
326      VN_RELE(vp);
327      if (dir != NULL)
328          VN_RELE(dir);
329      pn_free(&resolvepn);
330      goto fail;
331  }

333  /*
334   * Free floating point registers (sun4u only)
335   */
336  ASSERT(lwp != NULL);
337  lwp_freeregs(lwp, 1);

339  /*
340   * Free thread and process context ops.
341   */
342  if (curthread->t_ctx)
343      freectx(curthread, 1);
344  if (p->p_pctx)
345      freepctx(p, 1);

347  /*
348   * Remember file name for accounting; clear any cached DTrace predicate.
349   */
350  up->u_acflag &= ~AFORK;
351  bcopy(exec_file, up->u_comm, MAXCOMLEN+1);
352  curthread->t_predcache = NULL;

354  /*
355   * Clear contract template state
356   */
357  lwp_tmpl_clear(lwp);

359  /*
360   * Save the directory in which we found the executable for expanding
361   * the %d token used in core file patterns.
362   */
363  mutex_enter(&p->p_lock);
364  tmpvp = p->p_execdir;
365  p->p_execdir = dir;
366  if (p->p_execdir != NULL)
367      VN_HOLD(p->p_execdir);
368  mutex_exit(&p->p_lock);

370  if (tmpvp != NULL)
371      VN_RELE(tmpvp);

373  /*
374   * Reset stack state to the user stack, clear set of signals
375   * caught on the signal stack, and reset list of signals that
376   * restart system calls; the new program's environment should
377   * not be affected by detritus from the old program. Any
378   * pending held signals remain held, so don't clear t_hold.
379   */
380  mutex_enter(&p->p_lock);
381  lwp->lwp_oldcontext = 0;
382  lwp->lwp_ustack = 0;
383  lwp->lwp_old_stk_ctl = 0;
384  sigemptyset(&sup->u_signodefer);
385  sigemptyset(&sup->u_sigonstack);
386  sigemptyset(&sup->u_sigresethand);
387  lwp->lwp_sigaltstack.ss_sp = 0;
388  lwp->lwp_sigaltstack.ss_size = 0;

```

```

389     lwp->lwp_sigaltstack.ss_flags = SS_DISABLE;
391     /*
392     * Make saved resource limit == current resource limit.
393     */
394     for (i = 0; i < RLIM_NLIMITS; i++) {
395         /*CONSTCOND*/
396         if (RLIM_SAVED(i)) {
397             (void) rctl_rlimit_get(rctlproc_legacy[i], p,
398                 &up->u_saved_rlimit[i]);
399         }
400     }
402     /*
403     * If the action was to catch the signal, then the action
404     * must be reset to SIG_DFL.
405     */
406     sigdefault(p);
407     p->p_flag &= ~(SNOWAIT|SJCTL);
408     p->p_flag |= (SEXECEX|SMSACCT|SMSFORK);
409     up->u_signal[SIGCLD - 1] = SIG_DFL;
411     /*
412     * Delete the dot4 sigqueues/signotifies.
413     */
414     sigqfree(p);
416     mutex_exit(&p->p_lock);
418     mutex_enter(&p->p_plock);
419     p->p_prof.pr_base = NULL;
420     p->p_prof.pr_size = 0;
421     p->p_prof.pr_off = 0;
422     p->p_prof.pr_scale = 0;
423     p->p_prof.pr_samples = 0;
424     mutex_exit(&p->p_plock);
426     ASSERT(curthread->t_schedctl == NULL);
428 #if defined(__sparc)
429     if (p->p_utrap != NULL)
430         utrap_free(p);
431 #endif /* __sparc */
433     /*
434     * Close all close-on-exec files.
435     */
436     close_exec(P_FINFO(p));
437     TRACE_2(TR_FAC_PROC, TR_PROC_EXEC, "proc_exec:p %p up %p", p, up);
439     /* Unbrand ourself if necessary. */
440     if (PROC_IS_BRANDED(p) && (brand_action == EBA_NATIVE))
441         brand_clearbrand(p, B_FALSE);
443     setregs(&args);
445     /* Mark this as an executable vnode */
446     mutex_enter(&vp->v_lock);
447     vp->v_flag |= VVMEXEC;
448     mutex_exit(&vp->v_lock);
450     VN_RELE(vp);
451     if (dir != NULL)
452         VN_RELE(dir);
453     pn_free(&resolvepn);

```

```

455     /*
456     * Allocate a new lwp directory and lwpid hash table if necessary.
457     */
458     if (curthread->t_tid != 1 || p->p_lwpdir_sz != 2) {
459         lwpdir = kmem_zalloc(2 * sizeof(lwpdir_t), KM_SLEEP);
460         lwpdir->ld_next = lwpdir + 1;
461         tidhash = kmem_zalloc(2 * sizeof(tidhash_t), KM_SLEEP);
462         if (p->p_lwpdir != NULL)
463             lep = p->p_lwpdir[curthread->t_dslot].ld_entry;
464         else
465             lep = kmem_zalloc(sizeof(*lep), KM_SLEEP);
466     }
468     if (PROC_IS_BRANDED(p))
469         BROP(p)->b_exec();
471     mutex_enter(&p->p_lock);
472     prbarrier(p);
474     /*
475     * Reset lwp id to the default value of 1.
476     * This is a single-threaded process now
477     * and lwp #1 is lwp_wait()able by default.
478     * The t_unpark flag should not be inherited.
479     */
480     ASSERT(p->p_lwpcnt == 1 && p->p_zombcnt == 0);
481     curthread->t_tid = 1;
482     kpreempt_disable();
483     ASSERT(curthread->t_lpl != NULL);
484     p->p_t1_lgrpid = curthread->t_lpl->lpl_lgrpid;
485     kpreempt_enable();
486     if (p->p_tr_lgrpid != LGRP_NONE && p->p_tr_lgrpid != p->p_t1_lgrpid) {
487         lgrp_update_trthr_migrations(1);
488     }
489     curthread->t_unpark = 0;
490     curthread->t_proc_flag |= TP_TWAIT;
491     curthread->t_proc_flag &= ~TP_DAEMON; /* daemons shouldn't exec */
492     p->p_lwpdaemon = 0; /* but oh well ... */
493     p->p_lwpid = 1;
495     /*
496     * Install the newly-allocated lwp directory and lwpid hash table
497     * and insert the current thread into the new hash table.
498     */
499     if (lwpdir != NULL) {
500         old_lwpdir = p->p_lwpdir;
501         old_lwpdir_sz = p->p_lwpdir_sz;
502         old_tidhash = p->p_tidhash;
503         old_tidhash_sz = p->p_tidhash_sz;
504         p->p_lwpdir = p->p_lwpfree = lwpdir;
505         p->p_lwpdir_sz = 2;
506         lep->le_thread = curthread;
507         lep->le_lwpid = curthread->t_tid;
508         lep->le_start = curthread->t_start;
509         lwp_hash_in(p, lep, tidhash, 2, 0);
510         p->p_tidhash = tidhash;
511         p->p_tidhash_sz = 2;
512     }
513     ret_tidhash = p->p_ret_tidhash;
514     p->p_ret_tidhash = NULL;
516     /*
517     * Restore the saved signal mask and
518     * inform /proc that the exec() has finished.
519     */
520     curthread->t_hold = savedmask;

```

```

521     prexecend();
522     mutex_exit(&p->p_lock);
523     if (old_lwmdir) {
524         kmem_free(old_lwmdir, old_lwmdir_sz * sizeof (lwmdir_t));
525         kmem_free(old_tidhash, old_tidhash_sz * sizeof (tidhash_t));
526     }
527     while (ret_tidhash != NULL) {
528         ret_tidhash_t *next = ret_tidhash->rth_next;
529         kmem_free(ret_tidhash->rth_tidhash,
530             ret_tidhash->rth_tidhash_sz * sizeof (tidhash_t));
531         kmem_free(ret_tidhash, sizeof (*ret_tidhash));
532         ret_tidhash = next;
533     }

535     ASSERT(error == 0);
536     DTRACE_PROC(exec__success);
537     return (0);

539 fail:
540     DTRACE_PROCl(exec__failure, int, error);
541 out:
542     /* error return */
543     mutex_enter(&p->p_lock);
544     curthread->t_hold = savedmask;
545     prexecend();
546     mutex_exit(&p->p_lock);
547     ASSERT(error != 0);
548     return (error);

551 /*
552  * Perform generic exec duties and switchout to object-file specific
553  * handler.
554  */
555 int
556 gexec(
557     struct vnode **vpp,
558     struct execa *uap,
559     struct uarg *args,
560     struct intpdata *idatap,
561     int level,
562     long *execsz,
563     caddr_t exec_file,
564     struct cred *cred,
565     int brand_action)
566 {
567     struct vnode *vp, *execvp = NULL;
568     proc_t *pp = ttoproc(curthread);
569     struct execsw *eswp;
570     int error = 0;
571     int suidflags = 0;
572     ssize_t resid;
573     uid_t uid, gid;
574     struct vattr vattr;
575     char magbuf[MAGIC_BYTES];
576     int setid;
577     cred_t *oldcred, *newcred = NULL;
578     int privflags = 0;
579     int setidfl;
580     priv_set_t fset;
581     secflagset_t old_secflags;

583     secflags_copy(&old_secflags, &pp->p_secflags.psf_effective);
584 #endif /* ! codereview */

586     /*

```

```

587     * If the SNOCD or SUGID flag is set, turn it off and remember the
588     * previous setting so we can restore it if we encounter an error.
589     */
590     if (level == 0 && (pp->p_flag & PSUIDFLAGS)) {
591         mutex_enter(&pp->p_lock);
592         suidflags = pp->p_flag & PSUIDFLAGS;
593         pp->p_flag &= ~PSUIDFLAGS;
594         mutex_exit(&pp->p_lock);
595     }

597     if ((error = execpermissions(*vpp, &vattr, args)) != 0)
598         goto bad_noclose;

600     /* need to open vnode for stateful file systems */
601     if ((error = VOP_OPEN(vpp, FREAD, CRED(), NULL)) != 0)
602         goto bad_noclose;
603     vp = *vpp;

605     /*
606     * Note: to support binary compatibility with SunOS a.out
607     * executables, we read in the first four bytes, as the
608     * magic number is in bytes 2-3.
609     */
610     if (error = vn_rdwr(UIO_READ, vp, magbuf, sizeof (magbuf),
611         (offset_t)0, UIO_SYSSPACE, 0, (rlim64_t)0, CRED(), &resid))
612         goto bad;
613     if (resid != 0)
614         goto bad;

616     if ((eswp = findexec_by_hdr(magbuf)) == NULL)
617         goto bad;

619     if (level == 0 &&
620         (privflags = execsetid(vp, &vattr, &uid, &gid, &fset,
621             args->pfcred == NULL ? cred : args->pfcred, args->pathname)) != 0) {
623         /* Pfcred is a credential with a ref count of 1 */

625         if (args->pfcred != NULL) {
626             privflags |= PRIV_INCREASE|PRIV_RESET;
627             newcred = cred = args->pfcred;
628         } else {
629             newcred = cred = crdup(cred);
630         }

632         /* If we can, drop the PA bit */
633         if ((privflags & PRIV_RESET) != 0)
634             priv_adjust_PA(cred);

636         if (privflags & PRIV_SETID) {
637             cred->cr_uid = uid;
638             cred->cr_gid = gid;
639             cred->cr_suid = uid;
640             cred->cr_sgid = gid;
641         }

643         if (privflags & MAC_FLAGS) {
644             if (!(CR_FLAGS(cred) & NET_MAC_AWARE_INHERIT))
645                 CR_FLAGS(cred) &= ~NET_MAC_AWARE;
646             CR_FLAGS(cred) &= ~NET_MAC_AWARE_INHERIT;
647         }

649         /*
650         * Implement the privilege updates:
651         *
652         * Restrict with L:

```



```

653      *
654      *      I' = I & L
655      *
656      *      E' = P' = (I' + F) & A
657      *
658      * But if running under ptrace, we cap I and F with P.
659      */
660      if ((privflags & (PRIV_RESET|PRIV_FORCED)) != 0) {
661          if ((privflags & PRIV_INCREASE) != 0 &&
662              (pp->p_proc_flag & P_PR_PTRACE) != 0) {
663              priv_intersect(&CR_OPPRIV(cred),
664                  &CR_IPRIV(cred));
665              priv_intersect(&CR_OPPRIV(cred), &fset);
666          }
667          priv_intersect(&CR_LPRIV(cred), &CR_IPRIV(cred));
668          CR_EPRIV(cred) = CR_PPRIV(cred) = CR_IPRIV(cred);
669          if (privflags & PRIV_FORCED) {
670              priv_set_PA(cred);
671              priv_union(&fset, &CR_EPRIV(cred));
672              priv_union(&fset, &CR_PPRIV(cred));
673          }
674          priv_adjust_PA(cred);
675      }
676      } else if (level == 0 && args->pfcrcd != NULL) {
677          newcred = cred = args->pfcrcd;
678          privflags |= PRIV_INCREASE;
679          /* pfcrcd is not forced to adhere to these settings */
680          priv_intersect(&CR_LPRIV(cred), &CR_IPRIV(cred));
681          CR_EPRIV(cred) = CR_PPRIV(cred) = CR_IPRIV(cred);
682          priv_adjust_PA(cred);
683      }
684
685      /* The new image gets the inheritable secflags as its secflags */
686      secflags_promote(pp);
687
688      #endif /* ! codereview */
689      /* SunOS 4.x buy-back */
690      if ((vp->v_vfsp->vfs_flag & VFS_NOSETUID) &&
691          (vattr.va_mode & (VSUID|VSGID))) {
692          char path[MAXNAMELEN];
693          refstr_t *mntpt = NULL;
694          int ret = -1;
695
696          bzero(path, sizeof (path));
697          zone_hold(pp->p_zone);
698
699          ret = vnodetopath(pp->p_zone->zone_rootvp, vp, path,
700              sizeof (path), cred);
701
702          /* fallback to mountpoint if a path can't be found */
703          if ((ret != 0) || (ret == 0 && path[0] == '\0'))
704              mntpt = vfs_getmntpoint(vp->v_vfsp);
705
706          if (mntpt == NULL)
707              zcmn_err(pp->p_zone->zone_id, CE_NOTE,
708                  "uid %d: setuid execution not allowed, "
709                  "file=%s", cred->cr_uid, path);
710          else
711              zcmn_err(pp->p_zone->zone_id, CE_NOTE,
712                  "uid %d: setuid execution not allowed, "
713                  "fs=%s, file=%s", cred->cr_uid,
714                  ZONE_PATH_TRANSLATE(refstr_value(mntpt),
715                      pp->p_zone), exec_file);
716
717          if (!INGLOBALZONE(pp)) {
718              /* zone_rootpath always has trailing / */

```

```

719          if (mntpt == NULL)
720              cmn_err(CE_NOTE, "!zone: %s, uid: %d "
721                  "setuid execution not allowed, file=%s",
722                  pp->p_zone->zone_name, cred->cr_uid,
723                  pp->p_zone->zone_rootpath, path + 1);
724          else
725              cmn_err(CE_NOTE, "!zone: %s, uid: %d "
726                  "setuid execution not allowed, fs=%s, "
727                  "file=%s", pp->p_zone->zone_name,
728                  cred->cr_uid, refstr_value(mntpt),
729                  exec_file);
730          }
731
732          if (mntpt != NULL)
733              refstr_rele(mntpt);
734
735          zone_rele(pp->p_zone);
736      }
737
738      /*
739      * execsetid() told us whether or not we had to change the
740      * credentials of the process. In privflags, it told us
741      * whether we gained any privileges or executed a set-uid executable.
742      */
743      setid = (privflags & (PRIV_SETUGID|PRIV_INCREASE|PRIV_FORCED));
744
745      /*
746      * Use /etc/system variable to determine if the stack
747      * should be marked as executable by default.
748      */
749      if ((noexec_user_stack != 0) ||
750          secflag_enabled(pp, PROC_SEC_NOEXECSTACK))
751          if (noexec_user_stack)
752              args->stk_prot &= ~PROT_EXEC;
753
754      args->execswp = eswp; /* Save execsw pointer in uarg for exec_func */
755      args->ex_vp = vp;
756
757      /*
758      * Traditionally, the setid flags told the sub processes whether
759      * the file just executed was set-uid or set-gid; this caused
760      * some confusion as the 'setid' flag did not match the SUGID
761      * process flag which is only set when the uids/gids do not match.
762      * A script set-gid/set-uid to the real uid/gid would start with
763      * /dev/fd/X but an executable would happily trust LD_LIBRARY_PATH.
764      * Now we flag those cases where the calling process cannot
765      * be trusted to influence the newly exec'ed process, either
766      * because it runs with more privileges or when the uids/gids
767      * do in fact not match.
768      * This also makes the runtime linker agree with the on exec
769      * values of SNOCD and SUGID.
770      */
771      setidfl = 0;
772      if (cred->cr_uid != cred->cr_ruid || (cred->cr_rgid != cred->cr_gid &&
773          !supgroupmember(cred->cr_gid, cred))) {
774          setidfl |= EXECSETID_UGIDS;
775      }
776      if (setid & PRIV_SETUGID)
777          setidfl |= EXECSETID_SETID;
778      if (setid & PRIV_FORCED)
779          setidfl |= EXECSETID_PRIVS;
780
781      execvp = pp->p_exec;
782      if (execvp)
783          VN_HOLD(execvp);

```

```

784 error = (*eswp->exec_func)(vp, uap, args, idatap, level, execsz,
785     setidfl, exec_file, cred, brand_action);
786 rw_exit(eswp->exec_lock);
787 if (error != 0) {
788     if (execvp)
789         VN_RELE(execvp);
790     /*
791      * If this process's p_exec has been set to the vp of
792      * the executable by exec_func, we will return without
793      * calling VOP_CLOSE because proc_exit will close it
794      * on exit.
795      */
796     if (pp->p_exec == vp)
797         goto bad_noclose;
798     else
799         goto bad;
800 }

802 if (level == 0) {
803     uid_t oruid;

805     if (execvp != NULL) {
806         /*
807          * Close the previous executable only if we are
808          * at level 0.
809          */
810         (void) VOP_CLOSE(execvp, FREAD, 1, (offset_t)0,
811             cred, NULL);
812     }

814     mutex_enter(&pp->p_crlock);

816     oruid = pp->p_cred->cr_ruid;

818     if (newcred != NULL) {
819         /*
820          * Free the old credentials, and set the new ones.
821          * Do this for both the process and the (single) thread.
822          */
823         crfree(pp->p_cred);
824         pp->p_cred = newcred; /* cred already held for proc */
825         crhold(newcred); /* hold new cred for thread */
826         /*
827          * DTrace accesses t_cred in probe context. t_cred
828          * must always be either NULL, or point to a valid,
829          * allocated cred structure.
830          */
831         oldcred = curthread->t_cred;
832         curthread->t_cred = newcred;
833         crfree(oldcred);

835         if (priv_basic_test >= 0 &&
836             !PRIV_ISMEMBER(&CR_IPRIV(newcred),
158             !PRIV_ISASSERT(&CR_IPRIV(newcred),
837             priv_basic_test)) {
838             pid_t pid = pp->p_pid;
839             char *fn = PTOU(pp)->u_comm;

841             cmn_err(CE_WARN, "%s[%d]: exec: basic_test "
842                 "privilege removed from E/I", fn, pid);
843         }
844     }
845     /*
846      * On emerging from a successful exec(), the saved
847      * uid and gid equal the effective uid and gid.
848      */

```

```

849     cred->cr_suid = cred->cr_uid;
850     cred->cr_sgid = cred->cr_gid;

852     /*
853      * If the real and effective ids do not match, this
854      * is a setuid process that should not dump core.
855      * The group comparison is tricky; we prevent the code
856      * from flagging SNOCD when executing with an effective gid
857      * which is a supplementary group.
858      */
859     if (cred->cr_ruid != cred->cr_uid ||
860         (cred->cr_rgid != cred->cr_gid &&
861         !supgroupmember(cred->cr_gid, cred)) ||
862         (privflags & PRIV_INCREASE) != 0)
863         suidflags = PSUIDFLAGS;
864     else
865         suidflags = 0;

867     mutex_exit(&pp->p_crlock);
868     if (newcred != NULL && oruid != newcred->cr_ruid) {
869         /* Note that the process remains in the same zone. */
870         mutex_enter(&pidlock);
871         upcount_dec(oruid, crgetzoneid(newcred));
872         upcount_inc(newcred->cr_ruid, crgetzoneid(newcred));
873         mutex_exit(&pidlock);
874     }
875     if (suidflags) {
876         mutex_enter(&pp->p_lock);
877         pp->p_flag |= suidflags;
878         mutex_exit(&pp->p_lock);
879     }
880     if (setid && (pp->p_proc_flag & P_PR_PTRACE) == 0) {
881         /*
882          * If process is traced via /proc, arrange to
883          * invalidate the associated /proc vnode.
884          */
885         if (pp->p_pplist || (pp->p_proc_flag & P_PR_TRACE))
886             args->traceinval = 1;
887     }
888     if (pp->p_proc_flag & P_PR_PTRACE)
889         psignal(pp, SIGTRAP);
890     if (args->traceinval)
891         prinvalidate(&pp->p_user);
892 }
893 if (execvp)
894     VN_RELE(execvp);
895 return (0);

897 bad:
898     (void) VOP_CLOSE(vp, FREAD, 1, (offset_t)0, cred, NULL);

900 bad_noclose:
901     if (newcred != NULL)
902         crfree(newcred);
903     if (error == 0)
904         error = ENOEXEC;

906     mutex_enter(&pp->p_lock);
907 #endif /* ! codereview */
908     if (suidflags) {
909         mutex_enter(&pp->p_lock);
910         pp->p_flag |= suidflags;
911     }
912     /*
913      * Restore the effective seclags, to maintain the invariant they
914      * never change for a given process

```

```

914     */
915     secflags_copy(&p->p_secflags.psf_effective, &old_secflags);
916 #endif /* ! codereview */
917     mutex_exit(&p->p_lock);

230     }
919     return (error);
920 }
    unchanged_portion_omitted_

1824 /*
1825 * Though the actual stack base is constant, slew the %sp by a random aligned
1826 * amount in [0,aslr_max_stack_skew). Mostly, this makes life slightly more
1827 * complicated for buffer overflows hoping to overwrite the return address.
1828 *
1829 * On some platforms this helps avoid cache thrashing when identical processes
1830 * simultaneously share caches that don't provide enough associativity
1831 * (e.g. sun4v systems). In this case stack slewing makes the same hot stack
1832 * variables in different processes live in different cache sets increasing
1833 * effective associativity.
1834 */
1835 size_t
1836 exec_get_spslew(void)
1837 {
1838 #ifdef sun4v
1839     static uint_t sp_color_stride = 16;
1840     static uint_t sp_color_mask = 0x1f;
1841     static uint_t sp_current_color = (uint_t)-1;
1842 #endif
1843     size_t off;

1845     ASSERT(ISP2(aslr_max_stack_skew));

1847     if ((aslr_max_stack_skew == 0) ||
1848         !secflag_enabled(curproc, PROC_SEC_ASLR)) {
1849 #ifdef sun4v
1850         uint_t spcolor = atomic_inc_32_nv(&sp_current_color);
1851         return ((size_t)((spcolor & sp_color_mask) *
1852             SA(sp_color_stride)));
1853 #else
1854         return (0);
1855 #endif
1856     }

1858     (void) random_get_pseudo_bytes((uint8_t *)&off, sizeof (off));
1859     return (SA(P2PHASE(off, aslr_max_stack_skew)));
1860 }

1862 /*
1863 #endif /* ! codereview */
1864 * Initialize a new user stack with the specified arguments and environment.
1865 * The initial user stack layout is as follows:
1866 *
1867 * User Stack
1868 * +-----+ <--- curproc->p_usrstack
1869 * |
1870 * |   slew   |
1871 * |         |
1872 * +-----+
1873 * |  NULL   |
1874 * +-----+
1875 * | auxv strings |
1876 * |         |
1877 * +-----+
1878 * |
1879 * |

```

```

1880 * | envp strings |
1881 * +-----+
1882 * |
1883 * |   argv strings   |
1884 * |         |
1885 * +-----+ <--- ustrp
1886 * |
1887 * |   aux vector     |
1888 * |         |
1889 * +-----+ <--- auxv
1890 * |
1891 * |  NULL           |
1892 * +-----+
1893 * | envp[envc-1]   |
1894 * |         |
1895 * |   ...           |
1896 * +-----+
1897 * | envp[0]        |
1898 * +-----+ <--- envp[]
1899 * |  NULL           |
1900 * +-----+
1901 * | argv[argc-1]   |
1902 * |         |
1903 * |   ...           |
1904 * +-----+
1905 * | argv[0]        |
1906 * +-----+ <--- argv[]
1907 * |  argc           |
1908 * +-----+ <--- stack base
1909 */
1910 int
1911 exec_args(execa_t *uap, uarg_t *args, intpdata_t *intp, void **auxvpp)
1912 {
1913     size_t size;
1914     int error;
1915     proc_t *p = ttoproc(curthread);
1916     user_t *up = PTOU(p);
1917     char *usrstack;
1918     rctl_entity_p_t e;
1919     struct as *as;
1920     extern int use_stk_lpg;
1921     size_t sp_slew;

1923     args->from_model = p->p_model;
1924     if (p->p_model == DATAMODEL_NATIVE) {
1925         args->from_ptrsize = sizeof (long);
1926     } else {
1927         args->from_ptrsize = sizeof (int32_t);
1928     }

1930     if (args->to_model == DATAMODEL_NATIVE) {
1931         args->to_ptrsize = sizeof (long);
1932         args->ncargs = NCARGS;
1933         args->stk_align = STACK_ALIGN;
1934         if (args->addr32)
1935             usrstack = (char *)USRSTACK64_32;
1936         else
1937             usrstack = (char *)USRSTACK;
1938     } else {
1939         args->to_ptrsize = sizeof (int32_t);
1940         args->ncargs = NCARGS32;
1941         args->stk_align = STACK_ALIGN32;
1942         usrstack = (char *)USRSTACK32;
1943     }

1945     ASSERT(P2PHASE((uintptr_t)usrstack, args->stk_align) == 0);

```

```

1947 #if defined(__sparc)
1948 /*
1949  * Make sure user register windows are empty before
1950  * attempting to make a new stack.
1951  */
1952 (void) flush_user_windows_to_stack(NULL);
1953 #endif

1955 for (size = PAGE_SIZE; ; size *= 2) {
1956     args->stk_size = size;
1957     args->stk_base = kmem_alloc(size, KM_SLEEP);
1958     args->stk_strp = args->stk_base;
1959     args->stk_offp = (int *) (args->stk_base + size);
1960     error = stk_copyin(uap, args, intp, auxvpp);
1961     if (error == 0)
1962         break;
1963     kmem_free(args->stk_base, size);
1964     if (error != E2BIG && error != ENAMETOOLONG)
1965         return (error);
1966     if (size >= args->ncargs)
1967         return (E2BIG);
1968 }

1970 size = args->usrstack_size;

1972 ASSERT(error == 0);
1973 ASSERT(P2PHASE(size, args->stk_align) == 0);
1974 ASSERT((ssize_t)STK_AVAIL(args) >= 0);

1976 if (size > args->ncargs) {
1977     kmem_free(args->stk_base, args->stk_size);
1978     return (E2BIG);
1979 }

1981 /*
1982  * Leave only the current lwp and force the other lwps to exit.
1983  * If another lwp beat us to the punch by calling exit(), bail out.
1984  */
1985 if ((error = exitlwps(0)) != 0) {
1986     kmem_free(args->stk_base, args->stk_size);
1987     return (error);
1988 }

1990 /*
1991  * Revoke any doors created by the process.
1992  */
1993 if (p->p_door_list)
1994     door_exit();

1996 /*
1997  * Release schedctl data structures.
1998  */
1999 if (p->p_pagep)
2000     schedctl_proc_cleanup();

2002 /*
2003  * Clean up any DTrace helpers for the process.
2004  */
2005 if (p->p_dtrace_helpers != NULL) {
2006     ASSERT(dtrace_helpers_cleanup != NULL);
2007     (*dtrace_helpers_cleanup)();
2008 }

2010 mutex_enter(&p->p_lock);
2011 /*

```

```

2012     * Cleanup the DTrace provider associated with this process.
2013     */
2014     if (p->p_dtrace_probes) {
2015         ASSERT(dtrace_fasttrap_exec_ptr != NULL);
2016         dtrace_fasttrap_exec_ptr(p);
2017     }
2018     mutex_exit(&p->p_lock);

2020 /*
2021  * discard the lwpchan cache.
2022  */
2023     if (p->p_lcp != NULL)
2024         lwpchan_destroy_cache(1);

2026 /*
2027  * Delete the POSIX timers.
2028  */
2029     if (p->p_itimer != NULL)
2030         timer_exit();

2032 /*
2033  * Delete the ITIMER_REALPROF interval timer.
2034  * The other ITIMER_* interval timers are specified
2035  * to be inherited across exec().
2036  */
2037     delete_itimer_realprof();

2039     if (AU_AUDITING())
2040         audit_exec(args->stk_base, args->stk_base + args->arglen,
2041                 args->na - args->ne, args->ne, args->pfcred);

2043 /*
2044  * Ensure that we don't change resource associations while we
2045  * change address spaces.
2046  */
2047     mutex_enter(&p->p_lock);
2048     pool_barrier_enter();
2049     mutex_exit(&p->p_lock);

2051 /*
2052  * Destroy the old address space and create a new one.
2053  * From here on, any errors are fatal to the exec()ing process.
2054  * On error we return -1, which means the caller must SIGKILL
2055  * the process.
2056  */
2057     relvm();

2059     mutex_enter(&p->p_lock);
2060     pool_barrier_exit();
2061     mutex_exit(&p->p_lock);

2063     up->u_execsw = args->execswp;

2065     p->p_brkbase = NULL;
2066     p->p_brksize = 0;
2067     p->p_brkpageszc = 0;
2068     p->p_stksize = 0;
2069     p->p_stkpageszc = 0;
2070     p->p_model = args->to_model;
2071     p->p_usrstack = usrstack;
2072     p->p_stkprot = args->stk_prot;
2073     p->p_datprot = args->dat_prot;

2075 /*
2076  * Reset resource controls such that all controls are again active as
2077  * well as appropriate to the potentially new address model for the

```

```
2078     * process.
2079     */
2080     e.rcep_p.proc = p;
2081     e.rcep_t = RCENTITY_PROCESS;
2082     rctl_set_reset(p->p_rctls, p, &e);

2084     /* Too early to call map_pgsz for the heap */
2085     if (use_stk_lpg) {
2086         p->p_stkpageszc = page_szc(map_pgsz(MAPPGSZ_STK, p, 0, 0, 0));
2087     }

2089     mutex_enter(&p->p_lock);
2090     p->p_flag |= SAUTOLPG; /* kernel controls page sizes */
2091     mutex_exit(&p->p_lock);

1137     /*
1138     * Some platforms may choose to randomize real stack start by adding a
1139     * small slew (not more than a few hundred bytes) to the top of the
1140     * stack. This helps avoid cache thrashing when identical processes
1141     * simultaneously share caches that don't provide enough associativity
1142     * (e.g. sun4v systems). In this case stack slewing makes the same hot
1143     * stack variables in different processes to live in different cache
1144     * sets increasing effective associativity.
1145     */
2093     sp_slew = exec_get_spslew();
2094     ASSERT(P2PHASE(sp_slew, args->stk_align) == 0);
2095     /* Be certain we don't underflow */
2096     VERIFY((curproc->p_usrstack - (size + sp_slew)) < curproc->p_usrstack);
2097 #endif /* ! codereview */
2098     exec_set_sp(size + sp_slew);

2100     as = as_alloc();
2101     p->p_as = as;
2102     as->a_proc = p;
2103     if (p->p_model == DATAMODEL_ILP32 || args->addr32)
2104         as->a_userlimit = (caddr_t)USERLIMIT32;
2105     (void) hat_setup(as->a_hat, HAT_ALLOC);
2106     hat_join_srd(as->a_hat, args->ex_vp);

2108     /*
2109     * Finally, write out the contents of the new stack.
2110     */
2111     error = stk_copyout(args, usrstack - sp_slew, auxvpp, up);
2112     kmem_free(args->stk_base, args->stk_size);
2113     return (error);
2114 }
```

```

*****
37198 Thu Jun 30 22:03:27 2016
new/usr/src/uts/common/os/fork.c
Code review comments from jeffpc
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
_____unchanged_portion_omitted_____

925 /*
926  * create a child proc struct.
927  */
928 static int
929 getproc(proc_t **cpp, pid_t pid, uint_t flags)
930 {
931     proc_t      *pp, *cp;
932     pid_t       newpid;
933     struct user *uarea;
934     extern uint_t nproc;
935     struct cred *cr;
936     uid_t       ruid;
937     zoneid_t    zoneid;
938     task_t      *task;
939     kproject_t  *proj;
940     zone_t      *zone;
941     int         rctlfail = 0;

943     if (zone_status_get(curproc->p_zone) >= ZONE_IS_SHUTTING_DOWN)
944         return (-1); /* no point in starting new processes */

946     pp = (flags & GETPROC_KERNEL) ? &p0 : curproc;
947     task = pp->p_task;
948     proj = task->tk_proj;
949     zone = pp->p_zone;

951     mutex_enter(&pp->p_lock);
952     mutex_enter(&zone->zone_nlwps_lock);
953     if (proj != proj0p) {
954         if (task->tk_nprocs >= task->tk_nprocs_ctl)
955             if (rctl_test(rc_task_nprocs, task->tk_rctls,
956                 pp, 1, 0) & RCT_DENY)
957                 rctlfail = 1;

959         if (proj->kpj_nprocs >= proj->kpj_nprocs_ctl)
960             if (rctl_test(rc_project_nprocs, proj->kpj_rctls,
961                 pp, 1, 0) & RCT_DENY)
962                 rctlfail = 1;

964         if (zone->zone_nprocs >= zone->zone_nprocs_ctl)
965             if (rctl_test(rc_zone_nprocs, zone->zone_rctls,
966                 pp, 1, 0) & RCT_DENY)
967                 rctlfail = 1;

969         if (rctlfail) {
970             mutex_exit(&zone->zone_nlwps_lock);
971             mutex_exit(&pp->p_lock);
972             atomic_inc_32(&zone->zone_ffcap);
973             goto punish;
974         }
975     }
976     task->tk_nprocs++;
977     proj->kpj_nprocs++;
978     zone->zone_nprocs++;
979     mutex_exit(&zone->zone_nlwps_lock);

```

```

980     mutex_exit(&pp->p_lock);

982     cp = kmem_cache_alloc(process_cache, KM_SLEEP);
983     bzero(cp, sizeof (proc_t));

985     /*
986     * Make proc entry for child process
987     */
988     mutex_init(&cp->p_spllock, NULL, MUTEX_DEFAULT, NULL);
989     mutex_init(&cp->p_crlock, NULL, MUTEX_DEFAULT, NULL);
990     mutex_init(&cp->p_pflock, NULL, MUTEX_DEFAULT, NULL);
991     #if defined(__x86)
992     mutex_init(&cp->p_ldtlock, NULL, MUTEX_DEFAULT, NULL);
993     #endif
994     mutex_init(&cp->p_maplock, NULL, MUTEX_DEFAULT, NULL);
995     cp->p_stat = SIDL;
996     cp->p_mstart = gethrtime();
997     cp->p_as = &kas;
998     /*
999     * p_zone must be set before we call pid_allocate since the process
1000    * will be visible after that and code such as pfind_zone will
1001    * look at the p_zone field.
1002    */
1003     cp->p_zone = pp->p_zone;
1004     cp->p_tl_lgrp_id = LGRP_NONE;
1005     cp->p_tr_lgrp_id = LGRP_NONE;

1007     if ((newpid = pid_allocate(cp, pid, PID_ALLOC_PROC)) == -1) {
1008         if (nproc == v.v_proc) {
1009             CPU_STATS_ADDQ(CPU, sys, procvf, 1);
1010             cmn_err(CE_WARN, "out of processes");
1011         }
1012         goto bad;
1013     }

1015     mutex_enter(&pp->p_lock);
1016     cp->p_exec = pp->p_exec;
1017     cp->p_execdir = pp->p_execdir;
1018     mutex_exit(&pp->p_lock);

1020     if (cp->p_exec) {
1021         VN_HOLD(cp->p_exec);
1022         /*
1023         * Each VOP_OPEN() must be paired with a corresponding
1024         * VOP_CLOSE(). In this case, the executable will be
1025         * closed for the child in either proc_exit() or gexec().
1026         */
1027         if (VOP_OPEN(&cp->p_exec, FREAD, CRED(), NULL) != 0) {
1028             VN_RELE(cp->p_exec);
1029             cp->p_exec = NULLVP;
1030             cp->p_execdir = NULLVFP;
1031             goto bad;
1032         }
1033     }
1034     if (cp->p_execdir)
1035         VN_HOLD(cp->p_execdir);

1037     /*
1038     * If not privileged make sure that this user hasn't exceeded
1039     * v.v_maxup processes, and that users collectively haven't
1040     * exceeded v.v_maxupttl processes.
1041     */
1042     mutex_enter(&pidlock);
1043     ASSERT(nproc < v.v_proc); /* otherwise how'd we get our pid? */
1044     cr = CRED();
1045     ruid = crgetruid(cr);

```

```

1046     zoneid = crgetzoneid(cr);
1047     if (nproc >= v.v_maxup &&          /* short-circuit; usually false */
1048         (nproc >= v.v_maxupttl ||
1049          upcount_get(ruid, zoneid) >= v.v_maxup) &&
1050         secpolicy_newproc(cr) != 0) {
1051         mutex_exit(&pidlock);
1052         zcmn_err(zoneid, CE_NOTE,
1053             "out of per-user processes for uid %d", ruid);
1054         goto bad;
1055     }

1057     /*
1058     * Everything is cool, put the new proc on the active process list.
1059     * It is already on the pid list and in /proc.
1060     * Increment the per uid process count (upcount).
1061     */
1062     nproc++;
1063     upcount_inc(ruid, zoneid);

1065     cp->p_next = practive;
1066     practive->p_prev = cp;
1067     practive = cp;

1069     cp->p_ignore = pp->p_ignore;
1070     cp->p_siginfo = pp->p_siginfo;
1071     cp->p_flag = pp->p_flag & (SJCTL|SNOWAIT|SNOCD);
1072     cp->p_sessp = pp->p_sessp;
1073     sess_hold(pp);
1074     cp->p_brand = pp->p_brand;
1075     if (PROC_IS_BRANDED(pp))
1076         BROP(pp)->b_copy_procddata(cp, pp);
1077     cp->p_bssbase = pp->p_bssbase;
1078     cp->p_brkbase = pp->p_brkbase;
1079     cp->p_brksize = pp->p_brksize;
1080     cp->p_brkpageszc = pp->p_brkpageszc;
1081     cp->p_stksize = pp->p_stksize;
1082     cp->p_stkpageszc = pp->p_stkpageszc;
1083     cp->p_stkprot = pp->p_stkprot;
1084     cp->p_datprot = pp->p_datprot;
1085     cp->p_usrstack = pp->p_usrstack;
1086     cp->p_model = pp->p_model;
1087     cp->p_ppid = pp->p_pid;
1088     cp->p_ancpid = pp->p_pid;
1089     cp->p_portcnt = pp->p_portcnt;
1090     /*
1091     * Security flags are preserved on fork, the inherited copy come into
1092     * effect on exec
1093     */
1094     cp->p_secflags = pp->p_secflags;
1095 #endif /* ! codereview */

1097     /*
1098     * Initialize watchpoint structures
1099     */
1100     avl_create(&cp->p_warea, wa_compare, sizeof (struct watched_area),
1101         offsetof(struct watched_area, wa_link));

1103     /*
1104     * Initialize immediate resource control values.
1105     */
1106     cp->p_stk_ctl = pp->p_stk_ctl;
1107     cp->p_fsz_ctl = pp->p_fsz_ctl;
1108     cp->p_vmem_ctl = pp->p_vmem_ctl;
1109     cp->p_fno_ctl = pp->p_fno_ctl;

1111     /*

```

```

1112     * Link up to parent-child-sibling chain. No need to lock
1113     * in general since only a call to freeproc() (done by the
1114     * same parent as newproc()) diddles with the child chain.
1115     */
1116     cp->p_sibling = pp->p_child;
1117     if (pp->p_child)
1118         pp->p_child->p_psibling = cp;

1120     cp->p_parent = pp;
1121     pp->p_child = cp;

1123     cp->p_child_ns = NULL;
1124     cp->p_sibling_ns = NULL;

1126     cp->p_nextorph = pp->p_orphan;
1127     cp->p_nextofkin = pp;
1128     pp->p_orphan = cp;

1130     /*
1131     * Inherit profiling state; do not inherit REALPROF profiling state.
1132     */
1133     cp->p_prof = pp->p_prof;
1134     cp->p_rprof_cyclic = CYCLIC_NONE;

1136     /*
1137     * Inherit pool pointer from the parent. Kernel processes are
1138     * always bound to the default pool.
1139     */
1140     mutex_enter(&pp->p_lock);
1141     if (flags & GETPROC_KERNEL) {
1142         cp->p_pool = pool_default;
1143         cp->p_flag |= SSYS;
1144     } else {
1145         cp->p_pool = pp->p_pool;
1146     }
1147     atomic_inc_32(&cp->p_pool->pool_ref);
1148     mutex_exit(&pp->p_lock);

1150     /*
1151     * Add the child process to the current task. Kernel processes
1152     * are always attached to task0.
1153     */
1154     mutex_enter(&cp->p_lock);
1155     if (flags & GETPROC_KERNEL)
1156         task_attach(task0p, cp);
1157     else
1158         task_attach(pp->p_task, cp);
1159     mutex_exit(&cp->p_lock);
1160     mutex_exit(&pidlock);

1162     avl_create(&cp->p_ct_held, contract_compar, sizeof (contract_t),
1163         offsetof(contract_t, ct_ctlist));

1165     /*
1166     * Duplicate any audit information kept in the process table
1167     */
1168     if (audit_active) /* copy audit data to cp */
1169         audit_newproc(cp);

1171     crhold(cp->p_cred = cr);

1173     /*
1174     * Bump up the counts on the file structures pointed at by the
1175     * parent's file table since the child will point at them too.
1176     */
1177     fcnt_add(P_FINFO(pp), 1);

```

```

1179     if (PTOU(pp)->u_cdir) {
1180         VN_HOLD(PTOU(pp)->u_cdir);
1181     } else {
1182         ASSERT(pp == &p0);
1183         /*
1184          * We must be at or before vfs_mountroot(); it will take care of
1185          * assigning our current directory.
1186          */
1187     }
1188     if (PTOU(pp)->u_rdir)
1189         VN_HOLD(PTOU(pp)->u_rdir);
1190     if (PTOU(pp)->u_cwd)
1191         refstr_hold(PTOU(pp)->u_cwd);
1192
1193     /*
1194      * copy the parent's uarea.
1195      */
1196     uarea = PTOU(cp);
1197     bcopy(PTOU(pp), uarea, sizeof (*uarea));
1198     flist_fork(P_FINFO(pp), P_FINFO(cp));
1199
1200     gethrestime(&uarea->u_start);
1201     uarea->u_ticks = ddi_get_lbolt();
1202     uarea->u_mem = rm_asrss(pp->p_as);
1203     uarea->u_acflag = AFORK;
1204
1205     /*
1206      * If inherit-on-fork, copy /proc tracing flags to child.
1207      */
1208     if ((pp->p_proc_flag & P_PR_FORK) != 0) {
1209         cp->p_proc_flag |= pp->p_proc_flag & (P_PR_TRACE|P_PR_FORK);
1210         cp->p_sigmask = pp->p_sigmask;
1211         cp->p_fltmask = pp->p_fltmask;
1212     } else {
1213         sigemptyset(&cp->p_sigmask);
1214         premtypset(&cp->p_fltmask);
1215         uarea->u_systrap = 0;
1216         premtypset(&uarea->u_entrymask);
1217         premtypset(&uarea->u_exitmask);
1218     }
1219     /*
1220      * If microstate accounting is being inherited, mark child
1221      */
1222     if ((pp->p_flag & SMSFORK) != 0)
1223         cp->p_flag |= pp->p_flag & (SMSFORK|SMSACCT);
1224
1225     /*
1226      * Inherit fixalignment flag from the parent
1227      */
1228     cp->p_fixalignment = pp->p_fixalignment;
1229
1230     *cpp = cp;
1231     return (0);
1232
1233 bad:
1234     ASSERT(MUTEX_NOT_HELD(&pidlock));
1235
1236     mutex_destroy(&cp->p_crlock);
1237     mutex_destroy(&cp->p_plock);
1238 #if defined(__x86)
1239     mutex_destroy(&cp->p_ldtlock);
1240 #endif
1241     if (newpid != -1) {
1242         proc_entry_free(cp->p_pidp);
1243         (void) pid_rele(cp->p_pidp);

```

```

1244     }
1245     kmem_cache_free(process_cache, cp);
1246
1247     mutex_enter(&zone->zone_nlwps_lock);
1248     task->tk_nprocs--;
1249     proj->kpj_nprocs--;
1250     zone->zone_nprocs--;
1251     mutex_exit(&zone->zone_nlwps_lock);
1252     atomic_inc_32(&zone->zone_ffnoprocs);
1253
1254 punish:
1255     /*
1256      * We most likely got into this situation because some process is
1257      * forking out of control. As punishment, put it to sleep for a
1258      * bit so it can't eat the machine alive. Sleep interval is chosen
1259      * to allow no more than one fork failure per cpu per clock tick
1260      * on average (yes, I just made this up). This has two desirable
1261      * properties: (1) it sets a constant limit on the fork failure
1262      * rate, and (2) the busier the system is, the harsher the penalty
1263      * for abusing it becomes.
1264      */
1265     INCR_COUNT(&fork_fail_pending, &pidlock);
1266     delay(fork_fail_pending / ncpu + 1);
1267     DECR_COUNT(&fork_fail_pending, &pidlock);
1268
1269     return (-1); /* out of memory or proc slots */
1270 }
1271
1272 /*
1273  * Release virtual memory.
1274  * In the case of vfork(), the child was given exclusive access to its
1275  * parent's address space. The parent is waiting in vfwait() for the
1276  * child to release its exclusive claim via relvm().
1277  */
1278 void
1279 relvm()
1280 {
1281     proc_t *p = curproc;
1282
1283     ASSERT((unsigned)p->p_lwpcnt <= 1);
1284
1285     prrelvm(); /* inform /proc */
1286
1287     if (p->p_flag & SVFORK) {
1288         proc_t *pp = p->p_parent;
1289         /*
1290          * The child process is either exec'ing or exit'ing.
1291          * The child is now separated from the parent's address
1292          * space. The parent process is made dispatchable.
1293          */
1294         /* This is a delicate locking maneuver, involving
1295          * both the parent's p_lock and the child's p_lock.
1296          * As soon as the SVFORK flag is turned off, the
1297          * parent is free to run, but it must not run until
1298          * we wake it up using its p_cv because it might
1299          * exit and we would be referencing invalid memory.
1300          * Therefore, we hold the parent with its p_lock
1301          * while protecting our p_flags with our own p_lock.
1302          */
1303         try_again:
1304             mutex_enter(&p->p_lock); /* grab child's lock first */
1305             prbarrier(p); /* make sure /proc is blocked out */
1306             mutex_enter(&pp->p_lock);
1307
1308         /*
1309          * Check if parent is locked by /proc.

```



```

1310     */
1311     if (pp->p_proc_flag & P_PR_LOCK) {
1312         /*
1313          * Delay until /proc is done with the parent.
1314          * We must drop our (the child's) p->p_lock, wait
1315          * via prbarrier() on the parent, then start over.
1316          */
1317         mutex_exit(&p->p_lock);
1318         prbarrier(pp);
1319         mutex_exit(&pp->p_lock);
1320         goto try_again;
1321     }
1322     p->p_flag &= ~SVFORK;
1323     kpreempt_disable();
1324     p->p_as = &kas;
1325
1326     /*
1327     * notify hat of change in thread's address space
1328     */
1329     hat_thread_exit(curthread);
1330     kpreempt_enable();
1331
1332     /*
1333     * child sizes are copied back to parent because
1334     * child may have grown.
1335     */
1336     pp->p_brkbase = p->p_brkbase;
1337     pp->p_brksize = p->p_brksize;
1338     pp->p_stksize = p->p_stksize;
1339
1340     /*
1341     * Copy back the shm accounting information
1342     * to the parent process.
1343     */
1344     pp->p_segacct = p->p_segacct;
1345     p->p_segacct = NULL;
1346
1347     /*
1348     * The parent is no longer waiting for the vfork()d child.
1349     * Restore the parent's watched pages, if any. This is
1350     * safe because we know the parent is not locked by /proc
1351     */
1352     pp->p_flag &= ~SVFWAIT;
1353     if (avl_numnodes(&pp->p_wpage) != 0) {
1354         pp->p_as->a_wpage = pp->p_wpage;
1355         avl_create(&pp->p_wpage, wp_compare,
1356                 sizeof (struct watched_page),
1357                 offsetof(struct watched_page, wp_link));
1358     }
1359     cv_signal(&pp->p_cv);
1360     mutex_exit(&pp->p_lock);
1361     mutex_exit(&p->p_lock);
1362 } else {
1363     if (p->p_as != &kas) {
1364         struct as *as;
1365
1366         if (p->p_segacct)
1367             shmexit(p);
1368
1369         /*
1370          * We grab p_lock for the benefit of /proc
1371          */
1372         kpreempt_disable();
1373         mutex_enter(&p->p_lock);
1374         prbarrier(p); /* make sure /proc is blocked out */
1375         as = p->p_as;

```

```

1376         p->p_as = &kas;
1377         mutex_exit(&p->p_lock);
1378
1379         /*
1380          * notify hat of change in thread's address space
1381          */
1382         hat_thread_exit(curthread);
1383         kpreempt_enable();
1384
1385         as_free(as);
1386         p->p_tr_lgrpid = LGRP_NONE;
1387     }
1388 }
1389 }
1390
1391 /*
1392 * Wait for child to exec or exit.
1393 * Called by parent of vfork'ed process.
1394 * See important comments in relvm(), above.
1395 */
1396 void
1397 vfwait(pid_t pid)
1398 {
1399     int signalled = 0;
1400     proc_t *pp = ttoproc(curthread);
1401     proc_t *cp;
1402
1403     /*
1404     * Wait for child to exec or exit.
1405     */
1406     for (;;) {
1407         mutex_enter(&pidlock);
1408         cp = prfind(pid);
1409         if (cp == NULL || cp->p_parent != pp) {
1410             /*
1411              * Child has exit()ed.
1412              */
1413             mutex_exit(&pidlock);
1414             break;
1415         }
1416         /*
1417          * Grab the child's p_lock before releasing pidlock.
1418          * Otherwise, the child could exit and we would be
1419          * referencing invalid memory.
1420          */
1421         mutex_enter(&cp->p_lock);
1422         mutex_exit(&pidlock);
1423         if (!(cp->p_flag & SVFORK)) {
1424             /*
1425              * Child has exec()ed or is exit()ing.
1426              */
1427             mutex_exit(&cp->p_lock);
1428             break;
1429         }
1430         mutex_enter(&pp->p_lock);
1431         mutex_exit(&cp->p_lock);
1432         /*
1433          * We might be waked up spuriously from the cv_wait().
1434          * We have to do the whole operation over again to be
1435          * sure the child's SVFORK flag really is turned off.
1436          * We cannot make reference to the child because it can
1437          * exit before we return and we would be referencing
1438          * invalid memory.
1439          *
1440          * Because this is potentially a very long-term wait,
1441          * we call cv_wait_sig() (for its jobcontrol and /proc

```

```
1442     * side-effects) unless there is a current signal, in
1443     * which case we use cv_wait() because we cannot return
1444     * from this function until the child has released the
1445     * address space. Calling cv_wait_sig() with a current
1446     * signal would lead to an indefinite loop here because
1447     * cv_wait_sig() returns immediately in this case.
1448     */
1449     if (signalled)
1450         cv_wait(&pp->p_cv, &pp->p_lock);
1451     else
1452         signalled = !cv_wait_sig(&pp->p_cv, &pp->p_lock);
1453     mutex_exit(&pp->p_lock);
1454 }
1455
1456 /* restore watchpoints to parent */
1457 if (pr_watch_active(pp)) {
1458     struct as *as = pp->p_as;
1459     AS_LOCK_ENTER(as, RW_WRITER);
1460     as_setwatch(as);
1461     AS_LOCK_EXIT(as);
1462 }
1463
1464 mutex_enter(&pp->p_lock);
1465 prbarrier(pp); /* barrier against /proc locking */
1466 continuelwps(pp);
1467 mutex_exit(&pp->p_lock);
1468 }
```

```

*****
26471 Thu Jun 30 22:03:28 2016
new/usr/src/uts/common/os/grow.c
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */

22 /* Copyright 2013 OmniTI Computer Consulting, Inc. All rights reserved. */

24 /*
25  * Copyright 2009 Sun Microsystems, Inc. All rights reserved.
26  * Use is subject to license terms.
27 */

29 /*      Copyright (c) 1984, 1986, 1987, 1988, 1989 AT&T */
30 /*      All Rights Reserved      */

32 #include <sys/types.h>
33 #include <sys/inttypes.h>
34 #include <sys/param.h>
35 #include <sys/symmacros.h>
36 #include <sys/system.h>
37 #include <sys/signal.h>
38 #include <sys/user.h>
39 #include <sys/errno.h>
40 #include <sys/var.h>
41 #include <sys/proc.h>
42 #include <sys/tuneable.h>
43 #include <sys/debug.h>
44 #include <sys/cmn_err.h>
45 #include <sys/cred.h>
46 #include <sys/vnode.h>
47 #include <sys/vfs.h>
48 #include <sys/vm.h>
49 #include <sys/file.h>
50 #include <sys/mman.h>
51 #include <sys/vmparam.h>
52 #include <sys/fcntl.h>
53 #include <sys/lwpchan_impl.h>
54 #include <sys/nbmlck.h>

56 #include <vm/hat.h>
57 #include <vm/as.h>
58 #include <vm/seg.h>

```

```

59 #include <vm/seg_dev.h>
60 #include <vm/seg_vn.h>

62 int use_brk_lpg = 1;
63 int use_stk_lpg = 1;

65 static int brk_lpg(caddr_t nva);
66 static int grow_lpg(caddr_t sp);

68 intptr_t
69 int
70 brk(caddr_t nva)
71 {
72     int error;
73     proc_t *p = curproc;

74     /*
75      * As a special case to aid the implementation of sbrk(3C), if given a
76      * new brk of 0, return the current brk. We'll hide this in brk(3C).
77      */
78     if (nva == 0)
79         return ((intptr_t)(p->p_brkbase + p->p_brksize));

81     /*
82      *#endif /* ! codereview */
83      * Serialize brk operations on an address space.
84      * This also serves as the lock protecting p_brksize
85      * and p_brkpageszc.
86      */
87     as_rangelock(p->p_as);
88     if (use_brk_lpg && (p->p_flag & SAUTOLPG) != 0) {
89         error = brk_lpg(nva);
90     } else {
91         error = brk_internal(nva, p->p_brkpageszc);
92     }
93     as_rangeunlock(p->p_as);
94     return ((error != 0 ? set_errno(error) : 0));
95 }

97 /*
98  * Algorithm: call arch-specific map_pgsz to get best page size to use,
99  * then call brk_internal().
100  * Returns 0 on success.
101  */
102 static int
103 brk_lpg(caddr_t nva)
104 {
105     struct proc *p = curproc;
106     size_t pgsz, len;
107     caddr_t addr, brkend;
108     caddr_t bssbase = p->p_bssbase;
109     caddr_t brkbase = p->p_brkbase;
110     int oszc, szc;
111     int err;

113     oszc = p->p_brkpageszc;

115     /*
116      * If p_brkbase has not yet been set, the first call
117      * to brk_internal() will initialize it.
118      */
119     if (brkbase == 0) {
120         return (brk_internal(nva, oszc));
121     }

123     len = nva - bssbase;

```

```

125     pgsz = map_pgsz(MAPPGSZ_HEAP, p, bssbase, len, 0);
126     szc = page_szc(pgsz);

128     /*
129     * Covers two cases:
130     * 1. page_szc() returns -1 for invalid page size, so we want to
131     * ignore it in that case.
132     * 2. By design we never decrease page size, as it is more stable.
133     */
134     if (szc <= oszc) {
135         err = brk_internal(nva, oszc);
136         /* If failed, back off to base page size. */
137         if (err != 0 && oszc != 0) {
138             err = brk_internal(nva, 0);
139         }
140         return (err);
141     }

143     err = brk_internal(nva, szc);
144     /* If using szc failed, map with base page size and return. */
145     if (err != 0) {
146         if (szc != 0) {
147             err = brk_internal(nva, 0);
148         }
149         return (err);
150     }

152     /*
153     * Round up brk base to a large page boundary and remap
154     * anything in the segment already faulted in beyond that
155     * point.
156     */
157     addr = (caddr_t)P2ROUNDUP((uintptr_t)p->p_bssbase, pgsz);
158     brkend = brkbase + p->p_brksize;
159     len = brkend - addr;
160     /* Check that len is not negative. Update page size code for heap. */
161     if (addr >= p->p_bssbase && brkend > addr && IS_P2ALIGNED(len, pgsz)) {
162         (void) as_setpagesize(p->p_as, addr, len, szc, B_FALSE);
163         p->p_brkpageszc = szc;
164     }

166     ASSERT(err == 0);
167     return (err);          /* should always be 0 */
168 }

170 /*
171 * Returns 0 on success.
172 */
173 int
174 brk_internal(caddr_t nva, uint_t brkszc)
175 {
176     caddr_t ova;          /* current break address */
177     size_t size;
178     int error;
179     struct proc *p = curproc;
180     struct as *as = p->p_as;
181     size_t pgsz;
182     uint_t szc;
183     rctl_qty_t as_rctl;

185     /*
186     * extend heap to brkszc alignment but use current p->p_brkpageszc
187     * for the newly created segment. This allows the new extension
188     * segment to be concatenated successfully with the existing brk
189     * segment.

```

```

190     */
191     if ((szc = brkszc) != 0) {
192         pgsz = page_get_pagesize(szc);
193         ASSERT(pgsz > PAGE_SIZE);
194     } else {
195         pgsz = PAGE_SIZE;
196     }

198     mutex_enter(&p->p_lock);
199     as_rctl = rctl_enforced_value(rctlproc_legacy[RLIMIT_DATA],
200     p->p_rctls, p);
201     mutex_exit(&p->p_lock);

203     /*
204     * If p_brkbase has not yet been set, the first call
205     * to brk() will initialize it.
206     */
207     if (p->p_brkbase == 0)
208         p->p_brkbase = nva;

210     /*
211     * Before multiple page size support existed p_brksize was the value
212     * not rounded to the pagesize (i.e. it stored the exact user request
213     * for heap size). If pgsz is greater than PAGE_SIZE calculate the
214     * heap size as the real new heap size by rounding it up to pgsz.
215     * This is useful since we may want to know where the heap ends
216     * without knowing heap pagesize (e.g. some old code) and also if
217     * heap pagesize changes we can update p_brkpageszc but delay adding
218     * new mapping yet still know from p_brksize where the heap really
219     * ends. The user requested heap end is stored in libc variable.
220     */
221     if (pgsz > PAGE_SIZE) {
222         caddr_t tnva = (caddr_t)P2ROUNDUP((uintptr_t)nva, pgsz);
223         size = tnva - p->p_brkbase;
224         if (tnva < p->p_brkbase || (size > p->p_brksize &&
225         size > (size_t)as_rctl)) {
226             szc = 0;
227             pgsz = PAGE_SIZE;
228             size = nva - p->p_brkbase;
229         }
230     } else {
231         size = nva - p->p_brkbase;
232     }

234     /*
235     * use PAGE_SIZE to roundup ova because we want to know the real value
236     * of the current heap end in case p_brkpageszc changes since the last
237     * p_brksize was computed.
238     */
239     nva = (caddr_t)P2ROUNDUP((uintptr_t)nva, pgsz);
240     ova = (caddr_t)P2ROUNDUP((uintptr_t)(p->p_brkbase + p->p_brksize),
241     PAGE_SIZE);

243     if ((nva < p->p_brkbase) || (size > p->p_brksize &&
244     size > as_rctl)) {
245         mutex_enter(&p->p_lock);
246         (void) rctl_action(rctlproc_legacy[RLIMIT_DATA], p->p_rctls, p,
247         RCA_SAFE);
248         mutex_exit(&p->p_lock);
249         return (ENOMEM);
250     }

252     if (nva > ova) {
253         struct segvn_crargs crargs =
254             SEGVN_ZFOD_ARGS(PROT_ZFOD, PROT_ALL);

```

```

256     if (!(p->p_datprot & PROT_EXEC)) {
257         crargs.prot &= ~PROT_EXEC;
258     }
259
260     /*
261     * Add new zfod mapping to extend UNIX data segment
262     * AS_MAP_NO_LPOOB means use 0, and don't reapply OOB policies
263     * via map_pgszvec(). Use AS_MAP_HEAP to get intermediate
264     * page sizes if ova is not aligned to szc's pgsz.
265     */
266     if (szc > 0) {
267         caddr_t rbss;
268
269         rbss = (caddr_t)P2ROUNDUP((uintptr_t)p->p_bssbase,
270             pgsz);
271         if (IS_P2ALIGNED(p->p_bssbase, pgsz) || ova > rbss) {
272             crargs.szc = p->p_brkpageszc ? p->p_brkpageszc :
273                 AS_MAP_NO_LPOOB;
274         } else if (ova == rbss) {
275             crargs.szc = szc;
276         } else {
277             crargs.szc = AS_MAP_HEAP;
278         }
279     } else {
280         crargs.szc = AS_MAP_NO_LPOOB;
281     }
282     crargs.lgrp_mem_policy_flags = LGRP_MP_FLAG_EXTEND_UP;
283     error = as_map(as, ova, (size_t)(nva - ova), segvn_create,
284         &crargs);
285     if (error) {
286         return (error);
287     }
288
289 } else if (nva < ova) {
290     /*
291     * Release mapping to shrink UNIX data segment.
292     */
293     (void) as_unmap(as, nva, (size_t)(ova - nva));
294 }
295 p->p_brksize = size;
296 return (0);
297 }
298
299 /*
300 * Grow the stack to include sp. Return 1 if successful, 0 otherwise.
301 * This routine assumes that the stack grows downward.
302 */
303 int
304 grow(caddr_t sp)
305 {
306     struct proc *p = curproc;
307     struct as *as = p->p_as;
308     size_t oldsize = p->p_stksize;
309     size_t newsize;
310     int err;
311
312     /*
313     * Serialize grow operations on an address space.
314     * This also serves as the lock protecting p_stksize
315     * and p_stkpageszc.
316     */
317     as_rangelock(as);
318     if (use_stk_lpg && (p->p_flag & SAUTOLPG) != 0) {
319         err = grow_lpg(sp);
320     } else {
321         err = grow_internal(sp, p->p_stkpageszc);

```

```

322     }
323     as_rangeunlock(as);
324
325     if (err == 0 && (newsize = p->p_stksize) > oldsize) {
326         ASSERT(IS_P2ALIGNED(oldsize, PAGESIZE));
327         ASSERT(IS_P2ALIGNED(newsize, PAGESIZE));
328         /*
329         * Set up translations so the process doesn't have to fault in
330         * the stack pages we just gave it.
331         */
332         (void) as_fault(as->a_hat, as, p->p_usrstack - newsize,
333             newsize - oldsize, F_INVALID, S_WRITE);
334     }
335     return ((err == 0 ? 1 : 0));
336 }
337
338 /*
339 * Algorithm: call arch-specific map_pgsz to get best page size to use,
340 * then call grow_internal().
341 * Returns 0 on success.
342 */
343 static int
344 grow_lpg(caddr_t sp)
345 {
346     struct proc *p = curproc;
347     size_t pgsz;
348     size_t len, newsize;
349     caddr_t addr, saddr;
350     caddr_t growend;
351     int oszc, szc;
352     int err;
353
354     newsize = p->p_usrstack - sp;
355
356     oszc = p->p_stkpageszc;
357     pgsz = map_pgsz(MAPPGSZ_STK, p, sp, newsize, 0);
358     szc = page_szc(pgsz);
359
360     /*
361     * Covers two cases:
362     * 1. page_szc() returns -1 for invalid page size, so we want to
363     * ignore it in that case.
364     * 2. By design we never decrease page size, as it is more stable.
365     * This shouldn't happen as the stack never shrinks.
366     */
367     if (szc <= oszc) {
368         err = grow_internal(sp, oszc);
369         /* failed, fall back to base page size */
370         if (err != 0 && oszc != 0) {
371             err = grow_internal(sp, 0);
372         }
373         return (err);
374     }
375
376     /*
377     * We've grown sufficiently to switch to a new page size.
378     * So we are going to remap the whole segment with the new page size.
379     */
380     err = grow_internal(sp, szc);
381     /* The grow with szc failed, so fall back to base page size. */
382     if (err != 0) {
383         if (szc != 0) {
384             err = grow_internal(sp, 0);
385         }
386         return (err);
387     }

```

```

389  /*
390  * Round up stack pointer to a large page boundary and remap
391  * any pgsz pages in the segment already faulted in beyond that
392  * point.
393  */
394  saddr = p->p_usrstack - p->p_stksize;
395  addr = (caddr_t)P2ROUNDUP((uintptr_t)saddr, pgsz);
396  growend = (caddr_t)P2ALIGN((uintptr_t)p->p_usrstack, pgsz);
397  len = growend - addr;
398  /* Check that len is not negative. Update page size code for stack. */
399  if (addr >= saddr && growend > addr && IS_P2ALIGNED(len, pgsz)) {
400      (void) as_setpagesize(p->p_as, addr, len, szc, B_FALSE);
401      p->p_stkpageszc = szc;
402  }

404  ASSERT(err == 0);
405  return (err);          /* should always be 0 */
406 }

408 /*
409 * This routine assumes that the stack grows downward.
410 * Returns 0 on success, errno on failure.
411 */
412 int
413 grow_internal(caddr_t sp, uint_t growszc)
414 {
415     struct proc *p = curproc;
416     size_t newsize;
417     size_t oldsize;
418     int error;
419     size_t pgsz;
420     uint_t szc;
421     struct segvn_crargs crargs = SEGVN_ZFOD_ARGS(PROT_ZFOD, PROT_ALL);

423     ASSERT(sp < p->p_usrstack);
424     sp = (caddr_t)P2ALIGN((uintptr_t)sp, PAGESIZE);

426     /*
427     * grow to growszc alignment but use current p->p_stkpageszc for
428     * the segvn_crargs szc passed to segvn_create. For memcntl to
429     * increase the szc, this allows the new extension segment to be
430     * concatenated successfully with the existing stack segment.
431     */
432     if ((szc = growszc) != 0) {
433         pgsz = page_get_pagesize(szc);
434         ASSERT(pgsz > PAGESIZE);
435         newsize = p->p_usrstack - (caddr_t)P2ALIGN((uintptr_t)sp, pgsz);
436         if (newsize > (size_t)p->p_stk_ctl) {
437             szc = 0;
438             pgsz = PAGESIZE;
439             newsize = p->p_usrstack - sp;
440         }
441     } else {
442         pgsz = PAGESIZE;
443         newsize = p->p_usrstack - sp;
444     }

446     if (newsize > (size_t)p->p_stk_ctl) {
447         (void) rctl_action(rctlproc_legacy[RLIMIT_STACK], p->p_rctls, p,
448             RCA_UNSAFE_ALL);

450         return (ENOMEM);
451     }

453     oldsize = p->p_stksize;

```

```

454     ASSERT(P2PHASE(oldsize, PAGESIZE) == 0);

456     if (newsize <= oldsize) {          /* prevent the stack from shrinking */
457         return (0);
458     }

460     if (!(p->p_stkprot & PROT_EXEC)) {
461         crargs.prot &= ~PROT_EXEC;
462     }
463     /*
464     * extend stack with the proposed new growszc, which is different
465     * than p_stkpageszc only on a memcntl to increase the stack pagesize.
466     * AS_MAP_NO_LPOOB means use 0, and don't reapply OOB policies via
467     * map_pgszvec(). Use AS_MAP_STACK to get intermediate page sizes
468     * if not aligned to szc's pgsz.
469     */
470     if (szc > 0) {
471         caddr_t oldsp = p->p_usrstack - oldsize;
472         caddr_t austk = (caddr_t)P2ALIGN((uintptr_t)p->p_usrstack,
473             pgsz);

475         if (IS_P2ALIGNED(p->p_usrstack, pgsz) || oldsp < austk) {
476             crargs.szc = p->p_stkpageszc ? p->p_stkpageszc :
477                 AS_MAP_NO_LPOOB;
478         } else if (oldsp == austk) {
479             crargs.szc = szc;
480         } else {
481             crargs.szc = AS_MAP_STACK;
482         }
483     } else {
484         crargs.szc = AS_MAP_NO_LPOOB;
485     }
486     crargs.lgrp_mem_policy_flags = LGRP_MP_FLAG_EXTEND_DOWN;

488     if ((error = as_map(p->p_as, p->p_usrstack - newsize, newsize - oldsize,
489         segvn_create, &crargs)) != 0) {
490         if (error == EAGAIN) {
491             cmn_err(CE_WARN, "Sorry, no swap space to grow stack "
492                 "for pid %d (%s)", p->p_pid, PTOU(p)->u_comm);
493         }
494         return (error);
495     }
496     p->p_stksize = newsize;
497     return (0);
498 }

500 /*
501 * Find address for user to map. If MAP_FIXED is not specified, we can pick
502 * any address we want, but we will first try the value in *addrp if it is
503 * non-NULL and MAP_RANDOMIZE is not set. Thus this is implementing a way to
504 * try and get a preferred address.
505 * Find address for user to map.
506 * If MAP_FIXED is not specified, we can pick any address we want, but we will
507 * first try the value in *addrp if it is non-NULL. Thus this is implementing
508 * a way to try and get a preferred address.
509 */
510 int
511 choose_addr(struct as *as, caddr_t *addrp, size_t len, offset_t off,
512     int vacalign, uint_t flags)
513 {
514     caddr_t basep = (caddr_t)((uintptr_t)((uintptr_t)*addrp & PAGEMASK) +
515         size_t)lenp = len;

517     ASSERT(AS_ISCLAIMGAP(as));          /* searches should be serialized */
518     if (flags & MAP_FIXED) {
519         (void) as_unmap(as, *addrp, len);

```

```

516         return (0);
517     } else if (basep != NULL &&
518             ((flags & (MAP_ALIGN | _MAP_RANDOMIZE)) == 0) &&
91     } else if (basep != NULL && ((flags & MAP_ALIGN) == 0) &&
519         !as_gap(as, len, &basep, &lenp, 0, *addrp)) {
520         /* User supplied address was available */
521         *addrp = basep;
522     } else {
523         /*
524          * No user supplied address or the address supplied was not
525          * available.
526          */
527         map_addr(addrp, len, off, vacalign, flags);
528     }
529     if (*addrp == NULL)
530         return (ENOMEM);
531     return (0);
532 }
    unchanged_portion_omitted

599 static int
600 smmap_common(caddr_t *addrp, size_t len,
601             int prot, int flags, struct file *fp, offset_t pos)
602 {
603     struct vnode *vp;
604     struct as *as = curproc->p_as;
605     uint_t uprot, maxprot, type;
606     int error;
607     int in_crit = 0;

609     if ((flags & ~(MAP_SHARED | MAP_PRIVATE | MAP_FIXED | _MAP_NEW |
610                 _MAP_LOW32 | MAP_NORESERVE | MAP_ANON | MAP_ALIGN |
611                 MAP_TEXT | MAP_INITDATA)) != 0) {
612         /* | MAP_RENAME */          /* not implemented, let user know */
613         return (EINVAL);
614     }

616     if ((flags & MAP_TEXT) && !(prot & PROT_EXEC)) {
617         return (EINVAL);
618     }

620     if ((flags & (MAP_TEXT | MAP_INITDATA)) == (MAP_TEXT | MAP_INITDATA)) {
621         return (EINVAL);
622     }

624     if ((flags & (MAP_FIXED | _MAP_RANDOMIZE)) ==
625         (MAP_FIXED | _MAP_RANDOMIZE)) {
626         return (EINVAL);
627     }

629     /*
630      * If it's not a fixed allocation and mmap ASLR is enabled, randomize
631      * it.
632      */
633     if (((flags & MAP_FIXED) == 0) &&
634         secflag_enabled(curproc, PROC_SEC_ASLR))
635         flags |= _MAP_RANDOMIZE;

637 #endif /* ! codereview */
638 #if defined(__sparc)
639     /*
640      * See if this is an "old mmap call".  If so, remember this
641      * fact and convert the flags value given to mmap to indicate
642      * the specified address in the system call must be used.
643      * _MAP_NEW is turned set by all new uses of mmap.
644      */

```

```

645     if ((flags & _MAP_NEW) == 0)
646         flags |= MAP_FIXED;
647 #endif
648     flags &= ~_MAP_NEW;

650     type = flags & MAP_TYPE;
651     if (type != MAP_PRIVATE && type != MAP_SHARED)
652         return (EINVAL);

655     if (flags & MAP_ALIGN) {
656         if (flags & MAP_FIXED)
657             return (EINVAL);

659         /* alignment needs to be a power of 2 >= page size */
660         if (((uintptr_t)*addrp < PAGE_SIZE && (uintptr_t)*addrp != 0) ||
661             !ISP2((uintptr_t)*addrp))
662             return (EINVAL);
663     }
664     /*
665      * Check for bad lengths and file position.
666      * We let the VOP_MAP routine check for negative lengths
667      * since on some vnode types this might be appropriate.
668      */
669     if (len == 0 || (pos & (u_offset_t)PAGEOFFSET) != 0)
670         return (EINVAL);

672     maxprot = PROT_ALL;          /* start out allowing all accesses */
673     uprot = prot | PROT_USER;

675     if (fp == NULL) {
676         ASSERT(flags & MAP_ANON);
677         /* discard lwpchan mappings, like munmap() */
678         if ((flags & MAP_FIXED) && curproc->p_lcp != NULL)
679             lwpchan_delete_mapping(curproc, *addrp, *addrp + len);
680         as_rangelock(as);
681         error = zmap(as, addrp, len, uprot, flags, pos);
682         as_rangeunlock(as);
683         /*
684          * Tell machine specific code that lwp has mapped shared memory
685          */
686         if (error == 0 && (flags & MAP_SHARED)) {
687             /* EMPTY */
688             LWP_MMODEL_SHARED_AS(*addrp, len);
689         }
690         return (error);
691     } else if ((flags & MAP_ANON) != 0)
692         return (EINVAL);

694     vp = fp->f_vnode;

696     /* Can't execute code from "noexec" mounted filesystem. */
697     if ((vp->v_vfsp->vfs_flag & VFS_NOEXEC) != 0)
698         maxprot &= ~PROT_EXEC;

700     /*
701      * These checks were added as part of large files.
702      *
703      * Return ENXIO if the initial position is negative; return EOVERFLOW
704      * if (offset + len) would overflow the maximum allowed offset for the
705      * type of file descriptor being used.
706      */
707     if (vp->v_type == VREG) {
708         if (pos < 0)
709             return (ENXIO);

```

```

710         if ((offset_t)len > (OFFSET_MAX(fp) - pos))
711             return (EOVERFLOW);
712     }
713
714     if (type == MAP_SHARED && (fp->f_flag & FWRITE) == 0) {
715         /* no write access allowed */
716         maxprot &= ~PROT_WRITE;
717     }
718
719     /*
720     * XXX - Do we also adjust maxprot based on protections
721     * of the vnode? E.g. if no execute permission is given
722     * on the vnode for the current user, maxprot probably
723     * should disallow PROT_EXEC also? This is different
724     * from the write access as this would be a per vnode
725     * test as opposed to a per fd test for writability.
726     */
727
728     /*
729     * Verify that the specified protections are not greater than
730     * the maximum allowable protections. Also test to make sure
731     * that the file descriptor does allow for read access since
732     * "write only" mappings are hard to do since normally we do
733     * the read from the file before the page can be written.
734     */
735     if (((maxprot & uprot) != uprot) || (fp->f_flag & FREAD) == 0)
736         return (EACCES);
737
738     /*
739     * If the user specified an address, do some simple checks here
740     */
741     if ((flags & MAP_FIXED) != 0) {
742         caddr_t userlimit;
743
744         /*
745         * Use the user address. First verify that
746         * the address to be used is page aligned.
747         * Then make some simple bounds checks.
748         */
749         if (((uintptr_t)*addrp & PAGEOFFSET) != 0)
750             return (EINVAL);
751
752         userlimit = flags & _MAP_LOW32 ?
753             (caddr_t)USERLIMIT32 : as->a_userlimit;
754         switch (valid_usr_range(*addrp, len, uprot, as, userlimit)) {
755             case RANGE_OKAY:
756                 break;
757             case RANGE_BADPROT:
758                 return (ENOTSUP);
759             case RANGE_BADADDR:
760             default:
761                 return (ENOMEM);
762         }
763     }
764
765     if ((prot & (PROT_READ | PROT_WRITE | PROT_EXEC)) &&
766         nbl_need_check(vp)) {
767         int svmand;
768         nbl_op_t nop;
769
770         nbl_start_crit(vp, RW_READER);
771         in_crit = 1;
772         error = nbl_svmand(vp, fp->f_cred, &svmand);
773         if (error != 0)
774             goto done;
775         if ((prot & PROT_WRITE) && (type == MAP_SHARED)) {

```

```

776         if (prot & (PROT_READ | PROT_EXEC)) {
777             nop = NBL_READWRITE;
778         } else {
779             nop = NBL_WRITE;
780         }
781     } else {
782         nop = NBL_READ;
783     }
784     if (nbl_conflict(vp, nop, 0, LONG_MAX, svmand, NULL)) {
785         error = EACCES;
786         goto done;
787     }
788 }
789
790 /* discard lwpchan mappings, like munmap() */
791 if ((flags & MAP_FIXED) && curproc->p_lcp != NULL)
792     lwpchan_delete_mapping(curproc, *addrp, *addrp + len);
793
794 /*
795 * Ok, now let the vnode map routine do its thing to set things up.
796 */
797 error = VOP_MAP(vp, pos, as,
798               addrp, len, uprot, maxprot, flags, fp->f_cred, NULL);
799
800 if (error == 0) {
801     /*
802     * Tell machine specific code that lwp has mapped shared memory
803     */
804     if (flags & MAP_SHARED) {
805         /* EMPTY */
806         LWP_MMODEL_SHARED_AS(*addrp, len);
807     }
808     if (vp->v_type == VREG &&
809         (flags & (MAP_TEXT | MAP_INITDATA)) != 0) {
810         /*
811         * Mark this as an executable vnode
812         */
813         mutex_enter(&vp->v_lock);
814         vp->v_flag |= VVMEXEC;
815         mutex_exit(&vp->v_lock);
816     }
817 }
818
819 done:
820     if (in_crit)
821         nbl_end_crit(vp);
822     return (error);
823 }

```

unchanged portion omitted

new/usr/src/uts/common/os/mmapobj.c

1

```
*****
69576 Thu Jun 30 22:03:29 2016
new/usr/src/uts/common/os/mmapobj.c
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
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 * Copyright 2014 Joyent, Inc. All rights reserved.
25 */

27 #include <sys/types.h>
28 #include <sys/sysmacros.h>
29 #include <sys/kmem.h>
30 #include <sys/param.h>
31 #include <sys/system.h>
32 #include <sys/errno.h>
33 #include <sys/mman.h>
34 #include <sys/cmn_err.h>
35 #include <sys/cred.h>
36 #include <sys/vmsystem.h>
37 #include <sys/machsystem.h>
38 #include <sys/debug.h>
39 #include <vm/as.h>
40 #include <vm/seg.h>
41 #include <sys/vmparam.h>
42 #include <sys/vfs.h>
43 #include <sys/elf.h>
44 #include <sys/machelf.h>
45 #include <sys/corect1.h>
46 #include <sys/exec.h>
47 #include <sys/exechdr.h>
48 #include <sys/autoconf.h>
49 #include <sys/mem.h>
50 #include <vm/seg_dev.h>
51 #include <sys/vmparam.h>
52 #include <sys/mmapobj.h>
53 #include <sys/atomic.h>

55 /*
56  * Theory statement:
57  *
58  * The main driving force behind mmapobj is to interpret and map ELF files
```

new/usr/src/uts/common/os/mmapobj.c

2

```
59  * inside of the kernel instead of having the linker be responsible for this.
60  *
61  * mmapobj also supports the AOUT 4.x binary format as well as flat files in
62  * a read only manner.
63  *
64  * When interpreting and mapping an ELF file, mmapobj will map each PT_LOAD
65  * or PT_SUNWBSS segment according to the ELF standard. Refer to the "Linker
66  * and Libraries Guide" for more information about the standard and mapping
67  * rules.
68  *
69  * Having mmapobj interpret and map objects will allow the kernel to make the
70  * best decision for where to place the mappings for said objects. Thus, we
71  * can make optimizations inside of the kernel for specific platforms or cache
72  * mapping information to make mapping objects faster. The cache is ignored
73  * if ASLR is enabled.
74  *
75  * can make optimizations inside of the kernel for specific platforms or
76  * cache mapping information to make mapping objects faster.
77  *
78  * The lib_va_hash will be one such optimization. For each ELF object that
79  * mmapobj is asked to interpret, we will attempt to cache the information
80  * about the PT_LOAD and PT_SUNWBSS sections to speed up future mappings of
81  * the same objects. We will cache up to LIBVA_CACHED_SEGS (see below) program
82  * headers which should cover a majority of the libraries out there without
83  * wasting space. In order to make sure that the cached information is valid,
84  * we check the passed in vnode's mtime and ctime to make sure the vnode
85  * has not been modified since the last time we used it.
86  *
87  * In addition, the lib_va_hash may contain a preferred starting VA for the
88  * object which can be useful for platforms which support a shared context.
89  * This will increase the likelihood that library text can be shared among
90  * many different processes. We limit the reserved VA space for 32 bit objects
91  * in order to minimize fragmenting the processes address space.
92  *
93  * In addition to the above, the mmapobj interface allows for padding to be
94  * requested before the first mapping and after the last mapping created.
95  * When padding is requested, no additional optimizations will be made for
96  * that request.
97  */

98 /*
99  * Threshold to prevent allocating too much kernel memory to read in the
100 * program headers for an object. If it requires more than below,
101 * we will use a KM_NOSLEEP allocation to allocate memory to hold all of the
102 * program headers which could possibly fail. If less memory than below is
103 * needed, then we use a KM_SLEEP allocation and are willing to wait for the
104 * memory if we need to.
105 */
106 size_t mmapobj_alloc_threshold = 65536;

107 /* Debug stats for test coverage */
108 #ifdef DEBUG
109 struct mobj_stats {
110     uint_t  mobjs_unmap_called;
111     uint_t  mobjs_remap_devnull;
112     uint_t  mobjs_lookup_start;
113     uint_t  mobjs_alloc_start;
114     uint_t  mobjs_alloc_vmem;
115     uint_t  mobjs_add_collision;
116     uint_t  mobjs_get_addr;
117     uint_t  mobjs_map_flat_no_padding;
118     uint_t  mobjs_map_flat_padding;
119     uint_t  mobjs_map_ptload_text;
120     uint_t  mobjs_map_ptload_initdata;
121     uint_t  mobjs_map_ptload_preload;
122     uint_t  mobjs_map_ptload_unaligned_text;
123     uint_t  mobjs_map_ptload_unaligned_map_fail;
```

```

123     uint_t  mobjs_map_ptload_unaligned_read_fail;
124     uint_t  mobjs_zfoddiff;
125     uint_t  mobjs_zfoddiff_nowrite;
126     uint_t  mobjs_zfodextra;
127     uint_t  mobjs_ptload_failed;
128     uint_t  mobjs_map_elf_no_holes;
129     uint_t  mobjs_unmap_hole;
130     uint_t  mobjs_nomem_header;
131     uint_t  mobjs_inval_header;
132     uint_t  mobjs_overlap_header;
133     uint_t  mobjs_np2_align;
134     uint_t  mobjs_np2_align_overflow;
135     uint_t  mobjs_exec_padding;
136     uint_t  mobjs_exec_addr_mapped;
137     uint_t  mobjs_exec_addr_devnull;
138     uint_t  mobjs_exec_addr_in_use;
139     uint_t  mobjs_lvp_found;
140     uint_t  mobjs_no_loadable_yet;
141     uint_t  mobjs_nothing_to_map;
142     uint_t  mobjs_e2big;
143     uint_t  mobjs_dyn_pad_align;
144     uint_t  mobjs_dyn_pad_noalign;
145     uint_t  mobjs_alloc_start_fail;
146     uint_t  mobjs_lvp_nocache;
147     uint_t  mobjs_extra_padding;
148     uint_t  mobjs_lvp_not_needed;
149     uint_t  mobjs_no_mem_map_sz;
150     uint_t  mobjs_check_exec_failed;
151     uint_t  mobjs_lvp_used;
152     uint_t  mobjs_wrong_model;
153     uint_t  mobjs_noexec_fs;
154     uint_t  mobjs_e2big_et_rel;
155     uint_t  mobjs_et_rel_mapped;
156     uint_t  mobjs_unknown_elf_type;
157     uint_t  mobjs_phent32_too_small;
158     uint_t  mobjs_phent64_too_small;
159     uint_t  mobjs_inval_elf_class;
160     uint_t  mobjs_too_many_phdrs;
161     uint_t  mobjs_no_phsize;
162     uint_t  mobjs_phsize_large;
163     uint_t  mobjs_phsize_xtralarge;
164     uint_t  mobjs_fast_wrong_model;
165     uint_t  mobjs_fast_e2big;
166     uint_t  mobjs_fast;
167     uint_t  mobjs_fast_success;
168     uint_t  mobjs_fast_not_now;
169     uint_t  mobjs_small_file;
170     uint_t  mobjs_read_error;
171     uint_t  mobjs_unsupported;
172     uint_t  mobjs_flat_e2big;
173     uint_t  mobjs_phent_align32;
174     uint_t  mobjs_phent_align64;
175     uint_t  mobjs_lib_va_find_hit;
176     uint_t  mobjs_lib_va_find_delay_delete;
177     uint_t  mobjs_lib_va_find_delete;
178     uint_t  mobjs_lib_va_add_delay_delete;
179     uint_t  mobjs_lib_va_add_delete;
180     uint_t  mobjs_lib_va_create_failure;
181     uint_t  mobjs_min_align;
182 #if defined(__sparc)
183     uint_t  mobjs_aout_uzero_fault;
184     uint_t  mobjs_aout_64bit_try;
185     uint_t  mobjs_aout_noexec;
186     uint_t  mobjs_aout_e2big;
187     uint_t  mobjs_aout_lib;
188     uint_t  mobjs_aout_fixed;

```

```

189     uint_t  mobjs_aout_zfoddiff;
190     uint_t  mobjs_aout_map_bss;
191     uint_t  mobjs_aout_bss_fail;
192     uint_t  mobjs_aout_nlist;
193     uint_t  mobjs_aout_addr_in_use;
194 #endif
195 } mobj_stats;

```

```

unchanged_portion_omitted

704 /*
705  * Get the starting address for a given file to be mapped and return it
706  * to the caller.  If we're using lib_va and we need to allocate an address,
707  * we will attempt to allocate it from the global reserved pool such that the
708  * same address can be used in the future for this file.  If we can't use the
709  * reserved address then we just get one that will fit in our address space.
710  *
711  * Returns the starting virtual address for the range to be mapped or NULL
712  * if an error is encountered.  If we successfully insert the requested info
713  * into the lib_va hash, then *lvpp will be set to point to this lib_va
714  * structure.  The structure will have a hold on it and thus lib_va_release
715  * needs to be called on it by the caller.  This function will not fill out
716  * lv_mps or lv_num_segs since it does not have enough information to do so.
717  * The caller is responsible for doing this making sure that any modifications
718  * to lv_mps are visible before setting lv_num_segs.
719  */
720 static caddr_t
721 mmabobj_alloc_start_addr(struct lib_va **lvpp, size_t len, int use_lib_va,
722     int randomize, size_t align, vattr_t *vap)
723 {
724     proc_t *p = curproc;
725     struct as *as = p->p_as;
726     struct segvn_crargs crargs = SEGVN_ZFOD_ARGS(PROT_USER, PROT_ALL);
727     int error;
728     model_t model;
729     uint_t ma_flags = _MAP_LOW32;
730     caddr_t base = NULL;
731     vmem_t *model_vmem;
732     size_t lib_va_start;
733     size_t lib_va_end;
734     size_t lib_va_len;

```

```

736     ASSERT(lvpp != NULL);
737     ASSERT((randomize & use_lib_va) != 1);
738 #endif /* ! codereview */

```

```

740     MOBJ_STAT_ADD(alloc_start);
741     model = get_udatamodel();

```

```

743     if (model == DATAMODEL_LP64) {
744         ma_flags = 0;
745         model_vmem = lib_va_64_arena;
746     } else {
747         ASSERT(model == DATAMODEL_ILP32);
748         model_vmem = lib_va_32_arena;
749     }

```

```

751     if (align > 1) {
752         ma_flags |= MAP_ALIGN;
753     }

```

```

755     if (randomize != 0)
756         ma_flags |= _MAP_RANDOMIZE;

```

```

758 #endif /* ! codereview */
759     if (use_lib_va) {

```

```

760      /*
761      * The first time through, we need to setup the lib_va arenas.
762      * We call map_addr to find a suitable range of memory to map
763      * the given library, and we will set the highest address
764      * in our vmem arena to the end of this address range.
765      * We allow up to half of the address space to be used
766      * for lib_va addresses but we do not prevent any allocations
767      * in this range from other allocation paths.
768      */
769      if (lib_va_64_arena == NULL && model == DATAMODEL_LP64) {
770          mutex_enter(&lib_va_init_mutex);
771          if (lib_va_64_arena == NULL) {
772              base = (caddr_t)align;
773              as_rangelock(as);
774              map_addr(&base, len, 0, 1, ma_flags);
775              as_rangeunlock(as);
776              if (base == NULL) {
777                  mutex_exit(&lib_va_init_mutex);
778                  MOBJ_STAT_ADD(lib_va_create_failure);
779                  goto nolibva;
780              }
781              lib_va_end = (size_t)base + len;
782              lib_va_len = lib_va_end >> 1;
783              lib_va_len = P2ROUNDUP(lib_va_len, PAGE_SIZE);
784              lib_va_start = lib_va_end - lib_va_len;
785
786              /*
787              * Need to make sure we avoid the address hole.
788              * We know lib_va_end is valid but we need to
789              * make sure lib_va_start is as well.
790              */
791              if ((lib_va_end > (size_t)hole_end) &&
792                  (lib_va_start < (size_t)hole_end)) {
793                  lib_va_start = P2ROUNDUP(
794                      (size_t)hole_end, PAGE_SIZE);
795                  lib_va_len = lib_va_end - lib_va_start;
796              }
797              lib_va_64_arena = vmem_create("lib_va_64",
798                  (void *)lib_va_start, lib_va_len, PAGE_SIZE,
799                  NULL, NULL, NULL, 0,
800                  VM_NOSLEEP | VMC_IDENTIFIER);
801              if (lib_va_64_arena == NULL) {
802                  mutex_exit(&lib_va_init_mutex);
803                  goto nolibva;
804              }
805          }
806          model_vmem = lib_va_64_arena;
807          mutex_exit(&lib_va_init_mutex);
808      } else if (lib_va_32_arena == NULL &&
809          model == DATAMODEL_ILP32) {
810          mutex_enter(&lib_va_init_mutex);
811          if (lib_va_32_arena == NULL) {
812              base = (caddr_t)align;
813              as_rangelock(as);
814              map_addr(&base, len, 0, 1, ma_flags);
815              as_rangeunlock(as);
816              if (base == NULL) {
817                  mutex_exit(&lib_va_init_mutex);
818                  MOBJ_STAT_ADD(lib_va_create_failure);
819                  goto nolibva;
820              }
821              lib_va_end = (size_t)base + len;
822              lib_va_len = lib_va_end >> 1;
823              lib_va_len = P2ROUNDUP(lib_va_len, PAGE_SIZE);
824              lib_va_start = lib_va_end - lib_va_len;
825              lib_va_32_arena = vmem_create("lib_va_32",

```

```

826          (void *)lib_va_start, lib_va_len, PAGE_SIZE,
827          NULL, NULL, NULL, 0,
828          VM_NOSLEEP | VMC_IDENTIFIER);
829          if (lib_va_32_arena == NULL) {
830              mutex_exit(&lib_va_init_mutex);
831              goto nolibva;
832          }
833      }
834      model_vmem = lib_va_32_arena;
835      mutex_exit(&lib_va_init_mutex);
836  }
837
838  if (model == DATAMODEL_LP64 || libs_mapped_32 < lib_threshold) {
839      base = vmem_xalloc(model_vmem, len, align, 0, 0, NULL,
840          NULL, VM_NOSLEEP | VM_ENDALLOC);
841      MOBJ_STAT_ADD(alloc_vmem);
842  }
843
844  /*
845  * Even if the address fails to fit in our address space,
846  * or we can't use a reserved address,
847  * we should still save it off in lib_va_hash.
848  */
849  *lvpp = lib_va_add_hash(base, len, align, vap);
850
851  /*
852  * Check for collision on insertion and free up our VA space.
853  * This is expected to be rare, so we'll just reset base to
854  * NULL instead of looking it up in the lib_va hash.
855  */
856  if (*lvpp == NULL) {
857      if (base != NULL) {
858          vmem_xfree(model_vmem, base, len);
859          base = NULL;
860          MOBJ_STAT_ADD(add_collision);
861      }
862  }
863  }
864
865  nolibva:
866      as_rangelock(as);
867
868      /*
869      * If we don't have an expected base address, or the one that we want
870      * to use is not available or acceptable, go get an acceptable
871      * address range.
872      *
873      * If ASLR is enabled, we should never have used the cache, and should
874      * also start our real work here, in the consequent of the next
875      * condition.
876      #endif /* ! codereview */
877      /*
878      if (randomize != 0)
879          ASSERT(base == NULL);
880
881      #endif /* ! codereview */
882      if (base == NULL || as_gap(as, len, &base, &len, 0, NULL) ||
883          valid_usr_range(base, len, PROT_ALL, as, as->a_userlimit) !=
884          RANGE_OKAY || OVERLAPS_STACK(base + len, p)) {
885          MOBJ_STAT_ADD(get_addr);
886          base = (caddr_t)align;
887          map_addr(&base, len, 0, 1, ma_flags);
888      }
889
890      /*
891      * Need to reserve the address space we're going to use.

```

```

892  * Don't reserve swap space since we'll be mapping over this.
893  */
894  if (base != NULL) {
895      /* Don't reserve swap space since we'll be mapping over this */
896      crargs.flags |= MAP_NORESERVE;
897      error = as_map(as, base, len, segvn_create, &crargs);
898      if (error) {
899          base = NULL;
900      }
901  }

903  as_rangeunlock(as);
904  return (base);
905  }

907 /*
908 * Map the file associated with vp into the address space as a single
909 * read only private mapping.
910 * Returns 0 for success, and non-zero for failure to map the file.
911 */
912 static int
913 mmabobj_map_flat(vnode_t *vp, mmabobj_result_t *mrp, size_t padding,
914                 cred_t *fcred)
915 {
916     int error = 0;
917     struct as *as = curproc->p_as;
918     caddr_t addr = NULL;
919     caddr_t start_addr;
920     size_t len;
921     size_t pad_len;
922     int prot = PROT_USER | PROT_READ;
923     uint_t ma_flags = _MAP_LOW32;
924     vattr_t vattr;
925     struct segvn_crargs crargs = SEGVN_ZFOD_ARGS(PROT_USER, PROT_ALL);

927     if (get_umatamodel() == DATAMODEL_LP64) {
928         ma_flags = 0;
929     }

931     vattr.va_mask = AT_SIZE;
932     error = VOP_GETATTR(vp, &vattr, 0, fcred, NULL);
933     if (error) {
934         return (error);
935     }

937     len = vattr.va_size;

939     ma_flags |= MAP_PRIVATE;
940     if (padding == 0) {
941         MOBJ_STAT_ADD(map_flat_no_padding);
942         error = VOP_MAP(vp, 0, as, &addr, len, prot, PROT_ALL,
943                        ma_flags, fcred, NULL);
944         if (error == 0) {
945             mrp[0].mr_addr = addr;
946             mrp[0].mr_msize = len;
947             mrp[0].mr_fsize = len;
948             mrp[0].mr_offset = 0;
949             mrp[0].mr_prot = prot;
950             mrp[0].mr_flags = 0;
951         }
952         return (error);
953     }

955     /* padding was requested so there's more work to be done */
956     MOBJ_STAT_ADD(map_flat_padding);

```

```

958     /* No need to reserve swap space now since it will be reserved later */
959     crargs.flags |= MAP_NORESERVE;

961     /* Need to setup padding which can only be in PAGESIZE increments. */
962     ASSERT((padding & PAGEOFFSET) == 0);
963     pad_len = len + (2 * padding);

965     as_rangelock(as);
966     map_addr(&addr, pad_len, 0, 1, ma_flags);
967     error = as_map(as, addr, pad_len, segvn_create, &crargs);
968     as_rangeunlock(as);
969     if (error) {
970         return (error);
971     }
972     start_addr = addr;
973     addr += padding;
974     ma_flags |= MAP_FIXED;
975     error = VOP_MAP(vp, 0, as, &addr, len, prot, PROT_ALL, ma_flags,
976                   fcred, NULL);
977     if (error == 0) {
978         mrp[0].mr_addr = start_addr;
979         mrp[0].mr_msize = padding;
980         mrp[0].mr_fsize = 0;
981         mrp[0].mr_offset = 0;
982         mrp[0].mr_prot = 0;
983         mrp[0].mr_flags = MR_PADDING;

985         mrp[1].mr_addr = addr;
986         mrp[1].mr_msize = len;
987         mrp[1].mr_fsize = len;
988         mrp[1].mr_offset = 0;
989         mrp[1].mr_prot = prot;
990         mrp[1].mr_flags = 0;

992         mrp[2].mr_addr = addr + P2ROUNDUP(len, PAGESIZE);
993         mrp[2].mr_msize = padding;
994         mrp[2].mr_fsize = 0;
995         mrp[2].mr_offset = 0;
996         mrp[2].mr_prot = 0;
997         mrp[2].mr_flags = MR_PADDING;
998     } else {
999         /* Need to cleanup the as_map from earlier */
1000         (void) as_unmap(as, start_addr, pad_len);
1001     }
1002     return (error);
1003 }

1005 /*
1006 * Map a PT_LOAD or PT_SUNWBSS section of an executable file into the user's
1007 * address space.
1008 * vp - vnode to be mapped in
1009 * addr - start address
1010 * len - length of vp to be mapped
1011 * zfodlen - length of zero filled memory after len above
1012 * offset - offset into file where mapping should start
1013 * prot - protections for this mapping
1014 * fcred - credentials for the file associated with vp at open time.
1015 */
1016 static int
1017 mmabobj_map_ptload(struct vnode *vp, caddr_t addr, size_t len, size_t zfodlen,
1018                  off_t offset, int prot, cred_t *fcred)
1019 {
1020     int error = 0;
1021     caddr_t zfodbase, oldaddr;
1022     size_t oldlen;
1023     size_t end;

```

```

1024     size_t zfoddiff;
1025     label_t ljb;
1026     struct as *as = curproc->p_as;
1027     model_t model;
1028     int full_page;

1030     /*
1031      * See if addr and offset are aligned such that we can map in
1032      * full pages instead of partial pages.
1033      */
1034     full_page = (((uintptr_t)addr & PAGEOFFSET) ==
1035                 ((uintptr_t)offset & PAGEOFFSET));

1037     model = get_udatamodel();

1039     oldaddr = addr;
1040     addr = (caddr_t)((uintptr_t)addr & (uintptr_t)PAGEMASK);
1041     if (len) {
1042         spgcnt_t availm, npages;
1043         int preread;
1044         uint_t mflag = MAP_PRIVATE | MAP_FIXED;

1046         if (model == DATAMODEL_ILP32) {
1047             mflag |= _MAP_LOW32;
1048         }
1049         /* We may need to map in extra bytes */
1050         oldlen = len;
1051         len += ((size_t)oldaddr & PAGEOFFSET);

1053         if (full_page) {
1054             offset = (off_t)((uintptr_t)offset & PAGEMASK);
1055             if ((prot & (PROT_WRITE | PROT_EXEC)) == PROT_EXEC) {
1056                 mflag |= MAP_TEXT;
1057                 MOBJ_STAT_ADD(map_ptload_text);
1058             } else {
1059                 mflag |= MAP_INITDATA;
1060                 MOBJ_STAT_ADD(map_ptload_initdata);
1061             }

1063             /*
1064              * maxprot is passed as PROT_ALL so that mdb can
1065              * write to this segment.
1066              */
1067             if (error = VOP_MAP(vp, (offset_t)offset, as, &addr,
1068                                len, prot, PROT_ALL, mflag, fcred, NULL)) {
1069                 return (error);
1070             }

1072             /*
1073              * If the segment can fit and is relatively small, then
1074              * we predefault the entire segment in. This is based
1075              * on the model that says the best working set of a
1076              * small program is all of its pages.
1077              * We only do this if freemem will not drop below
1078              * lotsfree since we don't want to induce paging.
1079              */
1080             npages = (spgcnt_t)btopr(len);
1081             availm = freemem - lotsfree;
1082             preread = (npages < availm && len < PGTHRESH) ? 1 : 0;

1084             /*
1085              * If we aren't predefaulting the segment,
1086              * increment "deficit", if necessary to ensure
1087              * that pages will become available when this
1088              * process starts executing.
1089              */

```

```

1090         if (preread == 0 && npages > availm &&
1091             deficit < lotsfree) {
1092             deficit += MIN((pgcnt_t)(npages - availm),
1093                           lotsfree - deficit);
1094         }

1096         if (preread) {
1097             (void) as_faulta(as, addr, len);
1098             MOBJ_STAT_ADD(map_ptload_preread);
1099         }
1100     } else {
1101         /*
1102          * addr and offset were not aligned such that we could
1103          * use VOP_MAP, thus we need to as_map the memory we
1104          * need and then read the data in from disk.
1105          * This code path is a corner case which should never
1106          * be taken, but hand crafted binaries could trigger
1107          * this logic and it needs to work correctly.
1108          */
1109         MOBJ_STAT_ADD(map_ptload_unaligned_text);
1110         as_rangelock(as);
1111         (void) as_unmap(as, addr, len);

1113         /*
1114          * We use zfod_argsp because we need to be able to
1115          * write to the mapping and then we'll change the
1116          * protections later if they are incorrect.
1117          */
1118         error = as_map(as, addr, len, segvn_create, zfod_argsp);
1119         as_rangeunlock(as);
1120         if (error) {
1121             MOBJ_STAT_ADD(map_ptload_unaligned_map_fail);
1122             return (error);
1123         }

1125         /* Now read in the data from disk */
1126         error = vn_rdwr(UIO_READ, vp, oldaddr, oldlen, offset,
1127                        UIO_USERSPACE, 0, (rlim64_t)0, fcred, NULL);
1128         if (error) {
1129             MOBJ_STAT_ADD(map_ptload_unaligned_read_fail);
1130             return (error);
1131         }

1133         /*
1134          * Now set protections.
1135          */
1136         if (prot != PROT_ZFOD) {
1137             (void) as_setprot(as, addr, len, prot);
1138         }
1139     }
1140 }

1142 if (zfodlen) {
1143     end = (size_t)addr + len;
1144     zfodbase = (caddr_t)P2ROUNDUP(end, PAGESIZE);
1145     zfoddiff = (uintptr_t)zfodbase - end;
1146     if (zfoddiff) {
1147         /*
1148          * Before we go to zero the remaining space on the last
1149          * page, make sure we have write permission.
1150          *
1151          * We need to be careful how we zero-fill the last page
1152          * if the protection does not include PROT_WRITE. Using
1153          * as_setprot() can cause the VM segment code to call
1154          * segvn_vpage(), which must allocate a page struct for
1155          * each page in the segment. If we have a very large

```

```

1156     * segment, this may fail, so we check for that, even
1157     * though we ignore other return values from as_setprot.
1158     */
1159     MOBJ_STAT_ADD(zfoddiff);
1160     if ((prot & PROT_WRITE) == 0) {
1161         if (as_setprot(as, (caddr_t)end, zfoddiff,
1162             prot | PROT_WRITE) == ENOMEM)
1163             return (ENOMEM);
1164         MOBJ_STAT_ADD(zfoddiff_nowrite);
1165     }
1166     if (on_fault(&ljb)) {
1167         no_fault();
1168         if ((prot & PROT_WRITE) == 0) {
1169             (void) as_setprot(as, (caddr_t)end,
1170                 zfoddiff, prot);
1171         }
1172         return (EFAULT);
1173     }
1174     uzero((void *)end, zfoddiff);
1175     no_fault();
1176
1177     /*
1178     * Remove write protection to return to original state
1179     */
1180     if ((prot & PROT_WRITE) == 0) {
1181         (void) as_setprot(as, (caddr_t)end,
1182             zfoddiff, prot);
1183     }
1184 }
1185 if (zfodlen > zfoddiff) {
1186     struct segvn_crargs =
1187         SEGVN_ZFOD_ARGS(prot, PROT_ALL);
1188
1189     MOBJ_STAT_ADD(zfodextra);
1190     zfodlen -= zfoddiff;
1191     crargs.szc = AS_MAP_NO_LPOOB;
1192
1193     as_rangelock(as);
1194     (void) as_unmap(as, (caddr_t)zfodbase, zfodlen);
1195     error = as_map(as, (caddr_t)zfodbase,
1196         zfodlen, segvn_create, &crargs);
1197     as_rangeunlock(as);
1198     if (error) {
1199         return (error);
1200     }
1201 }
1202 }
1203 }
1204 return (0);
1205 }
1206
1207 /*
1208 * Map the ELF file represented by vp into the users address space. The
1209 * first mapping will start at start_addr and there will be num_elements
1210 * mappings. The mappings are described by the data in mrp which may be
1211 * modified upon returning from this function.
1212 * Returns 0 for success or errno for failure.
1213 */
1214 static int
1215 mmapobj_map_elf(struct vnode *vp, caddr_t start_addr, mmapobj_result_t *mrp,
1216     int num_elements, cred_t *fcred, ushort_t e_type)
1217 {
1218     int i;
1219     int ret;
1220     caddr_t lo;
1221     caddr_t hi;

```

```

1222     struct as *as = curproc->p_as;
1223
1224     for (i = 0; i < num_elements; i++) {
1225         caddr_t addr;
1226         size_t p_memsz;
1227         size_t p_filesz;
1228         size_t zfodlen;
1229         offset_t p_offset;
1230         size_t dif;
1231         int prot;
1232
1233         /* Always need to adjust mr_addr */
1234         addr = start_addr + (size_t)(mrp[i].mr_addr);
1235         mrp[i].mr_addr =
1236             (caddr_t)((uintptr_t)addr & (uintptr_t)PAGEMASK);
1237
1238         /* Padding has already been mapped */
1239         if (MR_GET_TYPE(mrp[i].mr_flags) == MR_PADDING) {
1240             continue;
1241         }
1242         p_memsz = mrp[i].mr_msize;
1243         p_filesz = mrp[i].mr_fsize;
1244         zfodlen = p_memsz - p_filesz;
1245         p_offset = mrp[i].mr_offset;
1246         dif = (uintptr_t)(addr) & PAGEOFFSET;
1247         prot = mrp[i].mr_prot | PROT_USER;
1248         ret = mmapobj_map_ptload(vp, addr, p_filesz, zfodlen,
1249             p_offset, prot, fcred);
1250         if (ret != 0) {
1251             MOBJ_STAT_ADD(ptload_failed);
1252             mmapobj_unmap(mrp, i, num_elements, e_type);
1253             return (ret);
1254         }
1255
1256         /* Need to cleanup mrp to reflect the actual values used */
1257         mrp[i].mr_msize += dif;
1258         mrp[i].mr_offset = (size_t)addr & PAGEOFFSET;
1259     }
1260
1261     /* Also need to unmap any holes created above */
1262     if (num_elements == 1) {
1263         MOBJ_STAT_ADD(map_elf_no_holes);
1264         return (0);
1265     }
1266     if (e_type == ET_EXEC) {
1267         return (0);
1268     }
1269
1270     as_rangelock(as);
1271     lo = start_addr;
1272     hi = mrp[0].mr_addr;
1273
1274     /* Remove holes made by the rest of the segments */
1275     for (i = 0; i < num_elements - 1; i++) {
1276         lo = (caddr_t)P2ROUNDUP((size_t)(mrp[i].mr_addr) +
1277             mrp[i].mr_msize, PAGESIZE);
1278         hi = mrp[i + 1].mr_addr;
1279         if (lo < hi) {
1280             /*
1281              * If as_unmap fails we just use up a bit of extra
1282              * space
1283              */
1284             (void) as_unmap(as, (caddr_t)lo,
1285                 (size_t)hi - (size_t)lo);
1286             MOBJ_STAT_ADD(unmap_hole);
1287         }

```

```

1288     }
1289     as_rangeunlock(as);

1291     return (0);
1292 }

1294 /* Ugly hack to get STRUCT_* macros to work below */
1295 struct myphdr {
1296     Phdr      x;      /* native version */
1297 };

1299 struct myphdr32 {
1300     Elf32_Phdr x;
1301 };

1303 /*
1304 * Calculate and return the number of loadable segments in the ELF Phdr
1305 * represented by phdrbase as well as the len of the total mapping and
1306 * the max alignment that is needed for a given segment.  On success,
1307 * 0 is returned, and *len, *loadable and *align have been filled out.
1308 * On failure, errno will be returned, which in this case is ENOTSUP
1309 * if we were passed an ELF file with overlapping segments.
1310 */
1311 static int
1312 calc_loadable(Ehdr *ehdrp, caddr_t phdrbase, int nphdrs, size_t *len,
1313              int *loadable, size_t *align)
1314 {
1315     int i;
1316     int hsize;
1317     model_t model;
1318     ushort_t e_type = ehdrp->e_type;      /* same offset 32 and 64 bit */
1319     uint_t p_type;
1320     offset_t p_offset;
1321     size_t p_memsz;
1322     size_t p_align;
1323     caddr_t vaddr;
1324     int num_segs = 0;
1325     caddr_t start_addr = NULL;
1326     caddr_t p_end = NULL;
1327     size_t max_align = 0;
1328     size_t min_align = PAGE_SIZE; /* needed for vmem_xalloc */
1329     STRUCT_HANDLE(myphdr, mph);
1330 #if defined(__sparc)
1331     extern int vac_size;
1332
1333     /*
1334     * Want to prevent aliasing by making the start address at least be
1335     * aligned to vac_size.
1336     */
1337     min_align = MAX(PAGE_SIZE, vac_size);
1338 #endif

1340     model = get_udatamodel();
1341     STRUCT_SET_HANDLE(mph, model, (struct myphdr *)phdrbase);

1343     /* hsize alignment should have been checked before calling this func */
1344     if (model == DATAMODEL_LP64) {
1345         hsize = ehdrp->e_phentsize;
1346         if (hsize & 7) {
1347             return (ENOTSUP);
1348         }
1349     } else {
1350         ASSERT(model == DATAMODEL_ILP32);
1351         hsize = ((Elf32_Ehdr *)ehdrp)->e_phentsize;
1352         if (hsize & 3) {
1353             return (ENOTSUP);

```

```

1354     }
1355 }

1357 /*
1358 * Determine the span of all loadable segments and calculate the
1359 * number of loadable segments.
1360 */
1361 for (i = 0; i < nphdrs; i++) {
1362     p_type = STRUCT_FGET(mph, x.p_type);
1363     if (p_type == PT_LOAD || p_type == PT_SUNWBSS) {
1364         vaddr = (caddr_t)(uintptr_t)STRUCT_FGET(mph, x.p_vaddr);
1365         p_memsz = STRUCT_FGET(mph, x.p_memsz);

1367         /*
1368         * Skip this header if it requests no memory to be
1369         * mapped.
1370         */
1371         if (p_memsz == 0) {
1372             STRUCT_SET_HANDLE(mph, model,
1373                             (struct myphdr *)((size_t)STRUCT_BUF(mph) +
1374                             hsize));
1375             MOBJ_STAT_ADD(nomem_header);
1376             continue;
1377         }
1378         if (num_segs++ == 0) {
1379             /*
1380             * The p_vaddr of the first PT_LOAD segment
1381             * must either be NULL or within the first
1382             * page in order to be interpreted.
1383             * Otherwise, its an invalid file.
1384             */
1385             if (e_type == ET_DYN &&
1386                 ((caddr_t)((uintptr_t)vaddr &
1387                 (uintptr_t)PAGE_MASK) != NULL)) {
1388                 MOBJ_STAT_ADD(inval_header);
1389                 return (ENOTSUP);
1390             }
1391             start_addr = vaddr;
1392             /*
1393             * For the first segment, we need to map from
1394             * the beginning of the file, so we will
1395             * adjust the size of the mapping to include
1396             * this memory.
1397             */
1398             p_offset = STRUCT_FGET(mph, x.p_offset);
1399         } else {
1400             p_offset = 0;
1401         }
1402         /*
1403         * Check to make sure that this mapping wouldn't
1404         * overlap a previous mapping.
1405         */
1406         if (vaddr < p_end) {
1407             MOBJ_STAT_ADD(overlap_header);
1408             return (ENOTSUP);
1409         }

1411         p_end = vaddr + p_memsz + p_offset;
1412         p_end = (caddr_t)P2ROUNDUP((size_t)p_end, PAGE_SIZE);

1414         p_align = STRUCT_FGET(mph, x.p_align);
1415         if (p_align > 1 && p_align > max_align) {
1416             max_align = p_align;
1417             if (max_align < min_align) {
1418                 max_align = min_align;
1419                 MOBJ_STAT_ADD(min_align);

```

```

1420     }
1421     }
1422     }
1423     STRUCT_SET_HANDLE(mph, model,
1424         (struct myphdr *)((size_t)STRUCT_BUF(mph) + hsize));
1425 }
1427 /*
1428  * The alignment should be a power of 2, if it isn't we forgive it
1429  * and round up. On overflow, we'll set the alignment to max_align
1430  * rounded down to the nearest power of 2.
1431  */
1432 if (max_align > 0 && !ISP2(max_align)) {
1433     MOBJ_STAT_ADD(np2_align);
1434     *align = 2 * (1L << (highbit(max_align) - 1));
1435     if (*align < max_align ||
1436         (*align > UINT_MAX && model == DATAMODEL_ILP32)) {
1437         MOBJ_STAT_ADD(np2_align_overflow);
1438         *align = 1L << (highbit(max_align) - 1);
1439     }
1440 } else {
1441     *align = max_align;
1442 }
1444 ASSERT(*align >= PAGESIZE || *align == 0);
1446 *loadable = num_segs;
1447 *len = p_end - start_addr;
1448 return (0);
1449 }
1451 /*
1452  * Check the address space to see if the virtual addresses to be used are
1453  * available. If they are not, return errno for failure. On success, 0
1454  * will be returned, and the virtual addresses for each mmapobj_result_t
1455  * will be reserved. Note that a reservation could have earlier been made
1456  * for a given segment via a /dev/null mapping. If that is the case, then
1457  * we can use that VA space for our mappings.
1458  * Note: this function will only be used for ET_EXEC binaries.
1459  */
1460 int
1461 check_exec_addrs(int loadable, mmapobj_result_t *mrp, caddr_t start_addr)
1462 {
1463     int i;
1464     struct as *as = curproc->p_as;
1465     struct segvn_crargs crargs = SEGVN_ZFOD_ARGS(PROT_ZFOD, PROT_ALL);
1466     int ret;
1467     caddr_t myaddr;
1468     size_t mylen;
1469     struct seg *seg;
1471     /* No need to reserve swap space now since it will be reserved later */
1472     crargs.flags |= MAP_NORESERVE;
1473     as_rangelock(as);
1474     for (i = 0; i < loadable; i++) {
1476         myaddr = start_addr + (size_t)mrp[i].mr_addr;
1477         mylen = mrp[i].mr_msize;
1479         /* See if there is a hole in the as for this range */
1480         if (as_gap(as, mylen, &myaddr, &mylen, 0, NULL) == 0) {
1481             ASSERT(myaddr == start_addr + (size_t)mrp[i].mr_addr);
1482             ASSERT(mylen == mrp[i].mr_msize);
1484 #ifdef DEBUG
1485             if (MR_GET_TYPE(mrp[i].mr_flags) == MR_PADDING) {

```

```

1486         MOBJ_STAT_ADD(exec_padding);
1487     }
1488 #endif
1489     ret = as_map(as, myaddr, mylen, segvn_create, &crargs);
1490     if (ret) {
1491         as_rangeunlock(as);
1492         mmapobj_unmap_exec(mrp, i, start_addr);
1493         return (ret);
1494     }
1495 } else {
1496     /*
1497      * There is a mapping that exists in the range
1498      * so check to see if it was a "reservation"
1499      * from /dev/null. The mapping is from
1500      * /dev/null if the mapping comes from
1501      * segdev and the type is neither MAP_SHARED
1502      * nor MAP_PRIVATE.
1503      */
1504     AS_LOCK_ENTER(as, RW_READER);
1505     seg = as_findseg(as, myaddr, 0);
1506     MOBJ_STAT_ADD(exec_addr_mapped);
1507     if (seg && seg->s_ops == &segdev_ops &&
1508         ((SEGOP_GETTYPE(seg, myaddr) &
1509          (MAP_SHARED | MAP_PRIVATE)) == 0) &&
1510         myaddr >= seg->s_base &&
1511         myaddr + mylen <=
1512         seg->s_base + seg->s_size) {
1513         MOBJ_STAT_ADD(exec_addr_devnull);
1514         AS_LOCK_EXIT(as);
1515         (void) as_unmap(as, myaddr, mylen);
1516         ret = as_map(as, myaddr, mylen, segvn_create,
1517             &crargs);
1518         mrp[i].mr_flags |= MR_RESV;
1519         if (ret) {
1520             as_rangeunlock(as);
1521             /* Need to remap what we unmapped */
1522             mmapobj_unmap_exec(mrp, i + 1,
1523                 start_addr);
1524             return (ret);
1525         }
1526     } else {
1527         AS_LOCK_EXIT(as);
1528         as_rangeunlock(as);
1529         mmapobj_unmap_exec(mrp, i, start_addr);
1530         MOBJ_STAT_ADD(exec_addr_in_use);
1531         return (EADDRINUSE);
1532     }
1533 }
1534 }
1535 as_rangeunlock(as);
1536 return (0);
1537 }
1539 /*
1540  * Walk through the ELF program headers and extract all useful information
1541  * for PT_LOAD and PT_SUNWBSS segments into mrp.
1542  * Return 0 on success or error on failure.
1543  */
1544 static int
1545 process_phdrs(Ehdr *ehdrp, caddr_t phdrbase, int nphdrs, mmapobj_result_t *mrp,
1546     process_phdr(Ehdr *ehdrp, caddr_t phdrbase, int nphdrs, mmapobj_result_t *mrp,
1547     vnode_t *vp, uint_t *num_mapped, size_t padding, cred_t *fcred)
1548 {
1549     int i;
1550     caddr_t start_addr = NULL;
1551     caddr_t vaddr;

```



```

1551     size_t len = 0;
1552     size_t lib_len = 0;
1553     int ret;
1554     int prot;
1555     struct lib_va *lvp = NULL;
1556     vattr_t vattr;
1557     struct as *as = curproc->p_as;
1558     int error;
1559     int loadable = 0;
1560     int current = 0;
1561     int use_lib_va = 1;
1562     size_t align = 0;
1563     size_t add_pad = 0;
1564     int hdr_seen = 0;
1565     ushort_t e_type = ehdrp->e_type;      /* same offset 32 and 64 bit */
1566     uint_t p_type;
1567     offset_t p_offset;
1568     size_t p_memsz;
1569     size_t p_filesz;
1570     uint_t p_flags;
1571     int hsize;
1572     model_t model;
1573     STRUCT_HANDLE(myphdr, mph);

1574     model = get_udatamodel();
1575     STRUCT_SET_HANDLE(mph, model, (struct myphdr *)phdrbase);

1576
1577     /*
1578     * Need to make sure that hsize is aligned properly.
1579     * For 32bit processes, 4 byte alignment is required.
1580     * For 64bit processes, 8 byte alignment is required.
1581     * If the alignment isn't correct, we need to return failure
1582     * since it could cause an alignment error panic while walking
1583     * the phdr array.
1584     */
1585     if (model == DATAMODEL_LP64) {
1586         hsize = ehdrp->e_phentsize;
1587         if (hsize & 7) {
1588             MOBJ_STAT_ADD(phent_align64);
1589             return (ENOTSUP);
1590         }
1591     } else {
1592         ASSERT(model == DATAMODEL_ILP32);
1593         hsize = ((Elf32_Ehdr *)ehdrp)->e_phentsize;
1594         if (hsize & 3) {
1595             MOBJ_STAT_ADD(phent_align32);
1596             return (ENOTSUP);
1597         }
1598     }
1599
1600     if ((padding != 0) || secflag_enabled(curproc, PROC_SEC_ASLR)) {
1601         if (padding != 0) {
1602             use_lib_va = 0;
1603         }
1604         if (e_type == ET_DYN) {
1605             vattr.va_mask = AT_FSID | AT_NODEID | AT_CTIME | AT_MTIME;
1606             error = VOP_GETATTR(vp, &vattr, 0, fcred, NULL);
1607             if (error) {
1608                 return (error);
1609             }
1610             /* Check to see if we already have a description for this lib */
1611             if (!secflag_enabled(curproc, PROC_SEC_ASLR))
1612 #endif /* ! codereview */
1613                 lvp = lib_va_find(&vattr);
1614
1615         if (lvp != NULL) {

```

```

1616             MOBJ_STAT_ADD(lvp_found);
1617             if (use_lib_va) {
1618                 start_addr = mmabobj_lookup_start_addr(lvp);
1619                 if (start_addr == NULL) {
1620                     lib_va_release(lvp);
1621                     return (ENOMEM);
1622                 }
1623             }
1624
1625             /*
1626             * loadable may be zero if the original allocator
1627             * of lvp hasn't finished setting it up but the rest
1628             * of the fields will be accurate.
1629             */
1630             loadable = lvp->lv_num_segs;
1631             len = lvp->lv_len;
1632             align = lvp->lv_align;
1633         }
1634     }
1635
1636     /*
1637     * Determine the span of all loadable segments and calculate the
1638     * number of loadable segments, the total len spanned by the mappings
1639     * and the max alignment, if we didn't get them above.
1640     */
1641     if (loadable == 0) {
1642         MOBJ_STAT_ADD(no_loadable_yet);
1643         ret = calc_loadable(ehdrp, phdrbase, nphdrs, &len,
1644             &loadable, &align);
1645         if (ret != 0) {
1646             /*
1647             * Since it'd be an invalid file, we shouldn't have
1648             * cached it previously.
1649             */
1650             ASSERT(lvp == NULL);
1651             return (ret);
1652         }
1653 #ifdef DEBUG
1654         if (lvp) {
1655             ASSERT(len == lvp->lv_len);
1656             ASSERT(align == lvp->lv_align);
1657         }
1658 #endif
1659     }
1660
1661     /* Make sure there's something to map. */
1662     if (len == 0 || loadable == 0) {
1663         /*
1664         * Since it'd be an invalid file, we shouldn't have
1665         * cached it previously.
1666         */
1667         ASSERT(lvp == NULL);
1668         MOBJ_STAT_ADD(nothing_to_map);
1669         return (ENOTSUP);
1670     }
1671
1672     lib_len = len;
1673     if (padding != 0) {
1674         loadable += 2;
1675     }
1676     if (loadable > *num_mapped) {
1677         *num_mapped = loadable;
1678         /* cleanup previous reservation */
1679         if (start_addr) {
1680             (void) as_unmap(as, start_addr, lib_len);
1681         }

```

```

1682     MOBJ_STAT_ADD(e2big);
1683     if (lvp) {
1684         lib_va_release(lvp);
1685     }
1686     return (E2BIG);
1687 }

1689 /*
1690  * We now know the size of the object to map and now we need to
1691  * get the start address to map it at. It's possible we already
1692  * have it if we found all the info we need in the lib_va cache.
1693  */
1694 if (e_type == ET_DYN && start_addr == NULL) {
1695     /*
1696      * Need to make sure padding does not throw off
1697      * required alignment. We can only specify an
1698      * alignment for the starting address to be mapped,
1699      * so we round padding up to the alignment and map
1700      * from there and then throw out the extra later.
1701      */
1702     if (padding != 0) {
1703         if (align > 1) {
1704             add_pad = P2ROUNDUP(padding, align);
1705             len += add_pad;
1706             MOBJ_STAT_ADD(dyn_pad_align);
1707         } else {
1708             MOBJ_STAT_ADD(dyn_pad_noalign);
1709             len += padding; /* at beginning */
1710         }
1711         len += padding; /* at end of mapping */
1712     }
1713     /*
1714      * At this point, if lvp is non-NULL, then above we
1715      * already found it in the cache but did not get
1716      * the start address since we were not going to use lib_va.
1717      * Since we know that lib_va will not be used, it's safe
1718      * to call mmapobj_alloc_start_addr and know that lvp
1719      * will not be modified.
1720      */
1721     ASSERT(lvp ? use_lib_va == 0 : 1);
1722     start_addr = mmapobj_alloc_start_addr(&lvp, len,
1723         use_lib_va,
1724         secflag_enabled(curproc, PROC_SEC_ASLR),
1725         align, &vattr);
1726     use_lib_va, align, &vattr);
1727     if (start_addr == NULL) {
1728         if (lvp) {
1729             lib_va_release(lvp);
1730         }
1731         MOBJ_STAT_ADD(alloc_start_fail);
1732         return (ENOMEM);
1733     }
1734     /*
1735      * If we can't cache it, no need to hang on to it.
1736      * Setting lv_num_segs to non-zero will make that
1737      * field active and since there are too many segments
1738      * to cache, all future users will not try to use lv_mps.
1739      */
1740     if (lvp != NULL && loadable > LIBVA_CACHED_SEGS && use_lib_va) {
1741         lvp->lv_num_segs = loadable;
1742         lib_va_release(lvp);
1743         lvp = NULL;
1744         MOBJ_STAT_ADD(lvp_nocache);
1745     }
1746     /*
1747      * Free the beginning of the mapping if the padding

```

```

1747         * was not aligned correctly.
1748         */
1749         if (padding != 0 && add_pad != padding) {
1750             (void) as_unmap(as, start_addr,
1751                 add_pad - padding);
1752             start_addr += (add_pad - padding);
1753             MOBJ_STAT_ADD(extra_padding);
1754         }
1755     }

1757     /*
1758      * At this point, we have reserved the virtual address space
1759      * for our mappings. Now we need to start filling out the mrp
1760      * array to describe all of the individual mappings we are going
1761      * to return.
1762      * For ET_EXEC there has been no memory reservation since we are
1763      * using fixed addresses. While filling in the mrp array below,
1764      * we will have the first segment biased to start at addr 0
1765      * and the rest will be biased by this same amount. Thus if there
1766      * is padding, the first padding will start at addr 0, and the next
1767      * segment will start at the value of padding.
1768      */

1770     /* We'll fill out padding later, so start filling in mrp at index 1 */
1771     if (padding != 0) {
1772         current = 1;
1773     }

1775     /* If we have no more need for lvp let it go now */
1776     if (lvp != NULL && use_lib_va == 0) {
1777         lib_va_release(lvp);
1778         MOBJ_STAT_ADD(lvp_not_needed);
1779         lvp = NULL;
1780     }

1782     /* Now fill out the mrp structs from the program headers */
1783     STRUCT_SET_HANDLE(mph, model, (struct myphdr *)phdrbase);
1784     for (i = 0; i < nphdrs; i++) {
1785         p_type = STRUCT_FGET(mph, x.p_type);
1786         if (p_type == PT_LOAD || p_type == PT_SUNWBSS) {
1787             vaddr = (caddr_t)(uintptr_t)STRUCT_FGET(mph, x.p_vaddr);
1788             p_memsz = STRUCT_FGET(mph, x.p_memsz);
1789             p_filesz = STRUCT_FGET(mph, x.p_filesz);
1790             p_offset = STRUCT_FGET(mph, x.p_offset);
1791             p_flags = STRUCT_FGET(mph, x.p_flags);

1793             /*
1794              * Skip this header if it requests no memory to be
1795              * mapped.
1796              */
1797             if (p_memsz == 0) {
1798                 STRUCT_SET_HANDLE(mph, model,
1799                     (struct myphdr *)((size_t)STRUCT_BUF(mph) +
1800                         hsize));
1801                 MOBJ_STAT_ADD(no_mem_map_sz);
1802                 continue;
1803             }

1805             prot = 0;
1806             if (p_flags & PF_R)
1807                 prot |= PROT_READ;
1808             if (p_flags & PF_W)
1809                 prot |= PROT_WRITE;
1810             if (p_flags & PF_X)
1811                 prot |= PROT_EXEC;

```

```

1813     ASSERT(current < loadable);
1814     mrp[current].mr_msize = p_memsz;
1815     mrp[current].mr_fsize = p_filesz;
1816     mrp[current].mr_offset = p_offset;
1817     mrp[current].mr_prot = prot;

1819     if (hdr_seen == 0 && p_filesz != 0) {
1820         mrp[current].mr_flags = MR_HDR_ELF;
1821         /*
1822          * We modify mr_offset because we
1823          * need to map the ELF header as well, and if
1824          * we didn't then the header could be left out
1825          * of the mapping that we will create later.
1826          * Since we're removing the offset, we need to
1827          * account for that in the other fields as well
1828          * since we will be mapping the memory from 0
1829          * to p_offset.
1830          */
1831         if (e_type == ET_DYN) {
1832             mrp[current].mr_offset = 0;
1833             mrp[current].mr_msize += p_offset;
1834             mrp[current].mr_fsize += p_offset;
1835         } else {
1836             ASSERT(e_type == ET_EXEC);
1837             /*
1838              * Save off the start addr which will be
1839              * our bias for the rest of the
1840              * ET_EXEC mappings.
1841              */
1842             start_addr = vaddr - padding;
1843         }
1844         mrp[current].mr_addr = (caddr_t)padding;
1845         hdr_seen = 1;
1846     } else {
1847         if (e_type == ET_EXEC) {
1848             /* bias mr_addr */
1849             mrp[current].mr_addr =
1850                 vaddr - (size_t)start_addr;
1851         } else {
1852             mrp[current].mr_addr = vaddr + padding;
1853         }
1854         mrp[current].mr_flags = 0;
1855     }
1856     current++;
1857 }

1859     /* Move to next phdr */
1860     STRUCT_SET_HANDLE(mph, model,
1861         (struct myphdr *)((size_t)STRUCT_BUF(mph) +
1862             hsize));
1863 }

1865 /* Now fill out the padding segments */
1866 if (padding != 0) {
1867     mrp[0].mr_addr = NULL;
1868     mrp[0].mr_msize = padding;
1869     mrp[0].mr_fsize = 0;
1870     mrp[0].mr_offset = 0;
1871     mrp[0].mr_prot = 0;
1872     mrp[0].mr_flags = MR_PADDING;

1874     /* Setup padding for the last segment */
1875     ASSERT(current == loadable - 1);
1876     mrp[current].mr_addr = (caddr_t)lib_len + padding;
1877     mrp[current].mr_msize = padding;
1878     mrp[current].mr_fsize = 0;

```

```

1879         mrp[current].mr_offset = 0;
1880         mrp[current].mr_prot = 0;
1881         mrp[current].mr_flags = MR_PADDING;
1882     }

1884     /*
1885     * Need to make sure address ranges desired are not in use or
1886     * are previously allocated reservations from /dev/null. For
1887     * ET_DYN, we already made sure our address range was free.
1888     */
1889     if (e_type == ET_EXEC) {
1890         ret = check_exec_addrs(loadable, mrp, start_addr);
1891         if (ret != 0) {
1892             ASSERT(lvp == NULL);
1893             MOBJ_STAT_ADD(check_exec_failed);
1894             return (ret);
1895         }
1896     }

1898     /* Finish up our business with lvp. */
1899     if (lvp) {
1900         ASSERT(e_type == ET_DYN);
1901         if (lvp->lv_num_segs == 0 && loadable <= LIBVA_CACHED_SEGS) {
1902             bcopy(mrp, lvp->lv_mps,
1903                 loadable * sizeof (mmabobj_result_t));
1904             membar_producer();
1905         }
1906         /*
1907          * Setting lv_num_segs to a non-zero value indicates that
1908          * lv_mps is now valid and can be used by other threads.
1909          * So, the above stores need to finish before lv_num_segs
1910          * is updated. lv_mps is only valid if lv_num_segs is
1911          * greater than LIBVA_CACHED_SEGS.
1912          */
1913         lvp->lv_num_segs = loadable;
1914         lib_va_release(lvp);
1915         MOBJ_STAT_ADD(lvp_used);
1916     }

1918     /* Now that we have mrp completely filled out go map it */
1919     ret = mmabobj_map_elf(vp, start_addr, mrp, loadable, fcred, e_type);
1920     if (ret == 0) {
1921         *num_mapped = loadable;
1922     }

1924     return (ret);
1925 }

1927 /*
1928 * Take the ELF file passed in, and do the work of mapping it.
1929 * num_mapped in - # elements in user buffer
1930 * num_mapped out - # sections mapped and length of mrp array if
1931 * no errors.
1932 */
1933 static int
1934 doelfwork(Ehdr *ehdrp, vnode_t *vp, mmabobj_result_t *mrp,
1935     uint_t *num_mapped, size_t padding, cred_t *fcred)
1936 {
1937     int error;
1938     offset_t phoff;
1939     int nphdrs;
1940     unsigned char ei_class;
1941     unsigned short phentsize;
1942     ssize_t phsizep;
1943     caddr_t phbasep;
1944     int to_map;

```

```

1945     model_t model;

1947     ei_class = ehdrp->e_ident[EI_CLASS];
1948     model = get_udatamodel();
1949     if ((model == DATAMODEL_ILP32 && ei_class == ELFCLASS64) ||
1950         (model == DATAMODEL_LP64 && ei_class == ELFCLASS32)) {
1951         MOBJ_STAT_ADD(wrong_model);
1952         return (ENOTSUP);
1953     }

1955     /* Can't execute code from "noexec" mounted filesystem. */
1956     if (ehdrp->e_type == ET_EXEC &&
1957         (vp->v_vfsp->vfs_flag & VFS_NOEXEC) != 0) {
1958         MOBJ_STAT_ADD(noexec_fs);
1959         return (EACCES);
1960     }

1962     /*
1963     * Relocatable and core files are mapped as a single flat file
1964     * since no interpretation is done on them by mmapobj.
1965     */
1966     if (ehdrp->e_type == ET_REL || ehdrp->e_type == ET_CORE) {
1967         to_map = padding ? 3 : 1;
1968         if (*num_mapped < to_map) {
1969             *num_mapped = to_map;
1970             MOBJ_STAT_ADD(e2big_et_rel);
1971             return (E2BIG);
1972         }
1973         error = mmapobj_map_flat(vp, mrp, padding, fcred);
1974         if (error == 0) {
1975             *num_mapped = to_map;
1976             mrp[padding ? 1 : 0].mr_flags = MR_HDR_ELF;
1977             MOBJ_STAT_ADD(et_rel_mapped);
1978         }
1979         return (error);
1980     }

1982     /* Check for an unknown ELF type */
1983     if (ehdrp->e_type != ET_EXEC && ehdrp->e_type != ET_DYN) {
1984         MOBJ_STAT_ADD(unknown_elf_type);
1985         return (ENOTSUP);
1986     }

1988     if (ei_class == ELFCLASS32) {
1989         Elf32_Ehdr *e32hdr = (Elf32_Ehdr *)ehdrp;
1990         ASSERT(model == DATAMODEL_ILP32);
1991         nphdrs = e32hdr->e_phnum;
1992         phentsize = e32hdr->e_phentsize;
1993         if (phentsize < sizeof (Elf32_Phdr)) {
1994             MOBJ_STAT_ADD(phent32_too_small);
1995             return (ENOTSUP);
1996         }
1997         phoff = e32hdr->e_phoff;
1998     } else if (ei_class == ELFCLASS64) {
1999         Elf64_Ehdr *e64hdr = (Elf64_Ehdr *)ehdrp;
2000         ASSERT(model == DATAMODEL_LP64);
2001         nphdrs = e64hdr->e_phnum;
2002         phentsize = e64hdr->e_phentsize;
2003         if (phentsize < sizeof (Elf64_Phdr)) {
2004             MOBJ_STAT_ADD(phent64_too_small);
2005             return (ENOTSUP);
2006         }
2007         phoff = e64hdr->e_phoff;
2008     } else {
2009         /* fallthrough case for an invalid ELF class */
2010         MOBJ_STAT_ADD(inval_elf_class);

```

```

2011         return (ENOTSUP);
2012     }

2014     /*
2015     * nphdrs should only have this value for core files which are handled
2016     * above as a single mapping. If other file types ever use this
2017     * sentinel, then we'll add the support needed to handle this here.
2018     */
2019     if (nphdrs == PN_XNUM) {
2020         MOBJ_STAT_ADD(too_many_phdrs);
2021         return (ENOTSUP);
2022     }

2024     phsizep = nphdrs * phentsize;

2026     if (phsizep == 0) {
2027         MOBJ_STAT_ADD(no_phsize);
2028         return (ENOTSUP);
2029     }

2031     /* Make sure we only wait for memory if it's a reasonable request */
2032     if (phsizep > mmapobj_alloc_threshold) {
2033         MOBJ_STAT_ADD(phsize_large);
2034         if ((phbasep = kmem_alloc(phsizep, KM_NOSLEEP)) == NULL) {
2035             MOBJ_STAT_ADD(phsize_xtralarge);
2036             return (ENOMEM);
2037         }
2038     } else {
2039         phbasep = kmem_alloc(phsizep, KM_SLEEP);
2040     }

2042     if ((error = vn_rdwr(UIO_READ, vp, phbasep, phsizep,
2043         (offset_t)phoff, UIO_SYSSPACE, 0, (rlim64_t)0,
2044         fcred, NULL)) != 0) {
2045         kmem_free(phbasep, phsizep);
2046         return (error);
2047     }

2049     /* Now process the phdr's */
2050     error = process_phdrs(ehdrp, phbasep, nphdrs, mrp, vp, num_mapped,
1127     error = process_phdr(ehdrp, phbasep, nphdrs, mrp, vp, num_mapped,
2051         padding, fcred);
2052     kmem_free(phbasep, phsizep);
2053     return (error);
2054 }
_____unchanged_portion_omitted_____
2288 #endif

2290 /*
2291  * These are the two types of files that we can interpret and we want to read
2292  * in enough info to cover both types when looking at the initial header.
2293  */
2294 #define MAX_HEADER_SIZE (MAX(sizeof (Ehdr), sizeof (struct exec)))

2296 /*
2297  * Map vp passed in in an interpreted manner. ELF and AOUT files will be
2298  * interpreted and mapped appropriately for execution.
2299  * num_mapped in - # elements in mrp
2300  * num_mapped out - # sections mapped and length of mrp array if
2301  * no errors or E2BIG returned.
2302  *
2303  * Returns 0 on success, errno value on failure.
2304  */
2305 static int
2306 mmapobj_map_interpret(vnode_t *vp, mmapobj_result_t *mrp,
2307     uint_t *num_mapped, size_t padding, cred_t *fcred)

```

```

2308 {
2309     int error = 0;
2310     vattr_t vattr;
2311     struct lib_va *lvp;
2312     caddr_t start_addr;
2313     model_t model;

2315     /*
2316      * header has to be aligned to the native size of ulong_t in order
2317      * to avoid an unaligned access when dereferencing the header as
2318      * a ulong_t. Thus we allocate our array on the stack of type
2319      * ulong_t and then have header, which we dereference later as a char
2320      * array point at lheader.
2321      */
2322     ulong_t lheader[(MAX_HEADER_SIZE / (sizeof (ulong_t))) + 1];
2323     caddr_t header = (caddr_t)&lheader;

2325     vattr.va_mask = AT_FSID | AT_NODEID | AT_CTIME | AT_MTIME | AT_SIZE;
2326     error = VOP_GETATTR(vp, &vattr, 0, fcred, NULL);
2327     if (error) {
2328         return (error);
2329     }

2331     /*
2332      * Check lib_va to see if we already have a full description
2333      * for this library. This is the fast path and only used for
2334      * ET_DYN ELF files (dynamic libraries).
2335      */
2336     if (padding == 0 && !secflag_enabled(curproc, PROC_SEC_ASLR) &&
2337         ((lvp = lib_va_find(&vattr)) != NULL)) {
1413     if (padding == 0 && (lvp = lib_va_find(&vattr)) != NULL) {
2338         int num_segs;

2340         model = get_udatamodel();
2341         if ((model == DATAMODEL_ILP32 &&
2342             lvp->lv_flags & LV_ELF64) ||
2343             (model == DATAMODEL_LP64 &&
2344             lvp->lv_flags & LV_ELF32)) {
2345             lib_va_release(lvp);
2346             MOBJ_STAT_ADD(fast_wrong_model);
2347             return (ENOTSUP);
2348         }
2349         num_segs = lvp->lv_num_segs;
2350         if (*num_mapped < num_segs) {
2351             *num_mapped = num_segs;
2352             lib_va_release(lvp);
2353             MOBJ_STAT_ADD(fast_e2big);
2354             return (E2BIG);
2355         }

2357         /*
2358          * Check to see if we have all the mappable program headers
2359          * cached.
2360          */
2361         if (num_segs <= LIBVA_CACHED_SEGS && num_segs != 0) {
2362             MOBJ_STAT_ADD(fast);
2363             start_addr = mmapobj_lookup_start_addr(lvp);
2364             if (start_addr == NULL) {
2365                 lib_va_release(lvp);
2366                 return (ENOMEM);
2367             }

2369             bcopy(lvp->lv_mps, mrp,
2370                  num_segs * sizeof (mmapobj_result_t));

2372             error = mmapobj_map_elf(vp, start_addr, mrp,

```

```

2373         num_segs, fcred, ET_DYN);

2375         lib_va_release(lvp);
2376         if (error == 0) {
2377             *num_mapped = num_segs;
2378             MOBJ_STAT_ADD(fast_success);
2379         }
2380         return (error);
2381     }
2382     MOBJ_STAT_ADD(fast_not_now);

2384     /* Release it for now since we'll look it up below */
2385     lib_va_release(lvp);
2386 }

2388     /*
2389      * Time to see if this is a file we can interpret. If it's smaller
2390      * than this, then we can't interpret it.
2391      */
2392     if (vattr.va_size < MAX_HEADER_SIZE) {
2393         MOBJ_STAT_ADD(small_file);
2394         return (ENOTSUP);
2395     }

2397     if ((error = vn_rdwr(UIO_READ, vp, header, MAX_HEADER_SIZE, 0,
2398         UIO_SYSSPACE, 0, (rlim64_t)0, fcred, NULL)) != 0) {
2399         MOBJ_STAT_ADD(read_error);
2400         return (error);
2401     }

2403     /* Verify file type */
2404     if (header[EI_MAG0] == ELFMAG0 && header[EI_MAG1] == ELFMAG1 &&
2405         header[EI_MAG2] == ELFMAG2 && header[EI_MAG3] == ELFMAG3) {
2406         return (doelfwork((Ehdr *)lheader, vp, mrp, num_mapped,
2407             padding, fcred));
2408     }

2410 #if defined(__sparc)
2411     /* On sparc, check for 4.X AOUT format */
2412     switch (((struct exec *)header)->a_magic) {
2413     case OMAGIC:
2414     case ZMAGIC:
2415     case NMAGIC:
2416         return (doaoutwork(vp, mrp, num_mapped,
2417             (struct exec *)lheader, fcred));
2418     }
2419 #endif

2421     /* Unsupported type */
2422     MOBJ_STAT_ADD(unsupported);
2423     return (ENOTSUP);
2424 }
    unchanged_portion_omitted

```

new/usr/src/uts/common/os/policy.c

1

```
*****
63215 Thu Jun 30 22:03:30 2016
new/usr/src/uts/common/os/policy.c
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
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 (c) 2003, 2010, Oracle and/or its affiliates. All rights reserved.
23  * Copyright 2013, Joyent, Inc. All rights reserved.
24 */

26 #include <sys/types.h>
27 #include <sys/sysmacros.h>
28 #include <sys/param.h>
29 #include <sys/system.h>
30 #include <sys/cred_impl.h>
31 #include <sys/vnode.h>
32 #include <sys/vfs.h>
33 #include <sys/stat.h>
34 #include <sys/errno.h>
35 #include <sys/kmem.h>
36 #include <sys/user.h>
37 #include <sys/proc.h>
38 #include <sys/acct.h>
39 #include <sys/ipc_impl.h>
40 #include <sys/cmn_err.h>
41 #include <sys/debug.h>
42 #include <sys/policy.h>
43 #include <sys/kobj.h>
44 #include <sys/msg.h>
45 #include <sys/devpolicy.h>
46 #include <c2/audit.h>
47 #include <sys/varargs.h>
48 #include <sys/klpd.h>
49 #include <sys/modctl.h>
50 #include <sys/disp.h>
51 #include <sys/zone.h>
52 #include <inet/optcom.h>
53 #include <sys/sdt.h>
54 #include <sys/vfs.h>
55 #include <sys/mntent.h>
56 #include <sys/contract_impl.h>
57 #include <sys/dld_ioc.h>
```

new/usr/src/uts/common/os/policy.c

2

```
59 /*
60  * There are two possible layers of privilege routines and two possible
61  * levels of secpolicy. Plus one other we may not be interested in, so
62  * we may need as many as 6 but no more.
63 */
64 #define MAXPRIVSTACK        6

66 int priv_debug = 0;
67 int priv_basic_test = -1;

69 /*
70  * This file contains the majority of the policy routines.
71  * Since the policy routines are defined by function and not
72  * by privilege, there is quite a bit of duplication of
73  * functions.
74  *
75  * The secpolicy functions must not make assumptions about
76  * locks held or not held as any lock can be held while they're
77  * being called.
78  *
79  * Credentials are read-only so no special precautions need to
80  * be taken while locking them.
81  *
82  * When a new policy check needs to be added to the system the
83  * following procedure should be followed:
84  *
85  *     Pick an appropriate secpolicy_*() function
86  *     -> done if one exists.
87  *     Create a new secpolicy function, preferably with
88  *     a descriptive name using the standard template.
89  *     Pick an appropriate privilege for the policy.
90  *     If no appropriate privilege exists, define new one
91  *     (this should be done with extreme care; in most cases
92  *     little is gained by adding another privilege)
93  *
94  * WHY ROOT IS STILL SPECIAL.
95  *
96  * In a number of the policy functions, there are still explicit
97  * checks for uid 0. The rationale behind this is that many root
98  * owned files/objects hold configuration information which can give full
99  * privileges to the user once written to. To prevent escalation
100 * of privilege by allowing just a single privilege to modify root owned
101 * objects, we've added these root specific checks where we considered
102 * them necessary: modifying root owned files, changing uids to 0, etc.
103 *
104 * PRIVILEGE ESCALATION AND ZONES.
105 *
106 * A number of operations potentially allow the caller to achieve
107 * privileges beyond the ones normally required to perform the operation.
108 * For example, if allowed to create a setuid 0 executable, a process can
109 * gain privileges beyond PRIV_FILE_SETUID. Zones, however, place
110 * restrictions on the ability to gain privileges beyond those available
111 * within the zone through file and process manipulation. Hence, such
112 * operations require that the caller have an effective set that includes
113 * all privileges available within the current zone, or all privileges
114 * if executing in the global zone.
115 *
116 * This is indicated in the priv_policy* policy checking functions
117 * through a combination of parameters. The "priv" parameter indicates
118 * the privilege that is required, and the "allzone" parameter indicates
119 * whether or not all privileges in the zone are required. In addition,
120 * priv can be set to PRIV_ALL to indicate that all privileges are
121 * required (regardless of zone). There are three scenarios of interest:
122 * (1) operation requires a specific privilege
123 * (2) operation requires a specific privilege, and requires all
124 *     privileges available within the zone (or all privileges if in
```

```

125 * the global zone)
126 * (3) operation requires all privileges, regardless of zone
127 *
128 * For (1), priv should be set to the specific privilege, and allzone
129 * should be set to B_FALSE.
130 * For (2), priv should be set to the specific privilege, and allzone
131 * should be set to B_TRUE.
132 * For (3), priv should be set to PRIV_ALL, and allzone should be set
133 * to B_FALSE.
134 *
135 */

137 /*
138 * The privileges are checked against the Effective set for
139 * ordinary processes and checked against the Limit set
140 * for euid 0 processes that haven't manipulated their privilege
141 * sets.
142 */
143 #define HAS_ALLPRIVS(cr) priv_isfullset(&CR_OEPRIV(cr))
144 #define ZONEPRIVS(cr) ((cr)->cr_zone->zone_privset)
145 #define HAS_ALLZONEPRIVS(cr) priv_issubset(ZONEPRIVS(cr), &CR_OEPRIV(cr))
146 #define HAS_PRIVILEGE(cr, pr) ((pr) == PRIV_ALL ? \
147 HAS_ALLPRIVS(cr) : \
148 PRIV_ISMEMBER(&CR_OEPRIV(cr), pr))
149 #define PRIV_ISASSERT(&CR_OEPRIV(cr), pr)
150 #define FAST_BASIC_CHECK(cr, priv) \
151 if (PRIV_ISMEMBER(&CR_OEPRIV(cr), priv)) { \
152 if (PRIV_ISASSERT(&CR_OEPRIV(cr), priv)) { \
153 DTRACE_PROBE2(priv_ok, int, priv, boolean_t, B_FALSE); \
154 return (0); \
155 }
156 */
157 * Policy checking functions.
158 *
159 * All of the system's policy should be implemented here.
160 */

162 /*
163 * Private functions which take an additional va_list argument to
164 * implement an object specific policy override.
165 */
166 static int priv_policy_ap(const cred_t *, int, boolean_t, int,
167 const char *, va_list);
168 static int priv_policy_va(const cred_t *, int, boolean_t, int,
169 const char *, ...);

171 /*
172 * Generic policy calls
173 *
174 * The "bottom" functions of policy control
175 */
176 static char *
177 mprintf(const char *fmt, ...)
178 {
179 va_list args;
180 char *buf;
181 size_t len;

183 va_start(args, fmt);
184 len = vsnprintf(NULL, 0, fmt, args) + 1;
185 va_end(args);

187 buf = kmem_alloc(len, KM_NOSLEEP);

```

```

189 if (buf == NULL)
190 return (NULL);

192 va_start(args, fmt);
193 (void) vsnprintf(buf, len, fmt, args);
194 va_end(args);

196 return (buf);
197 }
198
199 unchanged portion omitted

391 /*
392 * priv_policy_ap()
393 * return 0 or error.
394 * See block comment above for a description of "priv" and "allzone" usage.
395 */
396 static int
397 priv_policy_ap(const cred_t *cr, int priv, boolean_t allzone, int err,
398 const char *msg, va_list ap)
399 {
400 if ((HAS_PRIVILEGE(cr, priv) && (!allzone || HAS_ALLZONEPRIVS(cr))) ||
401 (!servicing_interrupt() &&
402 priv_policy_override(cr, priv, allzone, ap) == 0)) {
403 if ((allzone || priv == PRIV_ALL ||
404 !PRIV_ISMEMBER(priv_basic, priv)) &&
405 !PRIV_ISASSERT(priv_basic, priv)) &&
406 !servicing_interrupt()) {
407 PTOU(curproc)->u_aclflag |= ASU; /* Needed for SVVS */
408 if (AU_AUDITING())
409 audit_priv(priv,
410 allzone ? ZONEPRIVS(cr) : NULL, 1);
411 }
412 err = 0;
413 DTRACE_PROBE2(priv_ok, int, priv, boolean_t, allzone);
414 } else if (!servicing_interrupt()) {
415 /* Failure audited in this procedure */
416 priv_policy_err(cr, priv, allzone, msg);
417 }
418 return (err);
419 }
420
421 unchanged portion omitted

441 /*
442 * Return B_TRUE for sufficient privileges, B_FALSE for insufficient privileges.
443 */
444 boolean_t
445 priv_policy_choice(const cred_t *cr, int priv, boolean_t allzone)
446 {
447 boolean_t res = HAS_PRIVILEGE(cr, priv) &&
448 (!allzone || HAS_ALLZONEPRIVS(cr));

450 /* Audit success only */
451 if (res && AU_AUDITING() &&
452 (allzone || priv == PRIV_ALL || !PRIV_ISMEMBER(priv_basic, priv)) &&
453 !PRIV_ISASSERT(priv_basic, priv)) &&
454 !servicing_interrupt()) {
455 audit_priv(priv, allzone ? ZONEPRIVS(cr) : NULL, 1);
456 }
457 if (res) {
458 DTRACE_PROBE2(priv_ok, int, priv, boolean_t, allzone);
459 } else {
460 DTRACE_PROBE2(priv_err, int, priv, boolean_t, allzone);
461 }
462 return (res);
463 }
464
465 unchanged portion omitted

```

```

963 /*
964  * Like secpolicy_vnode_access() but we get the actual wanted mode and the
965  * current mode of the file, not the missing bits.
966  */
967 int
968 secpolicy_vnode_access2(const cred_t *cr, vnode_t *vp, uid_t owner,
969     mode_t curmode, mode_t wantmode)
970 {
971     mode_t mode;

973     /* Inline the basic privileges tests. */
974     if ((wantmode & VREAD) &&
975         !PRIV_ISMEMBER(&CR_OEPRIV(cr), PRIV_FILE_READ) &&
976         !PRIV_ISASSET(&CR_OEPRIV(cr), PRIV_FILE_READ) &&
977         priv_policy_va(cr, PRIV_FILE_READ, B_FALSE, EACCES, NULL,
978             KLPDARG_VNODE, vp, (char *)NULL, KLPDARG_NOMORE) != 0) {
979         return (EACCES);
980     }

981     if ((wantmode & VWRITE) &&
982         !PRIV_ISMEMBER(&CR_OEPRIV(cr), PRIV_FILE_WRITE) &&
983         !PRIV_ISASSET(&CR_OEPRIV(cr), PRIV_FILE_WRITE) &&
984         priv_policy_va(cr, PRIV_FILE_WRITE, B_FALSE, EACCES, NULL,
985             KLPDARG_VNODE, vp, (char *)NULL, KLPDARG_NOMORE) != 0) {
986         return (EACCES);
987     }

988     mode = ~curmode & wantmode;

990     if (mode == 0)
991         return (0);

993     if ((mode & VREAD) && priv_policy_va(cr, PRIV_FILE_DAC_READ, B_FALSE,
994         EACCES, NULL, KLPDARG_VNODE, vp, (char *)NULL,
995         KLPDARG_NOMORE) != 0) {
996         return (EACCES);
997     }

999     if (mode & VWRITE) {
1000         boolean_t allzone;

1002         if (owner == 0 && cr->cr_uid != 0)
1003             allzone = B_TRUE;
1004         else
1005             allzone = B_FALSE;
1006         if (priv_policy_va(cr, PRIV_FILE_DAC_WRITE, allzone, EACCES,
1007             NULL, KLPDARG_VNODE, vp, (char *)NULL,
1008             KLPDARG_NOMORE) != 0) {
1009             return (EACCES);
1010         }
1011     }

1013     if (mode & VEEXEC) {
1014         /*
1015          * Directories use file_dac_search to override the execute bit.
1016          */
1017         int p = vp->v_type == VDIR ? PRIV_FILE_DAC_SEARCH :
1018             PRIV_FILE_DAC_EXECUTE;

1020         return (priv_policy_va(cr, p, B_FALSE, EACCES, NULL,
1021             KLPDARG_VNODE, vp, (char *)NULL, KLPDARG_NOMORE));
1022     }
1023     return (0);
1024 }

```

unchanged portion omitted

```

1731 /* Process security flags */
1732 int
1733 secpolicy_psecflags(const cred_t *cr, proc_t *tp, proc_t *sp)
1734 {
1735     if (PRIV_POLICY(cr, PRIV_PROC_SECFLAGS, B_FALSE, EPERM, NULL) != 0)
1736         return (EPERM);

1738     if (!prochasprocperm(tp, sp, cr))
1739         return (EPERM);

1741     return (0);
1742 }

1744 #endif /* !codereview */
1745 /*
1746  * Processor set binding.
1747  */
1748 int
1749 secpolicy_pbind(const cred_t *cr)
1750 {
1751     if (PRIV_POLICY_ONLY(cr, PRIV_SYS_RES_CONFIG, B_FALSE))
1752         return (secpolicy_pset(cr));
1753     return (PRIV_POLICY(cr, PRIV_SYS_RES_BIND, B_FALSE, EPERM, NULL));
1754 }

1756 int
1757 secpolicy_ponline(const cred_t *cr)
1758 {
1759     return (PRIV_POLICY(cr, PRIV_SYS_RES_CONFIG, B_FALSE, EPERM, NULL));
1760 }

1762 int
1763 secpolicy_pool(const cred_t *cr)
1764 {
1765     return (PRIV_POLICY(cr, PRIV_SYS_RES_CONFIG, B_FALSE, EPERM, NULL));
1766 }

1768 int
1769 secpolicy_blacklist(const cred_t *cr)
1770 {
1771     return (PRIV_POLICY(cr, PRIV_SYS_RES_CONFIG, B_FALSE, EPERM, NULL));
1772 }

1774 /*
1775  * Catch all system configuration.
1776  */
1777 int
1778 secpolicy_sys_config(const cred_t *cr, boolean_t checkonly)
1779 {
1780     if (checkonly) {
1781         return (PRIV_POLICY_ONLY(cr, PRIV_SYS_CONFIG, B_FALSE) ? 0 :
1782             EPERM);
1783     } else {
1784         return (PRIV_POLICY(cr, PRIV_SYS_CONFIG, B_FALSE, EPERM, NULL));
1785     }
1786 }

1788 /*
1789  * Zone administration (halt, reboot, etc.) from within zone.
1790  */
1791 int
1792 secpolicy_zone_admin(const cred_t *cr, boolean_t checkonly)
1793 {
1794     if (checkonly) {
1795         return (PRIV_POLICY_ONLY(cr, PRIV_SYS_ADMIN, B_FALSE) ? 0 :

```



```

1796         EPERM);
1797     } else {
1798         return (PRIV_POLICY(cr, PRIV_SYS_ADMIN, B_FALSE, EPERM,
1799             NULL));
1800     }
1801 }

1803 /*
1804  * Zone configuration (create, halt, enter).
1805  */
1806 int
1807 secpolicy_zone_config(const cred_t *cr)
1808 {
1809     /*
1810      * Require all privileges to avoid possibility of privilege
1811      * escalation.
1812      */
1813     return (secpolicy_require_set(cr, PRIV_FULLSET, NULL, KLPDARG_NONE));
1814 }

1816 /*
1817  * Various other system configuration calls
1818  */
1819 int
1820 secpolicy_coreadm(const cred_t *cr)
1821 {
1822     return (PRIV_POLICY(cr, PRIV_SYS_ADMIN, B_FALSE, EPERM, NULL));
1823 }

1825 int
1826 secpolicy_systeminfo(const cred_t *cr)
1827 {
1828     return (PRIV_POLICY(cr, PRIV_SYS_ADMIN, B_FALSE, EPERM, NULL));
1829 }

1831 int
1832 secpolicy_dispadm(const cred_t *cr)
1833 {
1834     return (PRIV_POLICY(cr, PRIV_SYS_CONFIG, B_FALSE, EPERM, NULL));
1835 }

1837 int
1838 secpolicy_settime(const cred_t *cr)
1839 {
1840     return (PRIV_POLICY(cr, PRIV_SYS_TIME, B_FALSE, EPERM, NULL));
1841 }

1843 /*
1844  * For realtime users: high resolution clock.
1845  */
1846 int
1847 secpolicy_clock_highres(const cred_t *cr)
1848 {
1849     return (PRIV_POLICY(cr, PRIV_PROC_CLOCK_HIGHRES, B_FALSE, EPERM,
1850         NULL));
1851 }

1853 /*
1854  * drv_priv() is documented as callable from interrupt context, not that
1855  * anyone ever does, but still.  No debugging or auditing can be done when
1856  * it is called from interrupt context.
1857  * returns 0 on succes, EPERM on failure.
1858  */
1859 int
1860 drv_priv(cred_t *cr)
1861 {

```

```

1862     return (PRIV_POLICY(cr, PRIV_SYS_DEVICES, B_FALSE, EPERM, NULL));
1863 }

1865 int
1866 secpolicy_sys_devices(const cred_t *cr)
1867 {
1868     return (PRIV_POLICY(cr, PRIV_SYS_DEVICES, B_FALSE, EPERM, NULL));
1869 }

1871 int
1872 secpolicy_excl_open(const cred_t *cr)
1873 {
1874     return (PRIV_POLICY(cr, PRIV_SYS_DEVICES, B_FALSE, EBUSY, NULL));
1875 }

1877 int
1878 secpolicy_rctlsys(const cred_t *cr, boolean_t is_zone_rctl)
1879 {
1880     /* zone.* rctls can only be set from the global zone */
1881     if (is_zone_rctl && priv_policy_global(cr) != 0)
1882         return (EPERM);
1883     return (PRIV_POLICY(cr, PRIV_SYS_RESOURCE, B_FALSE, EPERM, NULL));
1884 }

1886 int
1887 secpolicy_resource(const cred_t *cr)
1888 {
1889     return (PRIV_POLICY(cr, PRIV_SYS_RESOURCE, B_FALSE, EPERM, NULL));
1890 }

1892 int
1893 secpolicy_resource_anon_mem(const cred_t *cr)
1894 {
1895     return (PRIV_POLICY_ONLY(cr, PRIV_SYS_RESOURCE, B_FALSE));
1896 }

1898 /*
1899  * Processes with a real uid of 0 escape any form of accounting, much
1900  * like before.
1901  */
1902 int
1903 secpolicy_newproc(const cred_t *cr)
1904 {
1905     if (cr->cr_ruid == 0)
1906         return (0);

1908     return (PRIV_POLICY(cr, PRIV_SYS_RESOURCE, B_FALSE, EPERM, NULL));
1909 }

1911 /*
1912  * Networking
1913  */
1914 int
1915 secpolicy_net_rawaccess(const cred_t *cr)
1916 {
1917     return (PRIV_POLICY(cr, PRIV_NET_RAWACCESS, B_FALSE, EACCES, NULL));
1918 }

1920 int
1921 secpolicy_net_observability(const cred_t *cr)
1922 {
1923     return (PRIV_POLICY(cr, PRIV_NET_OBSERVABILITY, B_FALSE, EACCES, NULL));
1924 }

1926 /*
1927  * Need this privilege for accessing the ICMP device

```

```

1928 */
1929 int
1930 secpolicy_net_icmpaccess(const cred_t *cr)
1931 {
1932     return (PRIV_POLICY(cr, PRIV_NET_ICMPACCESS, B_FALSE, EACCES, NULL));
1933 }

1935 /*
1936  * There are a few rare cases where the kernel generates ioctl() from
1937  * interrupt context with a credential of kcred rather than NULL.
1938  * In those cases, we take the safe and cheap test.
1939  */
1940 int
1941 secpolicy_net_config(const cred_t *cr, boolean_t checkonly)
1942 {
1943     if (checkonly) {
1944         return (PRIV_POLICY_ONLY(cr, PRIV_SYS_NET_CONFIG, B_FALSE) ?
1945             0 : EPERM);
1946     } else {
1947         return (PRIV_POLICY(cr, PRIV_SYS_NET_CONFIG, B_FALSE, EPERM,
1948             NULL));
1949     }
1950 }

1953 /*
1954  * PRIV_SYS_NET_CONFIG is a superset of PRIV_SYS_IP_CONFIG.
1955  */
1956  * There are a few rare cases where the kernel generates ioctl() from
1957  * interrupt context with a credential of kcred rather than NULL.
1958  * In those cases, we take the safe and cheap test.
1959  */
1960 int
1961 secpolicy_ip_config(const cred_t *cr, boolean_t checkonly)
1962 {
1963     if (PRIV_POLICY_ONLY(cr, PRIV_SYS_NET_CONFIG, B_FALSE))
1964         return (secpolicy_net_config(cr, checkonly));

1966     if (checkonly) {
1967         return (PRIV_POLICY_ONLY(cr, PRIV_SYS_IP_CONFIG, B_FALSE) ?
1968             0 : EPERM);
1969     } else {
1970         return (PRIV_POLICY(cr, PRIV_SYS_IP_CONFIG, B_FALSE, EPERM,
1971             NULL));
1972     }
1973 }

1975 /*
1976  * PRIV_SYS_NET_CONFIG is a superset of PRIV_SYS_DL_CONFIG.
1977  */
1978 int
1979 secpolicy_dl_config(const cred_t *cr)
1980 {
1981     if (PRIV_POLICY_ONLY(cr, PRIV_SYS_NET_CONFIG, B_FALSE))
1982         return (secpolicy_net_config(cr, B_FALSE));
1983     return (PRIV_POLICY(cr, PRIV_SYS_DL_CONFIG, B_FALSE, EPERM, NULL));
1984 }

1986 /*
1987  * PRIV_SYS_DL_CONFIG is a superset of PRIV_SYS_IPTUN_CONFIG.
1988  */
1989 int
1990 secpolicy iptun_config(const cred_t *cr)
1991 {
1992     if (PRIV_POLICY_ONLY(cr, PRIV_SYS_NET_CONFIG, B_FALSE))
1993         return (secpolicy_net_config(cr, B_FALSE));

```

```

1994     if (PRIV_POLICY_ONLY(cr, PRIV_SYS_DL_CONFIG, B_FALSE))
1995         return (secpolicy_dl_config(cr));
1996     return (PRIV_POLICY(cr, PRIV_SYS_IPTUN_CONFIG, B_FALSE, EPERM, NULL));
1997 }

1999 /*
2000  * Map IP pseudo privileges to actual privileges.
2001  * So we don't need to recompile IP when we change the privileges.
2002  */
2003 int
2004 secpolicy_ip(const cred_t *cr, int netpriv, boolean_t checkonly)
2005 {
2006     int priv = PRIV_ALL;

2008     switch (netpriv) {
2009     case OP_CONFIG:
2010         priv = PRIV_SYS_IP_CONFIG;
2011         break;
2012     case OP_RAW:
2013         priv = PRIV_NET_RAWACCESS;
2014         break;
2015     case OP_PRIVPORT:
2016         priv = PRIV_NET_PRIVADDR;
2017         break;
2018     }
2019     ASSERT(priv != PRIV_ALL);
2020     if (checkonly)
2021         return (PRIV_POLICY_ONLY(cr, priv, B_FALSE) ? 0 : EPERM);
2022     else
2023         return (PRIV_POLICY(cr, priv, B_FALSE, EPERM, NULL));
2024 }

2026 /*
2027  * Map network pseudo privileges to actual privileges.
2028  * So we don't need to recompile IP when we change the privileges.
2029  */
2030 int
2031 secpolicy_net(const cred_t *cr, int netpriv, boolean_t checkonly)
2032 {
2033     int priv = PRIV_ALL;

2035     switch (netpriv) {
2036     case OP_CONFIG:
2037         priv = PRIV_SYS_NET_CONFIG;
2038         break;
2039     case OP_RAW:
2040         priv = PRIV_NET_RAWACCESS;
2041         break;
2042     case OP_PRIVPORT:
2043         priv = PRIV_NET_PRIVADDR;
2044         break;
2045     }
2046     ASSERT(priv != PRIV_ALL);
2047     if (checkonly)
2048         return (PRIV_POLICY_ONLY(cr, priv, B_FALSE) ? 0 : EPERM);
2049     else
2050         return (PRIV_POLICY(cr, priv, B_FALSE, EPERM, NULL));
2051 }

2053 /*
2054  * Checks for operations that are either client-only or are used by
2055  * both clients and servers.
2056  */
2057 int
2058 secpolicy_nfs(const cred_t *cr)
2059 {

```

```

2060     return (PRIV_POLICY(cr, PRIV_SYS_NFS, B_FALSE, EPERM, NULL));
2061 }

2063 /*
2064 * Special case for opening rpcmod: have NFS privileges or network
2065 * config privileges.
2066 */
2067 int
2068 secpolicy_rpcmod_open(const cred_t *cr)
2069 {
2070     if (PRIV_POLICY_ONLY(cr, PRIV_SYS_NFS, B_FALSE))
2071         return (secpolicy_nfs(cr));
2072     else
2073         return (secpolicy_net_config(cr, NULL));
2074 }

2076 int
2077 secpolicy_chroot(const cred_t *cr)
2078 {
2079     return (PRIV_POLICY(cr, PRIV_PROC_CHROOT, B_FALSE, EPERM, NULL));
2080 }

2082 int
2083 secpolicy_tasksys(const cred_t *cr)
2084 {
2085     return (PRIV_POLICY(cr, PRIV_PROC_TASKID, B_FALSE, EPERM, NULL));
2086 }

2088 int
2089 secpolicy_meminfo(const cred_t *cr)
2090 {
2091     return (PRIV_POLICY(cr, PRIV_PROC_MEMINFO, B_FALSE, EPERM, NULL));
2092 }

2094 int
2095 secpolicy_pfexec_register(const cred_t *cr)
2096 {
2097     return (PRIV_POLICY(cr, PRIV_SYS_ADMIN, B_TRUE, EPERM, NULL));
2098 }

2100 /*
2101 * Basic privilege checks.
2102 */
2103 int
2104 secpolicy_basic_exec(const cred_t *cr, vnode_t *vp)
2105 {
2106     FAST_BASIC_CHECK(cr, PRIV_PROC_EXEC);

2108     return (priv_policy_va(cr, PRIV_PROC_EXEC, B_FALSE, EPERM, NULL,
2109         KLPDARG_VNODE, vp, (char *)NULL, KLPDARG_NOMORE));
2110 }

2112 int
2113 secpolicy_basic_fork(const cred_t *cr)
2114 {
2115     FAST_BASIC_CHECK(cr, PRIV_PROC_FORK);

2117     return (PRIV_POLICY(cr, PRIV_PROC_FORK, B_FALSE, EPERM, NULL));
2118 }

2120 int
2121 secpolicy_basic_proc(const cred_t *cr)
2122 {
2123     FAST_BASIC_CHECK(cr, PRIV_PROC_SESSION);

2125     return (PRIV_POLICY(cr, PRIV_PROC_SESSION, B_FALSE, EPERM, NULL));

```

```

2126 }

2128 /*
2129 * Slightly complicated because we don't want to trigger the policy too
2130 * often. First we shortcircuit access to "self" (tp == sp) or if
2131 * we don't have the privilege but if we have permission
2132 * just return (0) and we don't flag the privilege as needed.
2133 * Else, we test for the privilege because we either have it or need it.
2134 */
2135 int
2136 secpolicy_basic_procinfo(const cred_t *cr, proc_t *tp, proc_t *sp)
2137 {
2138     if (tp == sp ||
2139         !HAS_PRIVILEGE(cr, PRIV_PROC_INFO) && prochasprocpem(tp, sp, cr)) {
2140         return (0);
2141     } else {
2142         return (PRIV_POLICY(cr, PRIV_PROC_INFO, B_FALSE, EPERM, NULL));
2143     }
2144 }

2146 int
2147 secpolicy_basic_link(const cred_t *cr)
2148 {
2149     FAST_BASIC_CHECK(cr, PRIV_FILE_LINK_ANY);

2151     return (PRIV_POLICY(cr, PRIV_FILE_LINK_ANY, B_FALSE, EPERM, NULL));
2152 }

2154 int
2155 secpolicy_basic_net_access(const cred_t *cr)
2156 {
2157     FAST_BASIC_CHECK(cr, PRIV_NET_ACCESS);

2159     return (PRIV_POLICY(cr, PRIV_NET_ACCESS, B_FALSE, EACCESS, NULL));
2160 }

2162 /* ARGSUSED */
2163 int
2164 secpolicy_basic_file_read(const cred_t *cr, vnode_t *vp, const char *pn)
2165 {
2166     FAST_BASIC_CHECK(cr, PRIV_FILE_READ);

2168     return (priv_policy_va(cr, PRIV_FILE_READ, B_FALSE, EACCESS, NULL,
2169         KLPDARG_VNODE, vp, (char *)pn, KLPDARG_NOMORE));
2170 }

2172 /* ARGSUSED */
2173 int
2174 secpolicy_basic_file_write(const cred_t *cr, vnode_t *vp, const char *pn)
2175 {
2176     FAST_BASIC_CHECK(cr, PRIV_FILE_WRITE);

2178     return (priv_policy_va(cr, PRIV_FILE_WRITE, B_FALSE, EACCESS, NULL,
2179         KLPDARG_VNODE, vp, (char *)pn, KLPDARG_NOMORE));
2180 }

2182 /*
2183 * Additional device protection.
2184 *
2185 * Traditionally, a device has specific permissions on the node in
2186 * the filesystem which govern which devices can be opened by what
2187 * processes. In certain cases, it is desirable to add extra
2188 * restrictions, as writing to certain devices is identical to
2189 * having a complete run of the system.
2190 *
2191 * This mechanism is called the device policy.

```

```

2192 *
2193 * When a device is opened, its policy entry is looked up in the
2194 * policy cache and checked.
2195 */
2196 int
2197 secpolicy_spec_open(const cred_t *cr, struct vnode *vp, int oflag)
2198 {
2199     devplcy_t *plcy;
2200     int err;
2201     struct snode *csp = VTOS(common_specvp(vp));
2202     priv_set_t pset;
2203
2204     mutex_enter(&csp->s_lock);
2205
2206     if (csp->s_plcy == NULL || csp->s_plcy->dp_gen != devplcy_gen) {
2207         plcy = devpolicy_find(vp);
2208         if (csp->s_plcy)
2209             dpfree(csp->s_plcy);
2210         csp->s_plcy = plcy;
2211         ASSERT(plcy != NULL);
2212     } else
2213         plcy = csp->s_plcy;
2214
2215     if (plcy == nullpolicy) {
2216         mutex_exit(&csp->s_lock);
2217         return (0);
2218     }
2219
2220     dphold(plcy);
2221
2222     mutex_exit(&csp->s_lock);
2223
2224     if (oflag & FWRITE)
2225         pset = plcy->dp_wrp;
2226     else
2227         pset = plcy->dp_rdp;
2228     /*
2229     * Special case:
2230     * PRIV_SYS_NET_CONFIG is a superset of PRIV_SYS_IP_CONFIG.
2231     * If PRIV_SYS_NET_CONFIG is present and PRIV_SYS_IP_CONFIG is
2232     * required, replace PRIV_SYS_IP_CONFIG with PRIV_SYS_NET_CONFIG
2233     * in the required privilege set before doing the check.
2234     */
2235     if (priv_ismember(&pset, PRIV_SYS_IP_CONFIG) &&
2236         priv_ismember(&CR_OEPRIV(cr), PRIV_SYS_NET_CONFIG) &&
2237         !priv_ismember(&CR_OEPRIV(cr), PRIV_SYS_IP_CONFIG)) {
2238         priv_delset(&pset, PRIV_SYS_IP_CONFIG);
2239         priv_addset(&pset, PRIV_SYS_NET_CONFIG);
2240     }
2241
2242     err = secpolicy_require_set(cr, &pset, "devpolicy", KLPDARG_NONE);
2243     dpfree(plcy);
2244
2245     return (err);
2246 }
2247
2248 int
2249 secpolicy_modctl(const cred_t *cr, int cmd)
2250 {
2251     switch (cmd) {
2252     case MODINFO:
2253     case MODGETMAJBIND:
2254     case MODGETPATH:
2255     case MODGETPATHLEN:
2256     case MODGETNAME:
2257     case MODGETFNAME:

```

```

2258     case MODGETDEVPOLICY:
2259     case MODGETDEVPOLICYBYNAME:
2260     case MODDEVT2INSTANCE:
2261     case MODSIZEOF_DEVID:
2262     case MODGETDEVID:
2263     case MODSIZEOF_MINORNAME:
2264     case MODGETMINORNAME:
2265     case MODGETDEVFSPATH_LEN:
2266     case MODGETDEVFSPATH:
2267     case MODGETDEVFSPATH_MI_LEN:
2268     case MODGETDEVFSPATH_MI:
2269         /* Unprivileged */
2270         return (0);
2271     case MODLOAD:
2272     case MODSETDEVPOLICY:
2273         return (secpolicy_require_set(cr, PRIV_FULLSET, NULL,
2274             KLPDARG_NONE));
2275     default:
2276         return (secpolicy_sys_config(cr, B_FALSE));
2277     }
2278 }
2279
2280 int
2281 secpolicy_console(const cred_t *cr)
2282 {
2283     return (PRIV_POLICY(cr, PRIV_SYS_DEVICES, B_FALSE, EPERM, NULL));
2284 }
2285
2286 int
2287 secpolicy_power_mgmt(const cred_t *cr)
2288 {
2289     return (PRIV_POLICY(cr, PRIV_SYS_DEVICES, B_FALSE, EPERM, NULL));
2290 }
2291
2292 /*
2293 * Simulate terminal input; another escalation of privileges avenue.
2294 */
2295
2296 int
2297 secpolicy_sti(const cred_t *cr)
2298 {
2299     return (secpolicy_require_set(cr, PRIV_FULLSET, NULL, KLPDARG_NONE));
2300 }
2301
2302 boolean_t
2303 secpolicy_net_reply_equal(const cred_t *cr)
2304 {
2305     return (PRIV_POLICY(cr, PRIV_SYS_CONFIG, B_FALSE, EPERM, NULL));
2306 }
2307
2308 int
2309 secpolicy_swapctl(const cred_t *cr)
2310 {
2311     return (PRIV_POLICY(cr, PRIV_SYS_CONFIG, B_FALSE, EPERM, NULL));
2312 }
2313
2314 int
2315 secpolicy_cpc_cpu(const cred_t *cr)
2316 {
2317     return (PRIV_POLICY(cr, PRIV_CPC_CPU, B_FALSE, EACCES, NULL));
2318 }
2319
2320 /*
2321 * secpolicy_contract_identity
2322 * Determine if the subject may set the process contract FMRI value

```

```

2324 */
2325 int
2326 secpolicy_contract_identity(const cred_t *cr)
2327 {
2328     return (PRIV_POLICY(cr, PRIV_CONTRACT_IDENTITY, B_FALSE, EPERM, NULL));
2329 }

2331 /*
2332  * secpolicy_contract_observer
2333  *
2334  * Determine if the subject may observe a specific contract's events.
2335  */
2336 int
2337 secpolicy_contract_observer(const cred_t *cr, struct contract *ct)
2338 {
2339     if (contract_owned(ct, cr, B_FALSE))
2340         return (0);
2341     return (PRIV_POLICY(cr, PRIV_CONTRACT_OBSERVER, B_FALSE, EPERM, NULL));
2342 }

2344 /*
2345  * secpolicy_contract_observer_choice
2346  *
2347  * Determine if the subject may observe any contract's events. Just
2348  * tests privilege and audits on success.
2349  */
2350 boolean_t
2351 secpolicy_contract_observer_choice(const cred_t *cr)
2352 {
2353     return (PRIV_POLICY_CHOICE(cr, PRIV_CONTRACT_OBSERVER, B_FALSE));
2354 }

2356 /*
2357  * secpolicy_contract_event
2358  *
2359  * Determine if the subject may request critical contract events or
2360  * reliable contract event delivery.
2361  */
2362 int
2363 secpolicy_contract_event(const cred_t *cr)
2364 {
2365     return (PRIV_POLICY(cr, PRIV_CONTRACT_EVENT, B_FALSE, EPERM, NULL));
2366 }

2368 /*
2369  * secpolicy_contract_event_choice
2370  *
2371  * Determine if the subject may retain contract events in its critical
2372  * set when a change in other terms would normally require a change in
2373  * the critical set. Just tests privilege and audits on success.
2374  */
2375 boolean_t
2376 secpolicy_contract_event_choice(const cred_t *cr)
2377 {
2378     return (PRIV_POLICY_CHOICE(cr, PRIV_CONTRACT_EVENT, B_FALSE));
2379 }

2381 /*
2382  * secpolicy_gart_access
2383  *
2384  * Determine if the subject has sufficient privileges to make ioctls to agpgart
2385  * device.
2386  */
2387 int
2388 secpolicy_gart_access(const cred_t *cr)
2389 {

```

```

2390     return (PRIV_POLICY(cr, PRIV_GRAPHICS_ACCESS, B_FALSE, EPERM, NULL));
2391 }

2393 /*
2394  * secpolicy_gart_map
2395  *
2396  * Determine if the subject has sufficient privileges to map aperture range
2397  * through agpgart driver.
2398  */
2399 int
2400 secpolicy_gart_map(const cred_t *cr)
2401 {
2402     if (PRIV_POLICY_ONLY(cr, PRIV_GRAPHICS_ACCESS, B_FALSE)) {
2403         return (PRIV_POLICY(cr, PRIV_GRAPHICS_ACCESS, B_FALSE, EPERM,
2404             NULL));
2405     } else {
2406         return (PRIV_POLICY(cr, PRIV_GRAPHICS_MAP, B_FALSE, EPERM,
2407             NULL));
2408     }
2409 }

2411 /*
2412  * secpolicy_zinject
2413  *
2414  * Determine if the subject can inject faults in the ZFS fault injection
2415  * framework. Requires all privileges.
2416  */
2417 int
2418 secpolicy_zinject(const cred_t *cr)
2419 {
2420     return (secpolicy_require_set(cr, PRIV_FULLSET, NULL, KLPDARG_NONE));
2421 }

2423 /*
2424  * secpolicy_zfs
2425  *
2426  * Determine if the subject has permission to manipulate ZFS datasets
2427  * (not pools). Equivalent to the SYS_MOUNT privilege.
2428  */
2429 int
2430 secpolicy_zfs(const cred_t *cr)
2431 {
2432     return (PRIV_POLICY(cr, PRIV_SYS_MOUNT, B_FALSE, EPERM, NULL));
2433 }

2435 /*
2436  * secpolicy_idmap
2437  *
2438  * Determine if the calling process has permissions to register an SID
2439  * mapping daemon and allocate ephemeral IDs.
2440  */
2441 int
2442 secpolicy_idmap(const cred_t *cr)
2443 {
2444     return (PRIV_POLICY(cr, PRIV_FILE_SETID, B_TRUE, EPERM, NULL));
2445 }

2447 /*
2448  * secpolicy_ucose_update
2449  *
2450  * Determine if the subject has sufficient privilege to update microcode.
2451  */
2452 int
2453 secpolicy_ucose_update(const cred_t *scr)
2454 {
2455     return (PRIV_POLICY(scr, PRIV_ALL, B_FALSE, EPERM, NULL));

```

```

2456 }

2458 /*
2459  * secpolicy_sadopen
2460  *
2461  * Determine if the subject has sufficient privilege to access /dev/sad/admin.
2462  * /dev/sad/admin appear in global zone and exclusive-IP zones only.
2463  * In global zone, sys_config is required.
2464  * In exclusive-IP zones, sys_ip_config is required.
2465  * Note that sys_config is prohibited in non-global zones.
2466  */
2467 int
2468 secpolicy_sadopen(const cred_t *credp)
2469 {
2470     priv_set_t pset;

2472     priv_emptyset(&pset);

2474     if (crgetzoneid(credp) == GLOBAL_ZONEID)
2475         priv_addset(&pset, PRIV_SYS_CONFIG);
2476     else
2477         priv_addset(&pset, PRIV_SYS_IP_CONFIG);

2479     return (secpolicy_require_set(credp, &pset, "devpolicy", KLPDARG_NONE));
2480 }

2483 /*
2484  * Add privileges to a particular privilege set; this is called when the
2485  * current sets of privileges are not sufficient. I.e., we should always
2486  * call the policy override functions from here.
2487  * What we are allowed to have is in the Observed Permitted set; so
2488  * we compute the difference between that and the newset.
2489  */
2490 int
2491 secpolicy_require_privs(const cred_t *cr, const priv_set_t *nset)
2492 {
2493     priv_set_t rqd;

2495     rqd = CR_OPPriv(cr);

2497     priv_inverse(&rqd);
2498     priv_intersect(nset, &rqd);

2500     return (secpolicy_require_set(cr, &rqd, NULL, KLPDARG_NONE));
2501 }

2503 /*
2504  * secpolicy_smb
2505  *
2506  * Determine if the cred_t has PRIV_SYS_SMB privilege, indicating
2507  * that it has permission to access the smbsrv kernel driver.
2508  * PRIV_POLICY checks the privilege and audits the check.
2509  *
2510  * Returns:
2511  * 0      Driver access is allowed.
2512  * EPERM  Driver access is NOT permitted.
2513  */
2514 int
2515 secpolicy_smb(const cred_t *cr)
2516 {
2517     return (PRIV_POLICY(cr, PRIV_SYS_SMB, B_FALSE, EPERM, NULL));
2518 }

2520 /*
2521  * secpolicy_vscan

```

```

2522  *
2523  * Determine if cred_t has the necessary privileges to access a file
2524  * for virus scanning and update its extended system attributes.
2525  * PRIV_FILE_DAC_SEARCH, PRIV_FILE_DAC_READ - file access
2526  * PRIV_FILE_FLAG_SET - set extended system attributes
2527  *
2528  * PRIV_POLICY checks the privilege and audits the check.
2529  *
2530  * Returns:
2531  * 0      file access for virus scanning allowed.
2532  * EPERM  file access for virus scanning is NOT permitted.
2533  */
2534 int
2535 secpolicy_vscan(const cred_t *cr)
2536 {
2537     if ((PRIV_POLICY(cr, PRIV_FILE_DAC_SEARCH, B_FALSE, EPERM, NULL)) ||
2538         (PRIV_POLICY(cr, PRIV_FILE_DAC_READ, B_FALSE, EPERM, NULL)) ||
2539         (PRIV_POLICY(cr, PRIV_FILE_FLAG_SET, B_FALSE, EPERM, NULL))) {
2540         return (EPERM);
2541     }

2543     return (0);
2544 }

2546 /*
2547  * secpolicy_smbfs_login
2548  *
2549  * Determines if the caller can add and delete the smbfs login
2550  * password in the the nsmb kernel module for the CIFS client.
2551  *
2552  * Returns:
2553  * 0      access is allowed.
2554  * EPERM  access is NOT allowed.
2555  */
2556 int
2557 secpolicy_smbfs_login(const cred_t *cr, uid_t uid)
2558 {
2559     uid_t cruid = crgetruid(cr);

2561     if (cruid == uid)
2562         return (0);
2563     return (PRIV_POLICY(cr, PRIV_PROC_OWNER, B_FALSE,
2564         EPERM, NULL));
2565 }

2567 /*
2568  * secpolicy_xvm_control
2569  *
2570  * Determines if a caller can control the xvm hypervisor and/or running
2571  * domains (x86 specific).
2572  *
2573  * Returns:
2574  * 0      access is allowed.
2575  * EPERM  access is NOT allowed.
2576  */
2577 int
2578 secpolicy_xvm_control(const cred_t *cr)
2579 {
2580     if (PRIV_POLICY(cr, PRIV_XVM_CONTROL, B_FALSE, EPERM, NULL))
2581         return (EPERM);
2582     return (0);
2583 }

2585 /*
2586  * secpolicy_ppp_config
2587  *

```

```
2588 * Determine if the subject has sufficient privileges to configure PPP and
2589 * PPP-related devices.
2590 */
2591 int
2592 secpolicy_ppp_config(const cred_t *cr)
2593 {
2594     if (PRIV_POLICY_ONLY(cr, PRIV_SYS_NET_CONFIG, B_FALSE))
2595         return (secpolicy_net_config(cr, B_FALSE));
2596     return (PRIV_POLICY(cr, PRIV_SYS_PPP_CONFIG, B_FALSE, EPERM, NULL));
2597 }
```

```

*****
17612 Thu Jun 30 22:03:31 2016
new/usr/src/uts/common/os/priv.c
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
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 2010 Sun Microsystems, Inc. All rights reserved.
23 * Use is subject to license terms.
24 */

26 /*
27  * Privilege implementation.
28  *
29  * This file provides the infrastructure for privilege sets and limits
30  * the number of files that requires to include <sys/cred_impl.h> and/or
31  * <sys/priv_impl.h>.
32  *
33  * The Solaris privilege mechanism has been designed in a
34  * future proof manner. While the kernel may use fixed size arrays
35  * and fixed bitmasks and bit values, the representation of those
36  * is kernel private. All external interfaces as well as K-to-K interfaces
37  * have been constructed in a manner to provide the maximum flexibility.
38  *
39  * There can be X privilege sets each containing Y 32 bit words.
40  * <X, Y> are constant for a kernel invocation.
41  *
42  * As a consequence, all privilege set manipulation happens in functions
43  * below.
44  *
45  */

47 #include <sys/system.h>
48 #include <sys/ddi.h>
49 #include <sys/kmem.h>
50 #include <sys/sunddi.h>
51 #include <sys/errno.h>
52 #include <sys/debug.h>
53 #include <sys/priv_impl.h>
54 #include <sys/procfs.h>
55 #include <sys/policy.h>
56 #include <sys/cred_impl.h>
57 #include <sys/devpolicy.h>
58 #include <sys/atomic.h>

```

```

60 /*
61  * Privilege name to number mapping table consists in the generated
62  * priv_const.c file. This lock protects against updates of the privilege
63  * names and counts; all other priv_info fields are read-only.
64  * The actual protected values are:
65  *   global variable nprivs
66  *   the priv_max field
67  *   the priv_names field
68  *   the priv names info item (cnt/strings)
69  */
70 krwlock_t privinfo_lock;

72 static boolean_t priv_valid(const cred_t *);

74 priv_set_t priv_fullset; /* set of all privileges */
75 priv_set_t priv_unsafe; /* unsafe to exec set-uid root if these are not in L */

77 /*
78  * Privilege initialization functions.
79  * Called from common/os/cred.c when cred_init is called.
80  */

82 void
83 priv_init(void)
84 {
85 #ifdef DEBUG
86     int alloc_test_priv = 1;
87 #else
88     int alloc_test_priv = priv_debug;
89 #endif
90     rw_init(&privinfo_lock, NULL, RW_DRIVER, NULL);

92     PRIV_BASIC_ADDSET(priv_basic);
93     PRIV_UNSAFE_ADDSET(&priv_unsafe);
94     PRIV_BASIC_ASSERT(priv_basic);
95     PRIV_UNSAFE_ASSERT(&priv_unsafe);
96     priv_fillset(&priv_fullset);

97     /*
98      * When booting with priv_debug set or in a DEBUG kernel, then we'll
99      * add an additional basic privilege and we verify that it is always
100      * present in E.
101      */
102     if (alloc_test_priv != 0 &&
103         (priv_basic_test = priv_getbyname("basic_test", PRIV_ALLOC)) >= 0) {
104         priv_addset(priv_basic, priv_basic_test);
105     }

106     devpolicy_init();
107 }

    unchanged portion omitted

479 void
480 priv_addset(priv_set_t *set, int priv)
481 {
482     ASSERT(priv >= 0 && priv < MAX_PRIVILEGE);
483     __PRIV_ADDSET(set, priv);
484     __PRIV_ASSERT(set, priv);
485 }

486 void
487 priv_delset(priv_set_t *set, int priv)
488 {
489     ASSERT(priv >= 0 && priv < MAX_PRIVILEGE);
490     __PRIV_DELSET(set, priv);

```



```
490     __PRIV_CLEAR(set, priv);
491 }

493 boolean_t
494 priv_ismember(const priv_set_t *set, int priv)
495 {
496     ASSERT(priv >= 0 && priv < MAX_PRIVILEGE);
497     return (__PRIV_ISMEMBER(set, priv) ? B_TRUE : B_FALSE);
497     return (__PRIV_ISASSERT(set, priv) ? B_TRUE : B_FALSE);
498 }
    unchanged_portion_omitted_
```

new/usr/src/uts/common/os/priv_defs

1

```
*****
20918 Thu Jun 30 22:03:32 2016
new/usr/src/uts/common/os/priv_defs
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
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 (c) 2003, 2010, Oracle and/or its affiliates. All rights reserved.
23 * Copyright 2015, Joyent, Inc. All rights reserved.
24 *
25 INSERT COMMENT
26 */

28 #
29 # Privileges can be added to this file at any location, not
30 # necessarily at the end. For patches, it is probably best to
31 # add the new privilege at the end; for ordinary releases privileges
32 # should be ordered alphabetically.
33 #

35 privilege PRIV_CONTRACT_EVENT

37     Allows a process to request critical events without limitation.
38     Allows a process to request reliable delivery of all events on
39     any event queue.

41 privilege PRIV_CONTRACT_IDENTITY

43     Allows a process to set the service FMRI value of a process
44     contract template.

46 privilege PRIV_CONTRACT_OBSERVER

48     Allows a process to observe contract events generated by
49     contracts created and owned by users other than the process's
50     effective user ID.
51     Allows a process to open contract event endpoints belonging to
52     contracts created and owned by users other than the process's
53     effective user ID.

55 privilege PRIV_CPC_CPU

57     Allow a process to access per-CPU hardware performance counters.
```

new/usr/src/uts/common/os/priv_defs

2

```
59 privilege PRIV_DTRACE_KERNEL

61     Allows DTrace kernel-level tracing.

63 privilege PRIV_DTRACE_PROC

65     Allows DTrace process-level tracing.
66     Allows process-level tracing probes to be placed and enabled in
67     processes to which the user has permissions.

69 privilege PRIV_DTRACE_USER

71     Allows DTrace user-level tracing.
72     Allows use of the syscall and profile DTrace providers to
73     examine processes to which the user has permissions.

75 privilege PRIV_FILE_CHOWN

77     Allows a process to change a file's owner user ID.
78     Allows a process to change a file's group ID to one other than
79     the process' effective group ID or one of the process'
80     supplemental group IDs.

82 privilege PRIV_FILE_CHOWN_SELF

84     Allows a process to give away its files; a process with this
85     privilege will run as if {_POSIX_CHOWN_RESTRICTED} is not
86     in effect.

88 privilege PRIV_FILE_DAC_EXECUTE

90     Allows a process to execute an executable file whose permission
91     bits or ACL do not allow the process execute permission.

93 privilege PRIV_FILE_DAC_READ

95     Allows a process to read a file or directory whose permission
96     bits or ACL do not allow the process read permission.

98 privilege PRIV_FILE_DAC_SEARCH

100     Allows a process to search a directory whose permission bits or
101     ACL do not allow the process search permission.

103 privilege PRIV_FILE_DAC_WRITE

105     Allows a process to write a file or directory whose permission
106     bits or ACL do not allow the process write permission.
107     In order to write files owned by uid 0 in the absence of an
108     effective uid of 0 ALL privileges are required.

110 privilege PRIV_FILE_DOWNGRADE_SL

112     Allows a process to set the sensitivity label of a file or
113     directory to a sensitivity label that does not dominate the
114     existing sensitivity label.
115     This privilege is interpreted only if the system is configured
116     with Trusted Extensions.

118 privilege PRIV_FILE_FLAG_SET

120     Allows a process to set immutable, nounlink or appendonly
121     file attributes.

123 basic privilege PRIV_FILE_LINK_ANY
```

125 Allows a process to create hardlinks to files owned by a uid
 126 different from the process' effective uid.

128 privilege PRIV_FILE_OWNER

130 Allows a process which is not the owner of a file or directory
 131 to perform the following operations that are normally permitted
 132 only for the file owner: modify that file's access and
 133 modification times; remove or rename a file or directory whose
 134 parent directory has the 'save text image after execution'
 135 (sticky) bit set; mount a 'namefs' upon a file; modify
 136 permission bits or ACL except for the set-uid and set-gid
 137 bits.

139 basic privilege PRIV_FILE_READ

141 Allows a process to read objects in the filesystem.

143 privilege PRIV_FILE_SETID

145 Allows a process to change the ownership of a file or write to
 146 a file without the set-user-ID and set-group-ID bits being
 147 cleared.
 148 Allows a process to set the set-group-ID bit on a file or
 149 directory whose group is not the process' effective group or
 150 one of the process' supplemental groups.
 151 Allows a process to set the set-user-ID bit on a file with
 152 different ownership in the presence of PRIV_FILE_OWNER.
 153 Additional restrictions apply when creating or modifying a
 154 set-uid 0 file.

156 privilege PRIV_FILE_UPGRADE_SL

158 Allows a process to set the sensitivity label of a file or
 159 directory to a sensitivity label that dominates the existing
 160 sensitivity label.
 161 This privilege is interpreted only if the system is configured
 162 with Trusted Extensions.

164 basic privilege PRIV_FILE_WRITE

166 Allows a process to modify objects in the filesystem.

168 privilege PRIV_GRAPHICS_ACCESS

170 Allows a process to make privileged ioctls to graphics devices.
 171 Typically only xserver process needs to have this privilege.
 172 A process with this privilege is also allowed to perform
 173 privileged graphics device mappings.

175 privilege PRIV_GRAPHICS_MAP

177 Allows a process to perform privileged mappings through a
 178 graphics device.

180 privilege PRIV_IPC_DAC_READ

182 Allows a process to read a System V IPC
 183 Message Queue, Semaphore Set, or Shared Memory Segment whose
 184 permission bits do not allow the process read permission.
 185 Allows a process to read remote shared memory whose
 186 permission bits do not allow the process read permission.

188 privilege PRIV_IPC_DAC_WRITE

190 Allows a process to write a System V IPC

191 Message Queue, Semaphore Set, or Shared Memory Segment whose
 192 permission bits do not allow the process write permission.
 193 Allows a process to read remote shared memory whose
 194 permission bits do not allow the process write permission.
 195 Additional restrictions apply if the owner of the object has uid 0
 196 and the effective uid of the current process is not 0.

198 privilege PRIV_IPC_OWNER

200 Allows a process which is not the owner of a System
 201 V IPC Message Queue, Semaphore Set, or Shared Memory Segment to
 202 remove, change ownership of, or change permission bits of the
 203 Message Queue, Semaphore Set, or Shared Memory Segment.
 204 Additional restrictions apply if the owner of the object has uid 0
 205 and the effective uid of the current process is not 0.

207 basic privilege PRIV_NET_ACCESS

209 Allows a process to open a TCP, UDP, SDP or SCTP network endpoint.

211 privilege PRIV_NET_BINDMLP

213 Allow a process to bind to a port that is configured as a
 214 multi-level port(MLP) for the process's zone. This privilege
 215 applies to both shared address and zone-specific address MLPs.
 216 See tzzonecfg(4) from the Trusted Extensions manual pages for
 217 information on configuring MLP ports.
 218 This privilege is interpreted only if the system is configured
 219 with Trusted Extensions.

221 privilege PRIV_NET_ICMPACCESS

223 Allows a process to send and receive ICMP packets.

225 privilege PRIV_NET_MAC_AWARE

227 Allows a process to set NET_MAC_AWARE process flag by using
 228 setpflags(2). This privilege also allows a process to set
 229 SO_MAC_EXEMPT socket option by using setsockopt(3SOCKET).
 230 The NET_MAC_AWARE process flag and the SO_MAC_EXEMPT socket
 231 option both allow a local process to communicate with an
 232 unlabeled peer if the local process' label dominates the
 233 peer's default label, or if the local process runs in the
 234 global zone.
 235 This privilege is interpreted only if the system is configured
 236 with Trusted Extensions.

238 privilege PRIV_NET_MAC_IMPLICIT

240 Allows a process to set SO_MAC_IMPLICIT option by using
 241 setsockopt(3SOCKET). This allows a privileged process to
 242 transmit implicitly-labeled packets to a peer.
 243 This privilege is interpreted only if the system is configured
 244 with Trusted Extensions.

246 privilege PRIV_NET_OBSERVABILITY

248 Allows a process to access /dev/lo0 and the devices in /dev/ipnet/
 249 while not requiring them to need PRIV_NET_RAWACCESS.

251 privilege PRIV_NET_PRIVADDR

253 Allows a process to bind to a privileged port
 254 number. The privilege port numbers are 1-1023 (the traditional
 255 UNIX privileged ports) as well as those ports marked as
 256 "udp/tcp_extra_priv_ports" with the exception of the ports

```

257 reserved for use by NFS.
259 privilege PRIV_NET_RAWACCESS
261 Allows a process to have direct access to the network layer.
263 unsafe privilege PRIV_PROC_AUDIT
265 Allows a process to generate audit records.
266 Allows a process to get its own audit pre-selection information.
268 privilege PRIV_PROC_CHROOT
270 Allows a process to change its root directory.
272 privilege PRIV_PROC_CLOCK_HIGHRES
274 Allows a process to use high resolution timers.
276 basic privilege PRIV_PROC_EXEC
278 Allows a process to call execve().
280 basic privilege PRIV_PROC_FORK
282 Allows a process to call fork1()/forkall()/vfork()
284 basic privilege PRIV_PROC_INFO
286 Allows a process to examine the status of processes other
287 than those it can send signals to. Processes which cannot
288 be examined cannot be seen in /proc and appear not to exist.
290 privilege PRIV_PROC_LOCK_MEMORY
292 Allows a process to lock pages in physical memory.
294 privilege PRIV_PROC_MEMINFO
296 Allows a process to access physical memory information.
298 privilege PRIV_PROC_OWNER
300 Allows a process to send signals to other processes, inspect
301 and modify process state to other processes regardless of
302 ownership. When modifying another process, additional
303 restrictions apply: the effective privilege set of the
304 attaching process must be a superset of the target process'
305 effective, permitted and inheritable sets; the limit set must
306 be a superset of the target's limit set; if the target process
307 has any uid set to 0 all privilege must be asserted unless the
308 effective uid is 0.
309 Allows a process to bind arbitrary processes to CPUs.
311 privilege PRIV_PROC_PRIROUP
313 Allows a process to elevate its priority above its current level.
315 privilege PRIV_PROC_PRIOCNTL
317 Allows all that PRIV_PROC_PRIROUP allows.
318 Allows a process to change its scheduling class to any scheduling class,
319 including the RT class.
321 basic privilege PRIV_PROC_SECFLAGS

```

```

323 Allows a process to manipulate the secflags of processes (subject to,
324 additionally, the ability to signal that process)
326 #endif /* ! codereview */
327 basic privilege PRIV_PROC_SESSION
329 Allows a process to send signals or trace processes outside its
330 session.
332 unsafe privilege PRIV_PROC_SETID
334 Allows a process to set its uids at will.
335 Assuming uid 0 requires all privileges to be asserted.
337 privilege PRIV_PROC_TASKID
339 Allows a process to assign a new task ID to the calling process.
341 privilege PRIV_PROC_ZONE
343 Allows a process to trace or send signals to processes in
344 other zones.
346 privilege PRIV_SYS_ACCT
348 Allows a process to enable and disable and manage accounting through
349 acct(2), getacct(2), putacct(2) and wracct(2).
351 privilege PRIV_SYS_ADMIN
353 Allows a process to perform system administration tasks such
354 as setting node and domain name and specifying nscd and coreadm
355 settings.
357 privilege PRIV_SYS_AUDIT
359 Allows a process to start the (kernel) audit daemon.
360 Allows a process to view and set audit state (audit user ID,
361 audit terminal ID, audit sessions ID, audit pre-selection mask).
362 Allows a process to turn off and on auditing.
363 Allows a process to configure the audit parameters (cache and
364 queue sizes, event to class mappings, policy options).
366 privilege PRIV_SYS_CONFIG
368 Allows a process to perform various system configuration tasks.
369 Allows a process to add and remove swap devices; when adding a swap
370 device, a process must also have sufficient privileges to read from
371 and write to the swap device.
373 privilege PRIV_SYS_DEVICES
375 Allows a process to successfully call a kernel module that
376 calls the kernel drv_priv(9F) function to check for allowed
377 access.
378 Allows a process to open the real console device directly.
379 Allows a process to open devices that have been exclusively opened.
381 privilege PRIV_SYS_IPC_CONFIG
383 Allows a process to increase the size of a System V IPC Message
384 Queue buffer.
386 privilege PRIV_SYS_LINKDIR
388 Allows a process to unlink and link directories.

```

```

390 privilege PRIV_SYS_MOUNT

392     Allows filesystem specific administrative procedures, such as
393     filesystem configuration ioctls, quota calls and creation/deletion
394     of snapshots.
395     Allows a process to mount and unmount filesystems which would
396     otherwise be restricted (i.e., most filesystems except
397     namefs).
398     A process performing a mount operation needs to have
399     appropriate access to the device being mounted (read-write for
400     "rw" mounts, read for "ro" mounts).
401     A process performing any of the aforementioned
402     filesystem operations needs to have read/write/owner
403     access to the mount point.
404     Only regular files and directories can serve as mount points
405     for processes which do not have all zone privileges asserted.
406     Unless a process has all zone privileges, the mount(2)
407     system call will force the "nosuid" and "restrict" options, the
408     latter only for autofs mountpoints.
409     Regardless of privileges, a process running in a non-global zone may
410     only control mounts performed from within said zone.
411     Outside the global zone, the "nodevices" option is always forced.

413 privilege PRIV_SYS_IPTUN_CONFIG

415     Allows a process to configure IP tunnel links.

417 privilege PRIV_SYS_DL_CONFIG

419     Allows a process to configure all classes of datalinks, including
420     configuration allowed by PRIV_SYS_IPTUN_CONFIG.

422 privilege PRIV_SYS_IP_CONFIG

424     Allows a process to configure a system's IP interfaces and routes.
425     Allows a process to configure network parameters using ndd.
426     Allows a process access to otherwise restricted information using ndd.
427     Allows a process to configure IPsec.
428     Allows a process to pop anchored STREAMS modules with matching zoneid.

430 privilege PRIV_SYS_NET_CONFIG

432     Allows all that PRIV_SYS_IP_CONFIG, PRIV_SYS_DL_CONFIG, and
433     PRIV_SYS_PPP_CONFIG allow.
434     Allows a process to push the rpcmod STREAMS module.
435     Allows a process to INSERT/REMOVE STREAMS modules on locations other
436     than the top of the module stack.

438 privilege PRIV_SYS_NFS

440     Allows a process to perform Sun private NFS specific system calls.
441     Allows a process to bind to ports reserved by NFS: ports 2049 (nfs)
442     and port 4045 (lockd).

444 privilege PRIV_SYS_PPP_CONFIG

446     Allows a process to create and destroy PPP (sppp) interfaces.
447     Allows a process to configure PPP tunnels (spptun).

449 privilege PRIV_SYS_RES_BIND

451     Allows a process to bind processes to processor sets.

453 privilege PRIV_SYS_RES_CONFIG

```

```

455     Allows all that PRIV_SYS_RES_BIND allows.
456     Allows a process to create and delete processor sets, assign
457     CPUs to processor sets and override the PSET_NOESCAPE property.
458     Allows a process to change the operational status of CPUs in
459     the system using p_online(2).
460     Allows a process to configure resource pools and to bind
461     processes to pools

463 unsafe privilege PRIV_SYS_RESOURCE

465     Allows a process to modify the resource limits specified
466     by setrlimit(2) and setrctl(2) without restriction.
467     Allows a process to exceed the per-user maximum number of
468     processes.
469     Allows a process to extend or create files on a filesystem that
470     has less than minfree space in reserve.

472 privilege PRIV_SYS_SMB

474     Allows a process to access the Sun private SMB kernel module.
475     Allows a process to bind to ports reserved by NetBIOS and SMB:
476     ports 137 (NBNS), 138 (NetBIOS Datagram Service), 139 (NetBIOS
477     Session Service and SMB-over-NBT) and 445 (SMB-over-TCP).

479 privilege PRIV_SYS_SUSER_COMPAT

481     Allows a process to successfully call a third party loadable module
482     that calls the kernel suser() function to check for allowed access.
483     This privilege exists only for third party loadable module
484     compatibility and is not used by Solaris proper.

486 privilege PRIV_SYS_TIME

488     Allows a process to manipulate system time using any of the
489     appropriate system calls: stime, adjtime, ntp_adjtime and
490     the IA specific RTC calls.

492 privilege PRIV_SYS_TRANS_LABEL

494     Allows a process to translate labels that are not dominated
495     by the process' sensitivity label to and from an external
496     string form.
497     This privilege is interpreted only if the system is configured
498     with Trusted Extensions.

500 privilege PRIV_VIRT_MANAGE

502     Allows a process to manage virtualized environments such as
503     xVM(5).

505 privilege PRIV_WIN_COLORMAP

507     Allows a process to override colormap restrictions.
508     Allows a process to install or remove colormaps.
509     Allows a process to retrieve colormap cell entries allocated
510     by other processes.
511     This privilege is interpreted only if the system is configured
512     with Trusted Extensions.

514 privilege PRIV_WIN_CONFIG

516     Allows a process to configure or destroy resources that are
517     permanently retained by the X server.
518     Allows a process to use SetScreenSaver to set the screen
519     saver timeout value.
520     Allows a process to use ChangeHosts to modify the display

```

```

521 access control list.
522 Allows a process to use GrabServer.
523 Allows a process to use the SetCloseDownMode request which
524 may retain window, pixmap, colormap, property, cursor, font,
525 or graphic context resources.
526 This privilege is interpreted only if the system is configured
527 with Trusted Extensions.

529 privilege PRIV_WIN_DAC_READ

531 Allows a process to read from a window resource that it does
532 not own (has a different user ID).
533 This privilege is interpreted only if the system is configured
534 with Trusted Extensions.

536 privilege PRIV_WIN_DAC_WRITE

538 Allows a process to write to or create a window resource that
539 it does not own (has a different user ID). A newly created
540 window property is created with the window's user ID.
541 This privilege is interpreted only if the system is configured
542 with Trusted Extensions.

544 privilege PRIV_WIN_DEVICES

546 Allows a process to perform operations on window input devices.
547 Allows a process to get and set keyboard and pointer controls.
548 Allows a process to modify pointer button and key mappings.
549 This privilege is interpreted only if the system is configured
550 with Trusted Extensions.

552 privilege PRIV_WIN_DGA

554 Allows a process to use the direct graphics access (DGA) X protocol
555 extensions. Direct process access to the frame buffer is still
556 required. Thus the process must have MAC and DAC privileges that
557 allow access to the frame buffer, or the frame buffer must be
558 allocated to the process.
559 This privilege is interpreted only if the system is configured
560 with Trusted Extensions.

562 privilege PRIV_WIN_DOWNGRADE_SL

564 Allows a process to set the sensitivity label of a window resource
565 to a sensitivity label that does not dominate the existing
566 sensitivity label.
567 This privilege is interpreted only if the system is configured
568 with Trusted Extensions.

570 privilege PRIV_WIN_FONTPATH

572 Allows a process to set a font path.
573 This privilege is interpreted only if the system is configured
574 with Trusted Extensions.

576 privilege PRIV_WIN_MAC_READ

578 Allows a process to read from a window resource whose sensitivity
579 label is not equal to the process sensitivity label.
580 This privilege is interpreted only if the system is configured
581 with Trusted Extensions.

583 privilege PRIV_WIN_MAC_WRITE

585 Allows a process to create a window resource whose sensitivity
586 label is not equal to the process sensitivity label.

```

```

587 A newly created window property is created with the window's
588 sensitivity label.
589 This privilege is interpreted only if the system is configured
590 with Trusted Extensions.

592 privilege PRIV_WIN_SELECTION

594 Allows a process to request inter-window data moves without the
595 intervention of the selection confirmer.
596 This privilege is interpreted only if the system is configured
597 with Trusted Extensions.

599 privilege PRIV_WIN_UPGRADE_SL

601 Allows a process to set the sensitivity label of a window
602 resource to a sensitivity label that dominates the existing
603 sensitivity label.
604 This privilege is interpreted only if the system is configured
605 with Trusted Extensions.

607 privilege PRIV_XVM_CONTROL

609 Allows a process access to the xVM(5) control devices for
610 managing guest domains and the hypervisor. This privilege is
611 used only if booted into xVM on x86 platforms.

613 set PRIV_EFFECTIVE

615 Set of privileges currently in effect.

617 set PRIV_INHERITABLE

619 Set of privileges that comes into effect on exec.

621 set PRIV_PERMITTED

623 Set of privileges that can be put into the effective set without
624 restriction.

626 set PRIV_LIMIT

628 Set of privileges that determines the absolute upper bound of
629 privileges this process and its off-spring can obtain.

```

```

*****
11110 Thu Jun 30 22:03:33 2016
new/usr/src/uts/common/os/privs.awk
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
1 #
2 # Copyright 2005 Sun Microsystems, Inc. All rights reserved.
3 # Use is subject to license terms.
4 #
5 # CDDL HEADER START
6 #
7 # The contents of this file are subject to the terms of the
8 # Common Development and Distribution License, Version 1.0 only
9 # (the "License"). You may not use this file except in compliance
10 # with the License.
11 #
12 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
13 # or http://www.opensolaris.org/os/licensing.
14 # See the License for the specific language governing permissions
15 # and limitations under the License.
16 #
17 # When distributing Covered Code, include this CDDL HEADER in each
18 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
19 # If applicable, add the following below this CDDL HEADER, with the
20 # fields enclosed by brackets "[]" replaced with your own identifying
21 # information: Portions Copyright [yyyy] [name of copyright owner]
22 #
23 # CDDL HEADER END
24 #
25 #ident "%Z%M% %I% %E% SMI"
26 #
27 # This file generates three different C files:
28 #
29 # <sys/priv_const.h>
30 # An implementation private set of manifest integer constant
31 # for privileges and privilege sets and manifest constants for
32 # set size, number of sets, number of privileges
33 #
34 # os/priv_const.c
35 # A C source file containing the set names, privilege names
36 # arrays for the name <-> number mappings
37 #
38 # <sys/priv_names.h>
39 # A public header file containing the PRIV_* defines
40 # that map to strings; these are for convenience.
41 # (it's easy to misspell a string, harder to misspell a
42 # manifest constant)
43 #
44 # /etc/security/priv_names
45 # A privilege name to explanation mapping.
46 #
47 #
48 # The files are output on the awk variable privhfile, pubhfile, cfile,
49 # and pnamesfile respectively
50 #
51 # The input file should contain a standard Sun comment and ident string
52 # which is copied verbatim and lines of
53 #
54 # [keyword] privilege PRIV_<privilege>
55 # set PRIV_<set>
56 #
57 # Which are converted to privileges and privilege sets

```

```

58 #
59 #
60 #
61 BEGIN {
62     # Number of privileges read
63     npriv = 0
64 #
65     # Number of privilege sets
66     nset = 0
67 #
68     # Length of all strings concatenated, including \0
69     privbytes = 0
70     setbytes = 0
71 #
72     # Number of reserved privilege slots
73     slack = 10
74 #
75     privhcmnt = \
76     " * Privilege constant definitions; these constants are subject to\n" \
77     " * change, including renumbering, without notice and should not be\n" \
78     " * used in any code. Privilege names must be used instead.\n" \
79     " * Privileges and privilege sets must not be stored in binary\n" \
80     " * form; privileges and privilege sets must be converted to\n" \
81     " * textual representation before being committed to persistent store."
82 #
83     ccmt = \
84     " * Privilege name table and size definitions."
85 #
86     pubhcmnt = \
87     " * Privilege constant definitions. Privileges and privilege sets\n" \
88     " * are only known by name and should be mapped at runtime."
89 #
90     pnamescmnt = \
91     "#\n" \
92     "# Privilege name explanation file\n" \
93     "# The format of entries is a privilege name starting at the\n" \
94     "# beginning of a line directly followed by a new line followed\n" \
95     "# by several lines of texts starting with white space terminated\n" \
96     "# by a line with a single newline or not starting with white space\n" \
97     "#\n"
98 }
99 #
100 #
101 # Privilege strings are represented as lower case strings;
102 # PRIV_ is stripped from the strings.
103 #
104 /^[A-Za-z]* )?privilege / {
105     if (NF == 3) {
106         key = toupper($1)
107         priv = toupper($3)
108         if (set[key] != "")
109             set[key] = set[key] ";"
110         set[key] = set[key] "\\n\t\tPRIV_ADDSET((set), " priv ")"
111         set[key] = set[key] "\\n\t\tPRIV_ASSERT((set), " priv ")"
112     } else {
113         priv = toupper($2);
114     }
115     privs[npriv] = tolower(substr(priv, 6));
116     inset = 0
117     inpriv = 1
118 #
119 #
120     tabs = (32 - length(priv) - 1)/8
121     # length + \0 - PRIV_
122     privbytes += length(priv) - 4

```

```
123     pdef[npriv] = "#define\t" priv substr("\t\t\t\t\t", 1, tabs)
125     npriv++
126     next
127 }
_____
213 END     {
215     if (!pubhfile && !privhfile && !cfile && !pnamesfile) {
216         print "Output file parameter not set" > "/dev/stderr"
217         exit 1
218     }
220     setsize = int((npriv + slack)/(8 * 4)) + 1
221     maxnpriv = setsize * 8 * 4
222     # Assume allocated privileges are on average "NSDQ" bytes larger.
223     maxprivbytes = int((privbytes / npriv + 5.5)) * (maxnpriv - npriv)
224     maxprivbytes += privbytes
226     if (cfile) {
227         print "\n" > cfile
228         print pragma "\n" > cfile
229         print "#include <sys/types.h>" > cfile
230         print "#include <sys/priv_const.h>" > cfile
231         print "#include <sys/priv_impl.h>" > cfile
232         print "#include <sys/priv.h>" > cfile
233         print "#include <sys/sysmacros.h>" > cfile
234         print "\n" > cfile
235         #
236         # Create the entire priv info structure here.
237         # When adding privileges, the kernel needs to update
238         # too many fields as the number of privileges is kept in
239         # many places.
240         #
241         print \
242             "static struct _info {\n" \
243             "    priv_impl_info_t     impl_info;\n" \
244             "    priv_info_t           settype;\n" \
245             "    int                   nsets;\n" \
246             "    const char           sets[\" setbytes \"]; \n" \
247             "    priv_info_t         privtype;\n" \
248             "    int                   nprivs;\n" \
249             "    char                 privs[\" maxprivbytes \"]; \n" \
250             "    priv_info_t         sysset;\n" \
251             "    priv_set_t           basicset;\n" \
252             "} info = {\n" \
253             "    { sizeof (priv_impl_info_t), 0, PRIV_NSET, " \
254             "    PRIV_SETSIZE, " npriv ", \n" \
255             "    \t\t\t\t\t sizeof (priv_info_uint_t), \n" \
256             "    \t\t\t\t\t (sizeof (info) - sizeof (info.impl_info)), \n" \
257             "    { PRIV_INFO_SETNAMES, \n" \
258             "    \t\t\t\t\t offsetof(struct _info, privtype) - " \
259             "    \t\t\t\t\t offsetof(struct _info, settype)}, \n" \
261     sep = "\t\t"
262     len = 9;
263     for (i = 0; i < nset; i++) {
264         if (len + length(sets[i]) > 80) {
265             sep = "\\0\\n\\t\\n"
266             len = 9
267         }
268         printf sep sets[i] > cfile
269         len += length(sets[i]) + length(sep);
270         sep = "\\0"
271     }
}
```

```
272     print "\\0\\n," > cfile
274     print "\t{ PRIV_INFO_PRIVNAMES, \n\t" \
275         "    \t\t\t\t\t offsetof(struct _info, sysset) - " \
276         "    \t\t\t\t\t offsetof(struct _info, privtype)}, \n\t" npriv ", " \
277         > cfile
279     sep = "\t\t"
280     len = 9;
281     for (i = 0; i < npriv; i++) {
282         if (len + length(privs[i]) > 80) {
283             sep = "\\0\\n\\n\t\t"
284             len = 9
285         }
286         printf sep privs[i] > cfile
287         len += length(privs[i]) + length(sep);
288         sep = "\\0"
289     }
290     print "\\0\\n," > cfile
292     print "\t{ PRIV_INFO_BASICPRIVS, sizeof (info) - " \
293         "    \t\t\t\t\t offsetof(struct _info, sysset)}, " > cfile
295     print "}; \n" > cfile
297     print "\nconst char *priv_names[\" maxnpriv \"] = \n{" > cfile
298     for (i = 0; i < npriv; i++)
299         print "\t&info.privs[\" privind[i] \"]," > cfile
301     print "}; \n" > cfile
303     print "\nconst char *priv_setnames[\" nset \"] = \n{" > cfile
304     for (i = 0; i < nset; i++)
305         print "\t&info.sets[\" setind[i] \"]," > cfile
307     print "}; \n" > cfile
309     print "int nprivs = " npriv ";" > cfile
310     print "int privbytes = " privbytes ";" > cfile
311     print "int maxprivbytes = " maxprivbytes ";" > cfile
312     print "size_t privinfosize = sizeof (info);" > cfile
313     print "char *priv_str = info.privs;" > cfile
314     print "priv_set_t *priv_basic = &info.basicset;" > cfile
315     print "priv_impl_info_t *priv_info = &info.impl_info;" > cfile
316     print "priv_info_names_t *priv_ninfo = " \
317         "(priv_info_names_t *)&info.privtype;" > cfile
318     close(cfile)
319 }
321 # Kernel private
322 if (privhfile) {
323     print "#ifndef _SYS_PRIV_CONST_H" > privhfile
324     print "#define\t\t\t\t\t _SYS_PRIV_CONST_H" > privhfile
325     print pragma "\n" > privhfile
326     print "\n#include <sys/types.h>\n\n" > privhfile
327     print "#ifdef __cplusplus\nextern \"C\" {\n#endif\n" > privhfile
329     print "#if defined(_KERNEL) || defined(KMEMUSER)" > privhfile
330     print "#define\t\t\t\t\t PRIV_NSET\t\t\t\t\t nset > privhfile
331     print "#define\t\t\t\t\t PRIV_SETSIZE\t\t\t\t\t setsize > privhfile
332     print "#endif\n\n#ifdef _KERNEL" > privhfile
333     print "#define\t\t\t\t\t _PRIV_CONST_IMPL\n" > privhfile
334     print "extern const char *priv_names[);" > privhfile
335     print "extern const char *priv_setnames[);" > privhfile
337     print "extern int nprivs;" > privhfile
```



```

338     print "extern int privbytes;" > privhfile
339     print "extern int maxprivbytes;" > privhfile
340     print "extern size_t privinfosize;" > privhfile
341     print "extern char *priv_str;" > privhfile
342     print "extern struct priv_set *priv_basic;" > privhfile
343     print "extern struct priv_impl_info *priv_info;" > privhfile
344     print "extern struct priv_info_names *priv_ninfo;" > privhfile

346     print "\n/* Privileges */" > privhfile
347
348     for (i = 0; i < npriv; i++)
349         print pdef[i] sprintf("%3d", i) > privhfile

351     print "\n/* Privilege sets */" > privhfile
352     for (i = 0; i < nset; i++)
353         print sdef[i] sprintf("%3d", i) > privhfile

355     print "\n#define\tMAX_PRIVILEGE\t\t\t " setsize * 32 \
356           > privhfile

358     # Special privilege categories.
359     for (s in set)
360         print "\n#define\tPRIV_" s "_ADDSET(set)" set[s] \
360         print "\n#define\tPRIV_" s "_ASSERT(set)" set[s] \
361           > privhfile

363     print "\n#endif /* _KERNEL */" > privhfile
364     print "\n#ifdef __cplusplus\n\n#endif" > privhfile
365     print "\n#endif /* _SYS_PRIV_CONST_H */" > privhfile
366     close(privhfile)
367 }

369 if (pubhfile) {
370     cast="(const char *)"
371     print "#ifndef _SYS_PRIV_NAMES_H" > pubhfile
372     print "#define\t_SYS_PRIV_NAMES_H\n" > pubhfile

374     print pragma "\n" > pubhfile
375     print "#ifdef __cplusplus\nextern \"C\" {\n#endif\n" > pubhfile

377     print "#ifndef __PRIV_CONST_IMPL" > pubhfile
378     print "/*\n * Privilege names\n */" > pubhfile
379     for (i = 0; i < npriv; i++) {
380         print "/*\n" privcmt[i] " */" > pubhfile
381         print pdef[i] cast "\"" privs[i] "\"\n" > pubhfile
382     }

384     print "" > pubhfile

386     print "/*\n * Privilege set names\n */" > pubhfile
387     for (i = 0; i < nset; i++) {
388         print "/*\n" setcmt[i] " */" > pubhfile
389         print sdef[i] cast "\"" sets[i] "\"\n" > pubhfile
390     }

392     print "\n#endif /* __PRIV_CONST_IMPL */" > pubhfile
393     print "\n#ifdef __cplusplus\n\n#endif" > pubhfile
394     print "\n#endif /* _SYS_PRIV_NAMES_H */" > pubhfile
395     close(pubhfile)
396 }

398 if (pnamesfile) {
399     print pnamescmt > pnamesfile
400     for (i = 0; i < npriv; i++) {
401         print privs[i] > pnamesfile
402         print privcmt[i] > pnamesfile

```

```

403     }
404 }
406 }
_____unchanged_portion_omitted_

```

new/usr/src/uts/common/os/proc.c

1

4128 Thu Jun 30 22:03:34 2016

new/usr/src/uts/common/os/proc.c

7029 want per-process exploit mitigation features (secflags)

7030 want basic address space layout randomization (aslr)

7031 noexec_user_stack should be a secflag

7032 want a means to forbid mappings around NULL.

_____unchanged_portion_omitted_____

```
163 boolean_t
164 secflag_enabled(proc_t *p, secflag_t flag)
165 {
166     return (secflag_isset(p->p_secflags.psf_effective, flag));
167 }

169 void
170 secflags_promote(proc_t *p)
171 {
172     secflags_copy(&p->p_secflags.psf_effective, &p->p_secflags.psf_inherit);
173 }
174 #endif /* ! codereview */
```

```

*****
46455 Thu Jun 30 22:03:35 2016
new/usr/src/uts/common/os/sysent.c
Code review comments from pmoonney (sundry), and igork (screwups in zonecfg refac
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */

22 /*
23  * Copyright (c) 1988, 2010, Oracle and/or its affiliates. All rights reserved.
24  * Copyright 2012 Milan Jurik. All rights reserved.
25  * Copyright (c) 2013, OmniTI Computer Consulting, Inc. All rights reserved.
26  * Copyright (c) 2015, Joyent, Inc.
27 */

29 /*      Copyright (c) 1984, 1986, 1987, 1988, 1989 AT&T */
30 /*      All Rights Reserved      */

32 #include <sys/param.h>
33 #include <sys/types.h>
34 #include <sys/system.h>
35 #include <sys/strace.h>
36 #include <sys/procfs.h>
37 #include <sys/mman.h>
38 #include <sys/int_types.h>
39 #include <c2/audit.h>
40 #include <sys/stat.h>
41 #include <sys/times.h>
42 #include <sys/statfs.h>
43 #include <sys/stropts.h>
44 #include <sys/statvfs.h>
45 #include <sys/utsname.h>
46 #include <sys/timex.h>
47 #include <sys/socket.h>
48 #include <sys/sendfile.h>

50 struct hrtsysa;
51 struct mmmaplf32a;

53 /*
54  * This table is the switch used to transfer to the appropriate
55  * routine for processing a system call. Each row contains the
56  * number of arguments expected, a switch that tells sysstrap()
57  * in trap.c whether a setjmp() is not necessary, and a pointer

```

```

58  * to the routine.
59  */

61 int      access(char *, int);
62 int      alarm(int);
63 int      auditsys(struct auditcalls *, rval_t *);
64 int64_t  brandsys(int, uintptr_t, uintptr_t, uintptr_t, uintptr_t, uintptr_t,
65                uintptr_t);
66 uintptr_t brk(caddr_t);
67 int      brk(caddr_t);
68 int      chdir(char *);
69 int      chmod(char *, int);
70 int      chown(char *, uid_t, gid_t);
71 int      chroot(char *);
72 int      cladm(int, int, void *);
73 int      close(int);
74 int      exece(const char *, const char **, const char **);
75 int      faccessat(int, char *, int, int);
76 int      fchmodat(int, char *, int, int);
77 int      fchownat(int, char *, uid_t, gid_t, int);
78 int     fcntl(int, int, uintptr_t);
79 int64_t  vfork();
80 int      forksys(int, int);
81 int      fstat(int, struct stat *);
82 int64_t  fdsync(int, int);
83 int      getgid();
84 int64_t  getpid();
85 int64_t  getuid();
86 time_t   gtime();
87 int      getloadavg(int *, int);
88 int      rusagesys(int, void *, void *, void *, void *);
89 int      getpagesizes(int, size_t *, int);
90 int      gtty(int, uintptr_t);
91 #if defined(__i386) || defined(__amd64)
92 int      hrtsys(struct hrtsysa *, rval_t *);
93 #endif /* __i386 || __amd64 */
94 int      ioctl(int, int, uintptr_t);
95 int      kill();
96 int      labelsys(int, void *, void *, void *, void *, void *);
97 int      link(char *, char *);
98 int      linkat(int, char *, int, char *, int);
99 off32_t  lseek32(int32_t, off32_t, int32_t);
100 off_t    lseek64(int, off_t, int);
101 int      lgrpsys(int, long, void *);
102 int      mmapobjsys(int, uint_t, mmapobj_result_t *, uint_t *, void *);
103 int      mknod(char *, mode_t, dev_t);
104 int      mknodat(int, char *, mode_t, dev_t);
105 int      mount(long *, rval_t *);
106 int      nice(int);
107 int      nullsys();
108 int      open(char *, int, int);
109 int      openat(int, char *, int, int);
110 int      pause();
111 long     pcsample(void *, long);
112 int      privsys(int, priv_op_t, priv_ptype_t, void *, size_t, int);
113 int      profil(unsigned short *, size_t, ulong_t, uint_t);
114 ssize_t  pread(int, void *, size_t, off_t);
115 int      psecflags(procset_t *, psecflagwhich_t, secflagdelta_t *);
116 #endif /* !codereview */
117 ssize_t  pwrite(int, void *, size_t, off_t);
118 ssize_t  read(int, void *, size_t);
119 int      rename(char *, char *);
120 int      renameat(int, char *, int, char *);
121 void     rexit(int);
122 int      semsys();

```

```

123 int      setgid(gid_t);
124 int      setpgpr(int, int, int);
125 int      setuid(uid_t);
126 uintptr_t shmsys();
127 uint64_t  sidsys(int, int, int, int);
128 int      sigprocmask(int, sigset_t *, sigset_t *);
129 int      sigsuspend(sigset_t);
130 int      sigaltstack(struct sigaltstack *, struct sigaltstack *);
131 int      sigaction(int, struct sigaction *, struct sigaction *);
132 int      sigpending(int, sigset_t *);
133 int      sigresend(int, siginfo_t *, sigset_t *);
134 int      sigtimedwait(sigset_t *, siginfo_t *, timespec_t *);
135 int      getsetcontext(int, void *);
136 int      stat(char *, struct stat *);
137 int      fstatat(int, char *, struct stat *, int);
138 int      stime(time_t);
139 int      stty(int, intp_t);
140 int      sysync();
141 int      sysacct(char *);
142 clock_t  times(struct tms *);
143 long     ulimit(int, long);
144 int      getrlimit32(int, struct rlimit32 *);
145 int      setrlimit32(int, struct rlimit32 *);
146 int      umask(int);
147 int      umount2(char *, int);
148 int      unlink(char *);
149 int      unlinkat(int, char *, int);
150 int      utimesys(int, uintptr_t, uintptr_t, uintptr_t, uintptr_t);
151 int64_t  utssys32(void *, int, int, void *);
152 int64_t  utssys64(void *, long, int, void *);
153 int      uucopy(const void *, void *, size_t);
154 ssize_t  uucopystr(const char *, char *, size_t);
155 ssize_t  write(int, void *, size_t);
156 ssize_t  readv(int, struct iovec *, int);
157 ssize_t  writev(int, struct iovec *, int);
158 ssize_t  preadv(int, struct iovec *, int, off_t, off_t);
159 ssize_t  pwritev(int, struct iovec *, int, off_t, off_t);
160 int      syslwp_park(int, uintptr_t, uintptr_t);
161 int      rmdir(char *);
162 int      mkdir(char *, int);
163 int      mkdirat(int, char *, int);
164 int      getdents32(int, void *, size_t);
165 int      statfs32(char *, struct statfs32 *, int32_t, int32_t);
166 int      fstatfs32(int32_t, struct statfs32 *, int32_t, int32_t);
167 int      sysfs(int, long, long);
168 int      getmsg(int, struct strbuf *, struct strbuf *, int *);
169 int      pollsys(pollfd_t *, nfds_t, timespec_t *, sigset_t *);
170 int      putmsg(int, struct strbuf *, struct strbuf *, int);
171 int      uadmin();
172 int      lstat(char *, struct stat *);
173 int      symlink(char *, char *);
174 int      symlinkat(char *, int, char *);
175 ssize_t  readlink(char *, char *, size_t);
176 ssize_t  readlinkat(int, char *, char *, size_t);
177 int      resolvepath(char *, char *, size_t);
178 int      setgroups(int, gid_t *);
179 int      getgroups(int, gid_t *);
180 int      fchdir(int);
181 int      fchown(int, uid_t, uid_t);
182 int      fchmod(int, int);
183 int      getcwd(char *, size_t);
184 int      statvfs(char *, struct statvfs *);
185 int      fstatvfs(int, struct statvfs *);
186 offset_t llseek32(int32_t, uint32_t, uint32_t, int);

```

```
188 #if (defined(__i386) && !defined(__amd64)) || defined(__i386_COMPAT)
```

```

189 int      sysi86(short, uintptr_t, uintptr_t, uintptr_t);
190 #endif

192 int      acl(const char *, int, int, void *);
193 int      facl(int, int, int, void *);
194 long     priocntlsys(int, procset_t *, int, caddr_t, caddr_t);
195 int      waitsys(idtype_t, id_t, siginfo_t *, int);
196 int      sigsendsys(procset_t *, int);
197 int      mincore(caddr_t, size_t, char *);
198 caddr_t  mmap64(caddr_t, size_t, int, int, int, off_t);
199 caddr_t  mmap32(caddr32_t, size32_t, int, int, int, off32_t);
200 int      mmap1f32(struct mmap1f32a *, rval_t *);
201 int      mprotect(caddr_t, size_t, int);
202 int      munmap(caddr_t, size_t);
203 int      uname(struct utsname *);
204 int      lchown(char *, uid_t, gid_t);
205 int      getpmsg(int, struct strbuf *, struct strbuf *, int *, int *);
206 int      putpmsg(int, struct strbuf *, struct strbuf *, int, int);
207 int      memcntl(caddr_t, size_t, int, caddr_t, int, int);
208 long     sysconfig(int);
209 int      adjtime(struct timeval *, struct timeval *);
210 long     systeminfo(int, char *, long);
211 int      setegid(gid_t);
212 int      seteuid(uid_t);

214 int      setreuid(uid_t, uid_t);
215 int      setregid(gid_t, gid_t);
216 int      install_utrap(utrap_entry_t type, utrap_handler_t, utrap_handler_t *);
217 #ifdef __sparc
218 int      sparc_utrap_install(utrap_entry_t type, utrap_handler_t,
219 utrap_handler_t, utrap_handler_t *, utrap_handler_t *);
220 #endif

222 int      syslwp_create(ucontext_t *, int, id_t *);
223 void     syslwp_exit();
224 int      syslwp_suspend(id_t);
225 int      syslwp_continue(id_t);
226 int      syslwp_private(int, int, uintptr_t);
227 int      lwp_detach(id_t);
228 int      lwp_info(timestruc_t *);
229 int      lwp_kill(id_t, int);
230 int      lwp_self();
231 int64_t  lwp_sigmask(int, uint_t, uint_t, uint_t, uint_t);
232 int      yield();
233 int      lwp_wait(id_t, id_t *);
234 int      lwp_mutex_timedlock(lwp_mutex_t *, timespec_t *, uintptr_t);
235 int      lwp_mutex_wakeup(lwp_mutex_t *, int);
236 int      lwp_mutex_unlock(lwp_mutex_t *);
237 int      lwp_mutex_trylock(lwp_mutex_t *, uintptr_t);
238 int      lwp_mutex_register(lwp_mutex_t *, caddr_t);
239 int      lwp_rwlock_sys(int, lwp_rwlock_t *, timespec_t *);
240 int      lwp_sema_post(lwp_sema_t *);
241 int      lwp_sema_timedwait(lwp_sema_t *, timespec_t *, int);
242 int      lwp_sema_trywait(lwp_sema_t *);
243 int      lwp_cond_wait(lwp_cond_t *, lwp_mutex_t *, timespec_t *, int);
244 int      lwp_cond_signal(lwp_cond_t *);
245 int      lwp_cond_broadcast(lwp_cond_t *);
246 caddr_t  schedctl();

248 long     pathconf(char *, int);
249 long     fpathconf(int, int);
250 int      processor_bind(idtype_t, id_t, processorid_t, processorid_t *);
251 int      processor_info(processorid_t, processor_info_t *);
252 int      p_online(processorid_t, int);

```

```
254 /*
```

```

255 *      POSIX .4 system calls *
256 */
257 int      clock_gettime(clockid_t, timespec_t *);
258 int      clock_settime(clockid_t, timespec_t *);
259 int      clock_getres(clockid_t, timespec_t *);
260 int      timer_create(clockid_t, struct sigevent *, timer_t *);
261 int      timer_delete(timer_t);
262 int      timer_settime(timer_t, int, itimerspec_t *, itimerspec_t *);
263 int      timer_gettime(timer_t, itimerspec_t *);
264 int      timer_getoverrun(timer_t);
265 int      nanosleep(timespec_t *, timespec_t *);
266 int      sigqueue(pid_t, int, void *, int, int);
267 int      signotify(int, siginfo_t *, signotify_id_t *);

269 int      getdents64(int, void *, size_t);
270 int      stat64(char *, struct stat64 *);
271 int      lstat64(char *, struct stat64 *);
272 int      fstatat64(int, char *, struct stat64 *, int);
273 int      fstat64(int, struct stat64 *);
274 int      statvfs64(char *, struct statvfs64 *);
275 int      fstatvfs64(int, struct statvfs64 *);
276 int      setrlimit64(int, struct rlimit64 *);
277 int      getrlimit64(int, struct rlimit64 *);
278 int      pread64(int, void *, size32_t, uint32_t, uint32_t);
279 int      pwrite64(int, void *, size32_t, uint32_t, uint32_t);
280 int      open64(char *, int, int);
281 int      openat64(int, char *, int, int);

283 /*
284 * NTP syscalls
285 */

287 int ntp_gettime(struct ntptimeval *);
288 int ntp_adjtime(struct timex *);

290 /*
291 *      ++++++
292 *      ++ SunOS4.1 Buyback ++
293 *      ++++++
294 *
295 *      fchroot, vhangup, gettimeofday
296 */

298 int      fchroot(int);
299 int      vhangup();
300 int      gettimeofday(struct timeval *);
301 int      getitimer(uint_t, struct itimerval *);
302 int      setitimer(uint_t, struct itimerval *, struct itimerval *);

304 int      corectl(int, uintptr_t, uintptr_t, uintptr_t);
305 int      modctl(int, uintptr_t, uintptr_t, uintptr_t, uintptr_t);
306 int64_t  loadable_syscall();
307 int64_t  indir();

309 long     tasksys(int, projid_t, uint_t, void *, size_t);
310 long     rctlsys(int, char *, void *, void *, size_t, int);

312 long     zone();

314 int      so_socket(int, int, int, char *, int);
315 int      so_socketpair(int[2]);
316 int      bind(int, struct sockaddr *, socklen_t, int);
317 int      listen(int, int, int);
318 int      accept(int, struct sockaddr *, socklen_t *, int, int);
319 int      connect(int, struct sockaddr *, socklen_t, int);
320 int      shutdown(int, int, int);

```

```

321 ssize_t  recv(int, void *, size_t, int);
322 ssize_t  recvfrom(int, void *, size_t, int, struct sockaddr *, socklen_t *);
323 ssize_t  recvmsg(int, struct mmsghdr *, int);
324 ssize_t  send(int, void *, size_t, int);
325 ssize_t  sendmsg(int, struct mmsghdr *, int);
326 ssize_t  sendto(int, void *, size_t, int, struct sockaddr *, socklen_t);
327 int      getpeername(int, struct sockaddr *, socklen_t *, int);
328 int      getsockname(int, struct sockaddr *, socklen_t *, int);
329 int      getsockopt(int, int, int, void *, socklen_t *, int);
330 int      setsockopt(int, int, int, void *, socklen_t *, int);
331 int      sockconfig(int, void *, void *, void *, void *);
332 ssize_t  sendfilev(int, int, const struct sendfilevec *, int, size_t *);
333 int      getrandom(void *, size_t, int);

335 typedef int64_t (*llfcn_t)(); /* for casting one-word returns */

337 /*
338 * Sysent initialization macros.
339 * These take the name string of the system call even though that isn't
340 * currently used in the sysent entry. This might be useful someday.
341 *
342 * Initialization macro for system calls which take their args in the C style.
343 * These system calls return the longlong_t return value and must call
344 * set_errno() to return an error. For SPARC, nargs must be at most six.
345 * For more args, use the SYSENT_AP() routine.
346 *
347 * We are able to return two distinct values to userland via the rval_t.
348 * At this time, that corresponds to one 64-bit quantity, or two 32-bit
349 * quantities. The kernel does not currently need to return two 64-bit
350 * values, or one 128 bit value(!), but we may do one day, so the calling
351 * sequence between userland and the kernel should permit it.
352 *
353 * The interpretation of rval_t is provided by the sy_flags field
354 * which is used to determine how to arrange the results in registers
355 * (or on the stack) for return userland.
356 */
357 /* returns a 64-bit quantity for both ABIs */
358 #define SYSENT_C(name, call, nargs) \
359 { (nargs), SE_64RVAL, NULL, NULL, (llfcn_t)(call) }

361 /* returns one 32-bit value for both ABIs: r_val1 */
362 #define SYSENT_CI(name, call, nargs) \
363 { (nargs), SE_32RVAL1, NULL, NULL, (llfcn_t)(call) }

365 /* returns 2 32-bit values: r_val1 & r_val2 */
366 #define SYSENT_2CI(name, call, nargs) \
367 { (nargs), SE_32RVAL1|SE_32RVAL2, NULL, NULL, (llfcn_t)(call) }

369 /*
370 * Initialization macro for system calls which take their args in the standard
371 * Unix style of a pointer to the arg structure and a pointer to the rval_t.
372 *
373 * Deprecated wherever possible (slower on some architectures, and trickier
374 * to maintain two flavours).
375 */
376 #define SYSENT_AP(name, call, nargs) \
377 { (nargs), SE_64RVAL, (call), NULL, syscall_ap }

379 /*
380 * Conditional constructors to build the tables without #ifdef clutter
381 */
382 #if defined(LP64)
383 #define IF_LP64(true, false) true
384 #else
385 #define IF_LP64(true, false) false
386 #endif

```

```

388 #if defined(__sparc)
389 #define IF_sparc(true, false) true
390 #else
391 #define IF_sparc(true, false) false
392 #endif

394 #if defined(__i386) && !defined(__amd64)
395 #define IF_i386(true, false) true
396 #else
397 #define IF_i386(true, false) false
398 #endif

400 #if defined(__i386) || defined(__amd64)
401 #define IF_x86(true, false) true
402 #else
403 #define IF_x86(true, false) false
404 #endif

406 #if (defined(__i386) && !defined(__amd64)) || defined(__i386_COMPAT)
407 #define IF_386_ABI(true, false) true
408 #else
409 #define IF_386_ABI(true, false) false
410 #endif

412 /*
413  * Define system calls that return a native 'long' quantity i.e. a 32-bit
414  * or 64-bit integer - depending on how the kernel is itself compiled
415  * e.g. read(2) returns 'ssize_t' in the kernel and in userland.
416  */
417 #define SYSENT_CL(name, call, nargs) \
418     IF_LP64(SYSENT_C(name, call, nargs), SYSENT_CI(name, call, nargs))

420 /*
421  * Initialization macro for loadable native system calls.
422  */
423 #define SYSENT_LOADABLE() \
424     { 0, SE_LOADABLE, (int (*)(*))nosys, NULL, loadable_syscall }

426 /*
427  * Initialization macro for loadable 32-bit compatibility system calls.
428  */
429 #define SYSENT_LOADABLE32() SYSENT_LOADABLE()

431 #define SYSENT_NOSYS() SYSENT_C("nosys", nosys, 0)

433 struct sysent nosys_ent = SYSENT_NOSYS();

435 /*
436  * Native sysent table.
437  */
438 struct sysent sysent[NSYSCALL] =
439 {
440     /* 0 */ IF_LP64(
441         SYSENT_NOSYS(),
442         SYSENT_C("indir", indir, 1)),
443     /* 1 */ SYSENT_CI("exit", rexit, 1),
444     /* 2 */ SYSENT_CI("psecflags", psecflags, 3),
445     /* 3 */ SYSENT_LOADABLE(), /* (was forkall) */
446     /* 4 */ SYSENT_CL("read", read, 3),
447     /* 5 */ SYSENT_CL("write", write, 3),
448     /* 6 */ SYSENT_CI("open", open, 3),
449     /* 7 */ SYSENT_CI("close", close, 1),
450     /* 8 */ SYSENT_CI("linkat", linkat, 5),
451     /* 9 */ SYSENT_LOADABLE(), /* (was creat) */
452     /* 10 */ SYSENT_CI("link", link, 2),

```

```

452     /* 10 */ SYSENT_CI("unlink", unlink, 1),
453     /* 11 */ SYSENT_CI("symlinkat", symlinkat, 3),
454     /* 12 */ SYSENT_CI("chdir", chdir, 1),
455     /* 13 */ SYSENT_CL("time", gtime, 0),
456     /* 14 */ SYSENT_CI("mknod", mknod, 3),
457     /* 15 */ SYSENT_CI("chmod", chmod, 2),
458     /* 16 */ SYSENT_CI("chown", chown, 3),
459     /* 17 */ SYSENT_CI("brk", brk, 1),
460     /* 18 */ SYSENT_CI("stat", stat, 2),
461     /* 19 */ IF_LP64(
462         SYSENT_CL("lseek", lseek64, 3),
463         SYSENT_CL("lseek", lseek32, 3)),
464     /* 20 */ SYSENT_2CI("getpid", getpid, 0),
465     /* 21 */ SYSENT_AP("mount", mount, 8),
466     /* 22 */ SYSENT_CL("readlinkat", readlinkat, 4),
467     /* 23 */ SYSENT_CI("setuid", setuid, 1),
468     /* 24 */ SYSENT_2CI("getuid", getuid, 0),
469     /* 25 */ SYSENT_CI("stime", stime, 1),
470     /* 26 */ SYSENT_CL("pcsample", pcsample, 2),
471     /* 27 */ SYSENT_CI("alarm", alarm, 1),
472     /* 28 */ SYSENT_CI("fstat", fstat, 2),
473     /* 29 */ SYSENT_CI("pause", pause, 0),
474     /* 30 */ SYSENT_LOADABLE(), /* (was utime) */
475     /* 31 */ SYSENT_CI("stty", stty, 2),
476     /* 32 */ SYSENT_CI("gtty", gtty, 2),
477     /* 33 */ SYSENT_CI("access", access, 2),
478     /* 34 */ SYSENT_CI("nice", nice, 1),
479     /* 35 */ IF_LP64(
480         SYSENT_NOSYS(),
481         SYSENT_CI("statfs", statfs32, 4)),
482     /* 36 */ SYSENT_CI("sync", syssync, 0),
483     /* 37 */ SYSENT_CI("kill", kill, 2),
484     /* 38 */ IF_LP64(
485         SYSENT_NOSYS(),
486         SYSENT_CI("fstatfs", fstatfs32, 4)),
487     /* 39 */ SYSENT_CI("setpggrp", setpggrp, 3),
488     /* 40 */ SYSENT_CI("uucopystr", uucopystr, 3),
489     /* 41 */ SYSENT_LOADABLE(), /* (was dup) */
490     /* 42 */ SYSENT_LOADABLE(), /* pipe */
491     /* 43 */ SYSENT_CL("times", times, 1),
492     /* 44 */ SYSENT_CI("profil", profil, 4),
493     /* 45 */ SYSENT_CI("faccessat", faccessat, 4),
494     /* 46 */ SYSENT_CI("setgid", setgid, 1),
495     /* 47 */ SYSENT_2CI("getgid", getgid, 0),
496     /* 48 */ SYSENT_CI("mknodat", mknodat, 4),
497     /* 49 */ SYSENT_LOADABLE(), /* msgsys */
498     /* 50 */ IF_x86(
499         SYSENT_CI("sysi86", sysi86, 4),
500         SYSENT_LOADABLE()), /* (was sys3b) */
501     /* 51 */ SYSENT_LOADABLE(), /* sysacct */
502     /* 52 */ SYSENT_LOADABLE(), /* shmsys */
503     /* 53 */ SYSENT_LOADABLE(), /* semsys */
504     /* 54 */ SYSENT_CI("ioctl", ioctl, 3),
505     /* 55 */ SYSENT_CI("uadmin", uadmin, 3),
506     /* 56 */ SYSENT_CI("fchownat", fchownat, 5),
507     /* 57 */ IF_LP64(
508         SYSENT_2CI("utssys", utssys64, 4),
509         SYSENT_2CI("utssys", utssys32, 4)),
510     /* 58 */ SYSENT_CI("fdsync", fdsync, 2),
511     /* 59 */ SYSENT_CI("exece", exece, 3),
512     /* 60 */ SYSENT_CI("umask", umask, 1),
513     /* 61 */ SYSENT_CI("chroot", chroot, 1),
514     /* 62 */ SYSENT_CI("font1", font1, 3),
515     /* 63 */ SYSENT_CI("ulimit", ulimit, 2),
516     /* 64 */ SYSENT_CI("renameat", renameat, 4),
517     /* 65 */ SYSENT_CI("unlinkat", unlinkat, 3),

```

```

518 /* 66 */ SYSENT_CI("fstatat",          fstatat,          4),
519 /* 67 */ IF_LP64(
520     SYSENT_NOSYS(),
521     SYSENT_CI("fstatat64",          fstatat64,          4)),
522 /* 68 */ SYSENT_CI("openat",          openat,          4),
523 /* 69 */ IF_LP64(
524     SYSENT_NOSYS(),
525     SYSENT_CI("openat64",          openat64,          4)),
526 /* 70 */ SYSENT_CI("tasksys",          tasksys,          5),
527 /* 71 */ SYSENT_LOADABLE(),           /* acctctl */
528 /* 72 */ SYSENT_LOADABLE(),           /* exact */
529 /* 73 */ SYSENT_CI("getpagesizes",    getpagesizes,    3),
530 /* 74 */ SYSENT_CI("rctlsys",          rctlsys,          6),
531 /* 75 */ SYSENT_2CI("sidsys",          sidsys,          4),
532 /* 76 */ SYSENT_LOADABLE(),           /* (was fsat) */
533 /* 77 */ SYSENT_CI("lwp_park",          syslwp_park,     3),
534 /* 78 */ SYSENT_CL("sendfilev",        sendfilev,       5),
535 /* 79 */ SYSENT_CI("rmdir",            rmdir,            1),
536 /* 80 */ SYSENT_CI("mkdir",            mkdir,            2),
537 /* 81 */ IF_LP64(
538     SYSENT_CI("getdents",            getdents64,      3),
539     SYSENT_CI("getdents",            getdents32,     3)),
540 /* 82 */ SYSENT_CI("privsys",          privsys,          6),
541 /* 83 */ SYSENT_CI("ucreditsys",        ucreditsys,      3),
542 /* 84 */ SYSENT_CI("sysfs",            sysfs,            3),
543 /* 85 */ SYSENT_CI("getmsg",           getmsg,           4),
544 /* 86 */ SYSENT_CI("putmsg",           putmsg,           4),
545 /* 87 */ SYSENT_LOADABLE(),           /* (was poll) */
546 /* 88 */ SYSENT_CI("lstat",            lstat,            2),
547 /* 89 */ SYSENT_CI("symlink",          symlink,          2),
548 /* 90 */ SYSENT_CL("readlink",         readlink,         3),
549 /* 91 */ SYSENT_CI("setgroups",        setgroups,        2),
550 /* 92 */ SYSENT_CI("getgroups",        getgroups,        2),
551 /* 93 */ SYSENT_CI("fchmod",           fchmod,           2),
552 /* 94 */ SYSENT_CI("fchown",           fchown,           3),
553 /* 95 */ SYSENT_CI("sigprocmask",      sigprocmask,     3),
554 /* 96 */ SYSENT_CI("sigsuspend",       sigsuspend,       1),
555 /* 97 */ SYSENT_CI("sigaltstack",      sigaltstack,     2),
556 /* 98 */ SYSENT_CI("sigaction",        sigaction,        3),
557 /* 99 */ SYSENT_CI("sigpending",       sigpending,       2),
558 /* 100 */ SYSENT_CI("getsetcontext",    getsetcontext,   2),
559 /* 101 */ SYSENT_CI("fchmodat",         fchmodat,        4),
560 /* 102 */ SYSENT_CI("mkdirat",         mkdirat,          3),
561 /* 103 */ SYSENT_CI("statvfs",          statvfs,          2),
562 /* 104 */ SYSENT_CI("fstatvfs",         fstatvfs,         2),
563 /* 105 */ SYSENT_CI("getloadavg",      getloadavg,       2),
564 /* 106 */ SYSENT_LOADABLE(),           /* nfssys */
565 /* 107 */ SYSENT_CI("waitsys",          waitsys,          4),
566 /* 108 */ SYSENT_CI("sigsendset",      sigsendsys,      2),
567 /* 109 */ IF_x86(
568     SYSENT_AP("hrtsys",              hrtsys,          5),
569     SYSENT_LOADABLE()),
570 /* 110 */ SYSENT_CI("utimesys",         utimesys,        5),
571 /* 111 */ SYSENT_CI("sigresend",        sigresend,       3),
572 /* 112 */ SYSENT_CL("prioctlsys",      prioctlsys,      5),
573 /* 113 */ SYSENT_CL("pathconf",        pathconf,        2),
574 /* 114 */ SYSENT_CI("mincore",         mincore,         3),
575 /* 115 */ IF_LP64(
576     SYSENT_CL("mmap",                 smmap64,         6),
577     SYSENT_CL("mmap",                 smmap32,         6)),
578 /* 116 */ SYSENT_CI("mprotect",         mprotect,        3),
579 /* 117 */ SYSENT_CI("munmap",           munmap,          2),
580 /* 118 */ SYSENT_CL("fpathconf",        fpathconf,       2),
581 /* 119 */ SYSENT_2CI("vfork",          vfork,           0),
582 /* 120 */ SYSENT_CI("fchdir",           fchdir,          1),
583 /* 121 */ SYSENT_CL("readv",            readv,           3),

```

```

584 /* 122 */ SYSENT_CL("writev",          writev,          3),
585 /* 123 */ SYSENT_CL("preadv",          preadv,          5),
586 /* 124 */ SYSENT_CL("pwritev",         pwritev,         5),
587 /* 125 */ SYSENT_LOADABLE(),           /* (was fxstat) */
588 /* 126 */ SYSENT_CI("getrandom",       getrandom,       3),
589 /* 127 */ SYSENT_CI("mmapobj",        mmapobjsys,     5),
590 /* 128 */ IF_LP64(
591     SYSENT_CI("setrlimit",            setrlimit64,    2),
592     SYSENT_CI("setrlimit",            setrlimit32,    2)),
593 /* 129 */ IF_LP64(
594     SYSENT_CI("getrlimit",            getrlimit64,    2),
595     SYSENT_CI("getrlimit",            getrlimit32,    2)),
596 /* 130 */ SYSENT_CI("lchown",          lchown,          3),
597 /* 131 */ SYSENT_CI("memcntl",         memcntl,         6),
598 /* 132 */ SYSENT_CI("getpmsg",         getpmsg,         5),
599 /* 133 */ SYSENT_CI("putpmsg",         putpmsg,         5),
600 /* 134 */ SYSENT_CI("rename",          rename,          2),
601 /* 135 */ SYSENT_CI("uname",           uname,           1),
602 /* 136 */ SYSENT_CI("setegid",         setegid,         1),
603 /* 137 */ SYSENT_CL("sysconfig",       sysconfig,       1),
604 /* 138 */ SYSENT_CI("adjtime",         adjtime,         2),
605 /* 139 */ SYSENT_CL("systeminfo",      systeminfo,      3),
606 /* 140 */ SYSENT_LOADABLE(),           /* sharefs */
607 /* 141 */ SYSENT_CI("seteuid",         seteuid,         1),
608 /* 142 */ SYSENT_2CI("forksys",        forksys,         2),
609 /* 143 */ SYSENT_LOADABLE(),           /* (was fork1) */
610 /* 144 */ SYSENT_CI("sigtimedwait",    sigtimedwait,   3),
611 /* 145 */ SYSENT_CI("lwp_info",        lwp_info,        1),
612 /* 146 */ SYSENT_CI("yield",           yield,           0),
613 /* 147 */ SYSENT_LOADABLE(),           /* (was lwp_sema_wait) */
614 /* 148 */ SYSENT_CI("lwp_sema_post",    lwp_sema_post,  1),
615 /* 149 */ SYSENT_CI("lwp_sema_trywait", lwp_sema_trywait, 1),
616 /* 150 */ SYSENT_CI("lwp_detach",       lwp_detach,      1),
617 /* 151 */ SYSENT_CI("corectl",         corectl,         4),
618 /* 152 */ SYSENT_CI("modctl",         modctl,          6),
619 /* 153 */ SYSENT_CI("fchroot",          fchroot,         1),
620 /* 154 */ SYSENT_LOADABLE(),           /* (was utimes) */
621 /* 155 */ SYSENT_CI("vhangup",          vhangup,         0),
622 /* 156 */ SYSENT_CI("gettimeofday",     gettimeofday,    1),
623 /* 157 */ SYSENT_CI("getitimer",       getitimer,       2),
624 /* 158 */ SYSENT_CI("setitimer",       setitimer,       3),
625 /* 159 */ SYSENT_CI("lwp_create",       syslwp_create,   3),
626 /* 160 */ SYSENT_CI("lwp_exit",         (int (*)())syslwp_exit, 0),
627 /* 161 */ SYSENT_CI("lwp_suspend",      syslwp_suspend, 1),
628 /* 162 */ SYSENT_CI("lwp_continue",     syslwp_continue, 1),
629 /* 163 */ SYSENT_CI("lwp_kill",         lwp_kill,        2),
630 /* 164 */ SYSENT_CI("lwp_self",         lwp_self,        0),
631 /* 165 */ SYSENT_2CI("lwp_sigmask",     lwp_sigmask,    5),
632 /* 166 */ IF_x86(
633     SYSENT_CI("lwp_private",           syslwp_private, 3),
634     SYSENT_NOSYS()),
635 /* 167 */ SYSENT_CI("lwp_wait",         lwp_wait,        2),
636 /* 168 */ SYSENT_CI("lwp_mutex_wakeup", lwp_mutex_wakeup, 2),
637 /* 169 */ SYSENT_LOADABLE(),           /* (was lwp_mutex_lock) */
638 /* 170 */ SYSENT_CI("lwp_cond_wait",     lwp_cond_wait,   4),
639 /* 171 */ SYSENT_CI("lwp_cond_signal",   lwp_cond_signal, 1),
640 /* 172 */ SYSENT_CI("lwp_cond_broadcast", lwp_cond_broadcast, 1),
641 /* 173 */ SYSENT_CL("pread",            pread,           4),
642 /* 174 */ SYSENT_CL("pwrite",           pwrite,          4),
643 /*
644 * The 64-bit C library maps llseek() to lseek(), so this
645 * is needed as a native syscall only on the 32-bit kernel.
646 */
647 /* 175 */ IF_LP64(
648     SYSENT_NOSYS(),
649     SYSENT_C("llseek",                llseek32,        4)),

```

```

650 /* 176 */ SYSENT_LOADABLE(), /* inst_sync */
651 /* 177 */ SYSENT_CI("brandsys", brandsys, 6),
652 /* 178 */ SYSENT_LOADABLE(), /* kaio */
653 /* 179 */ SYSENT_LOADABLE(), /* cpc */
654 /* 180 */ SYSENT_CI("lgrpsys", lgrpsys, 3),
655 /* 181 */ SYSENT_CI("rusagesys", rusagesys, 5),
656 /* 182 */ SYSENT_LOADABLE(), /* portfs */
657 /* 183 */ SYSENT_CI("pollsys", pollsys, 4),
658 /* 184 */ SYSENT_CI("labelsys", labelsys, 5),
659 /* 185 */ SYSENT_CI("acl", acl, 4),
660 /* 186 */ SYSENT_AP("auditsys", auditsys, 6),
661 /* 187 */ SYSENT_CI("processor_bind", processor_bind, 4),
662 /* 188 */ SYSENT_CI("processor_info", processor_info, 2),
663 /* 189 */ SYSENT_CI("p_online", p_online, 2),
664 /* 190 */ SYSENT_CI("sigqueue", sigqueue, 5),
665 /* 191 */ SYSENT_CI("clock_gettime", clock_gettime, 2),
666 /* 192 */ SYSENT_CI("clock_settime", clock_settime, 2),
667 /* 193 */ SYSENT_CI("clock_getres", clock_getres, 2),
668 /* 194 */ SYSENT_CI("timer_create", timer_create, 3),
669 /* 195 */ SYSENT_CI("timer_delete", timer_delete, 1),
670 /* 196 */ SYSENT_CI("timer_settime", timer_settime, 4),
671 /* 197 */ SYSENT_CI("timer_gettime", timer_gettime, 2),
672 /* 198 */ SYSENT_CI("timer_getoverrun", timer_getoverrun, 1),
673 /* 199 */ SYSENT_CI("nanosleep", nanosleep, 2),
674 /* 200 */ SYSENT_CI("facl", facl, 4),
675 /* 201 */ SYSENT_LOADABLE(), /* door */
676 /* 202 */ SYSENT_CI("setreuid", setreuid, 2),
677 /* 203 */ SYSENT_CI("setregid", setregid, 2),
678 /* 204 */ SYSENT_CI("install_utrap", install_utrap, 3),
679 /* 205 */ SYSENT_CI("signotify", signotify, 3),
680 /* 206 */ SYSENT_CL("schedctl", schedctl, 0),
681 /* 207 */ SYSENT_LOADABLE(), /* pset */
682 /* 208 */ IF_LP64(
683     SYSENT_CI("sparc_utrap_install", sparc_utrap_install, 5),
684     SYSENT_NOSYS()),
685 /* 209 */ SYSENT_CI("resolvepath", resolvepath, 3),
686 /* 210 */ SYSENT_CI("lwp_mutex_timedlock", lwp_mutex_timedlock, 3),
687 /* 211 */ SYSENT_CI("lwp_sema_timedwait", lwp_sema_timedwait, 3),
688 /* 212 */ SYSENT_CI("lwp_rwlock_sys", lwp_rwlock_sys, 3),
689 /*
690 * Syscalls 213-225: 32-bit system call support for large files.
691 *
692 * (The 64-bit C library transparently maps these system calls
693 * back to their native versions, so almost all of them are only
694 * needed as native syscalls on the 32-bit kernel).
695 */
696 /* 213 */ IF_LP64(
697     SYSENT_NOSYS(),
698     SYSENT_CI("getdents64", getdents64, 3)),
699 /* 214 */ IF_LP64(
700     SYSENT_NOSYS(),
701     SYSENT_AP("smaplf32", smmaplf32, 7)),
702 /* 215 */ IF_LP64(
703     SYSENT_NOSYS(),
704     SYSENT_CI("stat64", stat64, 2)),
705 /* 216 */ IF_LP64(
706     SYSENT_NOSYS(),
707     SYSENT_CI("lstat64", lstat64, 2)),
708 /* 217 */ IF_LP64(
709     SYSENT_NOSYS(),
710     SYSENT_CI("fstat64", fstat64, 2)),
711 /* 218 */ IF_LP64(
712     SYSENT_NOSYS(),
713     SYSENT_CI("statvfs64", statvfs64, 2)),
714 /* 219 */ IF_LP64(
715     SYSENT_NOSYS(),

```

```

716     SYSENT_CI("fstatvfs64", fstatvfs64, 2)),
717 /* 220 */ IF_LP64(
718     SYSENT_NOSYS(),
719     SYSENT_CI("setrlimit64", setrlimit64, 2)),
720 /* 221 */ IF_LP64(
721     SYSENT_NOSYS(),
722     SYSENT_CI("getrlimit64", getrlimit64, 2)),
723 /* 222 */ IF_LP64(
724     SYSENT_NOSYS(),
725     SYSENT_CI("pread64", pread64, 5)),
726 /* 223 */ IF_LP64(
727     SYSENT_NOSYS(),
728     SYSENT_CI("pwrite64", pwrite64, 5)),
729 /* 224 */ SYSENT_LOADABLE(), /* (was creat64) */
730 /* 225 */ IF_LP64(
731     SYSENT_NOSYS(),
732     SYSENT_CI("open64", open64, 3)),
733 /* 226 */ SYSENT_LOADABLE(), /* rpcsys */
734 /* 227 */ SYSENT_CL("zone", zone, 5),
735 /* 228 */ SYSENT_LOADABLE(), /* autofssys */
736 /* 229 */ SYSENT_CI("getcwd", getcwd, 2),
737 /* 230 */ SYSENT_CI("so_socket", so_socket, 5),
738 /* 231 */ SYSENT_CI("so_socketpair", so_socketpair, 1),
739 /* 232 */ SYSENT_CI("bind", bind, 4),
740 /* 233 */ SYSENT_CI("listen", listen, 3),
741 /* 234 */ SYSENT_CI("accept", accept, 5),
742 /* 235 */ SYSENT_CI("connect", connect, 4),
743 /* 236 */ SYSENT_CI("shutdown", shutdown, 3),
744 /* 237 */ SYSENT_CL("recv", recv, 4),
745 /* 238 */ SYSENT_CL("recvfrom", recvfrom, 6),
746 /* 239 */ SYSENT_CL("recvmsg", recvmsg, 3),
747 /* 240 */ SYSENT_CL("send", send, 4),
748 /* 241 */ SYSENT_CL("sendmsg", sendmsg, 3),
749 /* 242 */ SYSENT_CL("sendto", sendto, 6),
750 /* 243 */ SYSENT_CI("getpeername", getpeername, 4),
751 /* 244 */ SYSENT_CI("getsockname", getsockname, 4),
752 /* 245 */ SYSENT_CI("getsockopt", getsockopt, 6),
753 /* 246 */ SYSENT_CI("setsockopt", setsockopt, 6),
754 /* 247 */ SYSENT_CI("sockconfig", sockconfig, 5),
755 /* 248 */ SYSENT_CI("ntp_gettime", ntp_gettime, 1),
756 /* 249 */ SYSENT_CI("ntp_adjtime", ntp_adjtime, 1),
757 /* 250 */ SYSENT_CI("lwp_mutex_unlock", lwp_mutex_unlock, 1),
758 /* 251 */ SYSENT_CI("lwp_mutex_trylock", lwp_mutex_trylock, 2),
759 /* 252 */ SYSENT_CI("lwp_mutex_register", lwp_mutex_register, 2),
760 /* 253 */ SYSENT_CI("cladm", cladm, 3),
761 /* 254 */ SYSENT_CI("uucopy", uucopy, 3),
762 /* 255 */ SYSENT_CI("umount2", umount2, 2)
763 };
764
765 #ifdef _SYSCALL32_IMPL
766
767 extern int ulimit32(int, int);
768 extern ssize_t read32(int32_t, caddr32_t, size32_t);
769 extern ssize_t write32(int32_t, caddr32_t, size32_t);
770 extern ssize_t pread32(int32_t, caddr32_t, size32_t, off32_t);
771 extern ssize_t pwrite32(int32_t, caddr32_t, size32_t, off32_t);
772 extern ssize_t readv32(int32_t, caddr32_t, int32_t);
773 extern ssize_t writev32(int32_t, caddr32_t, int32_t);
774 extern ssize_t readlink32(caddr32_t, caddr32_t, size32_t);
775 extern ssize_t readlinkat32(int, caddr32_t, caddr32_t, size32_t);
776 extern int open32(char *, int, int);
777 extern int openat32(int, char *, int, int);
778 extern int stat32(char *, struct stat32 *);
779 extern int fstatat32(int, char *, struct stat32 *, int);
780 extern int lstat32(char *, struct stat32 *);

```



```

782 extern int fstat32(int, struct stat32 *);
783 extern int fstatat64_32(int, char *, struct stat64_32 *, int);
784 extern int stat64_32(char *, struct stat64_32 *);
785 extern int lstat64_32(char *, struct stat64_32 *);
786 extern int fstat64_32(int, struct stat64_32 *);
787 extern int getmsg32(int, struct strbuf32 *, struct strbuf32 *, int32_t *);
788 extern int putmsg32(int, struct strbuf32 *, struct strbuf32 *, int32_t *);
789 extern int getpmsg32(int, struct strbuf32 *, struct strbuf32 *, int32_t *,
790     int32_t *);
791 extern int putpmsg32(int, struct strbuf32 *, struct strbuf32 *, int32_t,
792     int32_t);
793 extern int getsetcontext32(int, void *);
794 extern int statvfs32(char *, struct statvfs32 *);
795 extern int fstatvfs32(int, struct statvfs32 *);
796 extern int statvfs64_32(char *, struct statvfs64_32 *);
797 extern int fstatvfs64_32(int, struct statvfs64_32 *);
798 extern int sigaction32(int, struct sigaction32 *, struct sigaction32 *);
799 extern clock32_t times32(struct tms32 *);
800 extern int stime32(time32_t);
801 extern int getpagesizes32(int, size32_t *, int);
802 extern int sigaltstack32(struct sigaltstack32 *, struct sigaltstack32 *);
803 extern int sigqueue32(pid_t, int, caddr32_t, int, int);
804 extern off_t llseek32(int32_t, uint32_t, uint32_t, int);
805 extern int waitsys32(idtype_t, id_t, siginfo_t *, int);

807 extern ssize_t recv32(int32_t, caddr32_t, size32_t, int32_t);
808 extern ssize_t recvfrom32(int32_t, caddr32_t, size32_t, int32_t, caddr32_t,
809     caddr32_t);
810 extern ssize_t send32(int32_t, caddr32_t, size32_t, int32_t);
811 extern ssize_t sendto32(int32_t, caddr32_t, size32_t, int32_t, caddr32_t,
812     socklen_t);

814 extern int privsys32(int, priv_op_t, priv_ptype_t, caddr32_t, size32_t, int);
815 extern int ucredsys32(int, int, caddr32_t);

817 /*
818  * sysent table for ILP32 processes running on
819  * a LP64 kernel.
820  */
821 struct sysent sysent32[NSYSCALL] =
822 {
823     /* 0 */ SYSENT_C("indir",          indir,          1),
824     /* 1 */ SYSENT_CI("exit",          (int (*)())rexit, 1),
825     /* 2 */ SYSENT_CI("psecflags",     psecflags,      3),
826     /* 2 */ SYSENT_LOADABLE32(),      /* (was forkall) */
827     /* 3 */ SYSENT_CI("read",          read32,         3),
828     /* 4 */ SYSENT_CI("write",         write32,        3),
829     /* 5 */ SYSENT_CI("open",          open32,         3),
830     /* 6 */ SYSENT_CI("close",         close,          1),
831     /* 7 */ SYSENT_CI("linkat",        linkat,         5),
832     /* 8 */ SYSENT_LOADABLE32(),      /* (was creat32) */
833     /* 9 */ SYSENT_CI("link",          link,           2),
834     /* 10 */ SYSENT_CI("unlink",        unlink,         1),
835     /* 11 */ SYSENT_CI("symlinkat",    symlinkat,     3),
836     /* 12 */ SYSENT_CI("chdir",        chdir,         1),
837     /* 13 */ SYSENT_CI("time",         gtime,         0),
838     /* 14 */ SYSENT_CI("mknod",        mknod,         3),
839     /* 15 */ SYSENT_CI("chmod",        chmod,         2),
840     /* 16 */ SYSENT_CI("chown",        chown,         3),
841     /* 17 */ SYSENT_CI("brk",          brk,           1),
842     /* 18 */ SYSENT_CI("stat",         stat32,        2),
843     /* 19 */ SYSENT_CI("lseek",        lseek32,       3),
844     /* 20 */ SYSENT_2CI("getpid",      getpid,         0),
845     /* 21 */ SYSENT_AP("mount",        mount,         8),
846     /* 22 */ SYSENT_CI("readlinkat",   readlinkat32,  4),
847     /* 23 */ SYSENT_CI("setuid",      setuid,        1),

```

```

847     /* 24 */ SYSENT_2CI("getuid",      getuid,        0),
848     /* 25 */ SYSENT_CI("stime",        stime32,       1),
849     /* 26 */ SYSENT_CI("pcsample",     pcsample,      2),
850     /* 27 */ SYSENT_CI("alarm",        alarm,         1),
851     /* 28 */ SYSENT_CI("fstat",        fstat32,       2),
852     /* 29 */ SYSENT_CI("pause",        pause,         0),
853     /* 30 */ SYSENT_LOADABLE32(),      /* (was utime) */
854     /* 31 */ SYSENT_CI("stty",         stty,          2),
855     /* 32 */ SYSENT_CI("gtty",         gtty,          2),
856     /* 33 */ SYSENT_CI("access",       access,        2),
857     /* 34 */ SYSENT_CI("nice",         nice,          1),
858     /* 35 */ SYSENT_CI("statfs",       statfs32,     4),
859     /* 36 */ SYSENT_CI("sync",         syssync,      0),
860     /* 37 */ SYSENT_CI("kill",         kill,          2),
861     /* 38 */ SYSENT_CI("fstatfs",      fstatfs32,    4),
862     /* 39 */ SYSENT_CI("setpggrp",     setpggrp,     3),
863     /* 40 */ SYSENT_CI("uucopystr",    uucopystr,    3),
864     /* 41 */ SYSENT_LOADABLE32(),      /* (was dup) */
865     /* 42 */ SYSENT_LOADABLE32(),      /* pipe */
866     /* 43 */ SYSENT_CI("times",        times32,      1),
867     /* 44 */ SYSENT_CI("profil",       profil,       4),
868     /* 45 */ SYSENT_CI("faccessat",    faccessat,    4),
869     /* 46 */ SYSENT_CI("setgid",       setgid,       1),
870     /* 47 */ SYSENT_2CI("getgid",      getgid,       0),
871     /* 48 */ SYSENT_CI("mknodat",      mknodat,     4),
872     /* 49 */ SYSENT_LOADABLE32(),      /* msgsys */
873     /* 50 */ IF_386_ABI(
874         SYSENT_CI("sysi86",           sysi86,       4),
875         SYSENT_LOADABLE32()),          /* (was sys3b) */
876     /* 51 */ SYSENT_LOADABLE32(),      /* sysacct */
877     /* 52 */ SYSENT_LOADABLE32(),      /* shmsys */
878     /* 53 */ SYSENT_LOADABLE32(),      /* semsys */
879     /* 54 */ SYSENT_CI("ioctl",        ioctl,        3),
880     /* 55 */ SYSENT_CI("uadmin",       uadmin,       3),
881     /* 56 */ SYSENT_CI("fchownat",     fchownat,    5),
882     /* 57 */ SYSENT_2CI("utssys",      utssys32,    4),
883     /* 58 */ SYSENT_CI("fdsync",       fdsync,       2),
884     /* 59 */ SYSENT_CI("exece",        exece,        3),
885     /* 60 */ SYSENT_CI("umask",        umask,        1),
886     /* 61 */ SYSENT_CI("chroot",       chroot,       1),
887     /* 62 */ SYSENT_CI("fcntl",        fcntl,        3),
888     /* 63 */ SYSENT_CI("ulimit",       ulimit32,    2),
889     /* 64 */ SYSENT_CI("renameat",     renameat,    4),
890     /* 65 */ SYSENT_CI("unlinkat",     unlinkat,    3),
891     /* 66 */ SYSENT_CI("fstatat",      fstatat32,   4),
892     /* 67 */ SYSENT_CI("fstatat64",    fstatat64_32, 4),
893     /* 68 */ SYSENT_CI("openat",       openat32,    4),
894     /* 69 */ SYSENT_CI("openat64",     openat64,    4),
895     /* 70 */ SYSENT_CI("tasksys",      tasksys,     5),
896     /* 71 */ SYSENT_LOADABLE32(),      /* acctctl */
897     /* 72 */ SYSENT_LOADABLE32(),      /* exact */
898     /* 73 */ SYSENT_CI("getpagesizes", getpagesizes32, 3),
899     /* 74 */ SYSENT_CI("rctlsys",      rctlsys,     6),
900     /* 75 */ SYSENT_2CI("sidsys",      sidsys,      4),
901     /* 76 */ SYSENT_LOADABLE32(),      /* (was fsat) */
902     /* 77 */ SYSENT_CI("lwp_park",     syslwp_park, 3),
903     /* 78 */ SYSENT_CI("sendfilev",    sendfilev,   5),
904     /* 79 */ SYSENT_CI("rmdir",        rmdir,       1),
905     /* 80 */ SYSENT_CI("mkdir",        mkdir,       2),
906     /* 81 */ SYSENT_CI("getdents",     getdents32,  3),
907     /* 82 */ SYSENT_CI("privsys",      privsys32,   6),
908     /* 83 */ SYSENT_CI("ucredsys",     ucredsys32,  3),
909     /* 84 */ SYSENT_CI("sysfs",        sysfs,       3),
910     /* 85 */ SYSENT_CI("getmsg",       getmsg32,    4),
911     /* 86 */ SYSENT_CI("putmsg",       putmsg32,    4),
912     /* 87 */ SYSENT_LOADABLE32(),      /* (was poll) */

```

```

913 /* 88 */ SYSENT_CI("lstat", lstat32, 2),
914 /* 89 */ SYSENT_CI("symlink", symlink, 2),
915 /* 90 */ SYSENT_CI("readlink", readlink32, 3),
916 /* 91 */ SYSENT_CI("setgroups", setgroups, 2),
917 /* 92 */ SYSENT_CI("getgroups", getgroups, 2),
918 /* 93 */ SYSENT_CI("fchmod", fchmod, 2),
919 /* 94 */ SYSENT_CI("fchown", fchown, 3),
920 /* 95 */ SYSENT_CI("sigprocmask", sigprocmask, 3),
921 /* 96 */ SYSENT_CI("sigsuspend", sigsuspend, 1),
922 /* 97 */ SYSENT_CI("sigaltstack", sigaltstack32, 2),
923 /* 98 */ SYSENT_CI("sigaction", sigaction32, 3),
924 /* 99 */ SYSENT_CI("sigpending", sigpending, 2),
925 /* 100 */ SYSENT_CI("getsetcontext", getsetcontext32, 2),
926 /* 101 */ SYSENT_CI("fchmodat", fchmodat, 4),
927 /* 102 */ SYSENT_CI("mkdirat", mkdirat, 3),
928 /* 103 */ SYSENT_CI("statvfs", statvfs32, 2),
929 /* 104 */ SYSENT_CI("fstatvfs", fstatvfs32, 2),
930 /* 105 */ SYSENT_CI("getloadavg", getloadavg, 2),
931 /* 106 */ SYSENT_LOADABLE32(), /* nfssys */
932 /* 107 */ SYSENT_CI("waitsys", waitsys32, 4),
933 /* 108 */ SYSENT_CI("sigsendset", sigsendsys, 2),
934 /* 109 */ IF_x86(
935     SYSENT_AP("hrtsys", hrtsys, 5),
936     SYSENT_LOADABLE32()),
937 /* 110 */ SYSENT_CI("utimesys", utimesys, 5),
938 /* 111 */ SYSENT_CI("sigresend", sigresend, 3),
939 /* 112 */ SYSENT_CI("prioctlsys", prioctlsys, 5),
940 /* 113 */ SYSENT_CI("pathconf", pathconf, 2),
941 /* 114 */ SYSENT_CI("mincore", mincore, 3),
942 /* 115 */ SYSENT_CI("mmap", mmap32, 6),
943 /* 116 */ SYSENT_CI("mprotect", mprotect, 3),
944 /* 117 */ SYSENT_CI("munmap", munmap, 2),
945 /* 118 */ SYSENT_CI("fpathconf", fpathconf, 2),
946 /* 119 */ SYSENT_2CI("vfork", vfork, 0),
947 /* 120 */ SYSENT_CI("fchdir", fchdir, 1),
948 /* 121 */ SYSENT_CI("readv", readv32, 3),
949 /* 122 */ SYSENT_CI("writev", writev32, 3),
950 /* 123 */ SYSENT_CI("preadv", preadv, 5),
951 /* 124 */ SYSENT_CI("pwritev", pwritev, 5),
952 /* 125 */ SYSENT_LOADABLE32(), /* was fxstat32 */
953 /* 126 */ SYSENT_CI("getrandom", getrandom, 3),
954 /* 127 */ SYSENT_CI("mmapobj", mmapobjsys, 5),
955 /* 128 */ SYSENT_CI("setrlimit", setrlimit32, 2),
956 /* 129 */ SYSENT_CI("getrlimit", getrlimit32, 2),
957 /* 130 */ SYSENT_CI("lchown", lchown, 3),
958 /* 131 */ SYSENT_CI("memcntl", memcntl, 6),
959 /* 132 */ SYSENT_CI("getpmsg", getpmsg32, 5),
960 /* 133 */ SYSENT_CI("putpmsg", putpmsg32, 5),
961 /* 134 */ SYSENT_CI("rename", rename, 2),
962 /* 135 */ SYSENT_CI("uname", uname, 1),
963 /* 136 */ SYSENT_CI("setegid", setegid, 1),
964 /* 137 */ SYSENT_CI("sysconf", sysconf, 1),
965 /* 138 */ SYSENT_CI("adjtime", adjtime, 2),
966 /* 139 */ SYSENT_CI("systeminfo", systeminfo, 3),
967 /* 140 */ SYSENT_LOADABLE32(), /* sharefs */
968 /* 141 */ SYSENT_CI("seteuid", seteuid, 1),
969 /* 142 */ SYSENT_2CI("forksys", forksys, 2),
970 /* 143 */ SYSENT_LOADABLE32(), /* (was fork1) */
971 /* 144 */ SYSENT_CI("sigtimedwait", sigtimedwait, 3),
972 /* 145 */ SYSENT_CI("lwp_info", lwp_info, 1),
973 /* 146 */ SYSENT_CI("yield", yield, 0),
974 /* 147 */ SYSENT_LOADABLE32(), /* (was lwp_sema_wait) */
975 /* 148 */ SYSENT_CI("lwp_sema_post", lwp_sema_post, 1),
976 /* 149 */ SYSENT_CI("lwp_sema_trywait", lwp_sema_trywait, 1),
977 /* 150 */ SYSENT_CI("lwp_detach", lwp_detach, 1),
978 /* 151 */ SYSENT_CI("corectl", corectl, 4),

```

```

979 /* 152 */ SYSENT_CI("modctl", modctl, 6),
980 /* 153 */ SYSENT_CI("fchroot", fchroot, 1),
981 /* 154 */ SYSENT_LOADABLE32(), /* (was utimes) */
982 /* 155 */ SYSENT_CI("vhangup", vhangup, 0),
983 /* 156 */ SYSENT_CI("gettimeofday", gettimeofday, 1),
984 /* 157 */ SYSENT_CI("getitimer", getitimer, 2),
985 /* 158 */ SYSENT_CI("setitimer", setitimer, 3),
986 /* 159 */ SYSENT_CI("lwp_create", syslwp_create, 3),
987 /* 160 */ SYSENT_CI("lwp_exit", (int (*)())syslwp_exit, 0),
988 /* 161 */ SYSENT_CI("lwp_suspend", syslwp_suspend, 1),
989 /* 162 */ SYSENT_CI("lwp_continue", syslwp_continue, 1),
990 /* 163 */ SYSENT_CI("lwp_kill", lwp_kill, 2),
991 /* 164 */ SYSENT_CI("lwp_self", lwp_self, 0),
992 /* 165 */ SYSENT_2CI("lwp_sigmask", lwp_sigmask, 5),
993 /* 166 */ IF_x86(
994     SYSENT_CI("lwp_private", syslwp_private, 3),
995     SYSENT_NOSYS()),
996 /* 167 */ SYSENT_CI("lwp_wait", lwp_wait, 2),
997 /* 168 */ SYSENT_CI("lwp_mutex_wakeup", lwp_mutex_wakeup, 2),
998 /* 169 */ SYSENT_LOADABLE32(), /* (was lwp_mutex_lock) */
999 /* 170 */ SYSENT_CI("lwp_cond_wait", lwp_cond_wait, 4),
1000 /* 171 */ SYSENT_CI("lwp_cond_signal", lwp_cond_signal, 1),
1001 /* 172 */ SYSENT_CI("lwp_cond_broadcast", lwp_cond_broadcast, 1),
1002 /* 173 */ SYSENT_CI("pread", pread32, 4),
1003 /* 174 */ SYSENT_CI("pwrite", pwrite32, 4),
1004 /* 175 */ SYSENT_CI("llseek", llseek32, 4),
1005 /* 176 */ SYSENT_LOADABLE32(), /* inst_sync */
1006 /* 177 */ SYSENT_CI("brandsys", brandsys, 6),
1007 /* 178 */ SYSENT_LOADABLE32(), /* kaio */
1008 /* 179 */ SYSENT_LOADABLE32(), /* cpc */
1009 /* 180 */ SYSENT_CI("lgrpsys", lgrpsys, 3),
1010 /* 181 */ SYSENT_CI("rusagesys", rusagesys, 5),
1011 /* 182 */ SYSENT_LOADABLE32(), /* portfs */
1012 /* 183 */ SYSENT_CI("pollsys", pollsys, 4),
1013 /* 184 */ SYSENT_CI("labelsys", labelsys, 5),
1014 /* 185 */ SYSENT_CI("acl", acl, 4),
1015 /* 186 */ SYSENT_AP("auditsys", auditsys, 6),
1016 /* 187 */ SYSENT_CI("processor_bind", processor_bind, 4),
1017 /* 188 */ SYSENT_CI("processor_info", processor_info, 2),
1018 /* 189 */ SYSENT_CI("p_online", p_online, 2),
1019 /* 190 */ SYSENT_CI("sigqueue", sigqueue32, 5),
1020 /* 191 */ SYSENT_CI("clock_gettime", clock_gettime, 2),
1021 /* 192 */ SYSENT_CI("clock_settime", clock_settime, 2),
1022 /* 193 */ SYSENT_CI("clock_getres", clock_getres, 2),
1023 /* 194 */ SYSENT_CI("timer_create", timer_create, 3),
1024 /* 195 */ SYSENT_CI("timer_delete", timer_delete, 1),
1025 /* 196 */ SYSENT_CI("timer_settime", timer_settime, 4),
1026 /* 197 */ SYSENT_CI("timer_gettime", timer_gettime, 2),
1027 /* 198 */ SYSENT_CI("timer_getoverrun", timer_getoverrun, 1),
1028 /* 199 */ SYSENT_CI("nanosleep", nanosleep, 2),
1029 /* 200 */ SYSENT_CI("facl", facl, 4),
1030 /* 201 */ SYSENT_LOADABLE32(), /* door */
1031 /* 202 */ SYSENT_CI("setreuid", setreuid, 2),
1032 /* 203 */ SYSENT_CI("setregid", setregid, 2),
1033 /* 204 */ SYSENT_CI("install_ustrap", install_ustrap, 3),
1034 /* 205 */ SYSENT_CI("signotify", signotify, 3),
1035 /* 206 */ SYSENT_CI("schedctl", schedctl, 0),
1036 /* 207 */ SYSENT_LOADABLE32(), /* pset */
1037 /* 208 */ SYSENT_LOADABLE32(),
1038 /* 209 */ SYSENT_CI("resolvepath", resolvepath, 3),
1039 /* 210 */ SYSENT_CI("lwp_mutex_timedlock", lwp_mutex_timedlock, 3),
1040 /* 211 */ SYSENT_CI("lwp_sema_timedwait", lwp_sema_timedwait, 3),
1041 /* 212 */ SYSENT_CI("lwp_rwlock_sys", lwp_rwlock_sys, 3),
1042 /*
1043 * Syscalls 213-225: 32-bit system call support for large files.
1044 */

```

```

1045 /* 213 */ SYSENT_CI("getdents64",      getdents64,      3),
1046 /* 214 */ SYSENT_AP("smaplf32",        smmaplf32,       7),
1047 /* 215 */ SYSENT_CI("stat64",          stat64_32,       2),
1048 /* 216 */ SYSENT_CI("lstat64",         lstat64_32,      2),
1049 /* 217 */ SYSENT_CI("fstat64",         fstat64_32,      2),
1050 /* 218 */ SYSENT_CI("statvfs64",       statvfs64_32,    2),
1051 /* 219 */ SYSENT_CI("fstatvfs64",     fstatvfs64_32,   2),
1052 /* 220 */ SYSENT_CI("setrlimit64",     setrlimit64,     2),
1053 /* 221 */ SYSENT_CI("getrlimit64",     getrlimit64,     2),
1054 /* 222 */ SYSENT_CI("pread64",         pread64,         5),
1055 /* 223 */ SYSENT_CI("pwrite64",        pwrite64,        5),
1056 /* 224 */ SYSENT_LOADABLE32(),         /* (was creat64) */
1057 /* 225 */ SYSENT_CI("open64",          open64,          3),
1058 /* 226 */ SYSENT_LOADABLE32(),         /* rpcsys */
1059 /* 227 */ SYSENT_CI("zone",            zone,            6),
1060 /* 228 */ SYSENT_LOADABLE32(),         /* autofs */
1061 /* 229 */ SYSENT_CI("getcwd",          getcwd,          2),
1062 /* 230 */ SYSENT_CI("so_socket",       so_socket,       5),
1063 /* 231 */ SYSENT_CI("so_socketpair",   so_socketpair,  1),
1064 /* 232 */ SYSENT_CI("bind",            bind,            4),
1065 /* 233 */ SYSENT_CI("listen",          listen,          3),
1066 /* 234 */ SYSENT_CI("accept",          accept,          5),
1067 /* 235 */ SYSENT_CI("connect",         connect,         4),
1068 /* 236 */ SYSENT_CI("shutdown",        shutdown,        3),
1069 /* 237 */ SYSENT_CI("recv",            recv,            4),
1070 /* 238 */ SYSENT_CI("recvfrom",        recvfrom32,     6),
1071 /* 239 */ SYSENT_CI("recvmsg",         recvmsg,         3),
1072 /* 240 */ SYSENT_CI("send",            send32,          4),
1073 /* 241 */ SYSENT_CI("sendmsg",         sendmsg,         3),
1074 /* 242 */ SYSENT_CI("sendto",          sendto32,       6),
1075 /* 243 */ SYSENT_CI("getpeername",     getpeername,    4),
1076 /* 244 */ SYSENT_CI("getsockname",     getsockname,    4),
1077 /* 245 */ SYSENT_CI("getsockopt",      getsockopt,     6),
1078 /* 246 */ SYSENT_CI("setsockopt",      setsockopt,     6),
1079 /* 247 */ SYSENT_CI("sockconfig",      sockconfig,     5),
1080 /* 248 */ SYSENT_CI("ntp_gettime",     ntp_gettime,    1),
1081 /* 249 */ SYSENT_CI("ntp_adjtime",     ntp_adjtime,    1),
1082 /* 250 */ SYSENT_CI("lwp_mutex_unlock", lwp_mutex_unlock, 1),
1083 /* 251 */ SYSENT_CI("lwp_mutex_trylock", lwp_mutex_trylock, 2),
1084 /* 252 */ SYSENT_CI("lwp_mutex_register", lwp_mutex_register, 2),
1085 /* 253 */ SYSENT_CI("cladm",           cladm,          3),
1086 /* 254 */ SYSENT_CI("uucopy",          uucopy,         3),
1087 /* 255 */ SYSENT_CI("umount2",         umount2,        2)
1088 };
_____unchanged_portion_omitted_____

```

```

*****
196545 Thu Jun 30 22:03:36 2016
new/usr/src/uts/common/os/zone.c
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
_____unchanged_portion_omitted_____

2009 /*
2010 * Called very early on in boot to initialize the ZSD list so that
2011 * zone_key_create() can be called before zone_init(). It also initializes
2012 * portions of zone0 which may be used before zone_init() is called. The
2013 * variable "global_zone" will be set when zone0 is fully initialized by
2014 * zone_init().
2015 */
2016 void
2017 zone_zsd_init(void)
2018 {
2019     mutex_init(&zonehash_lock, NULL, MUTEX_DEFAULT, NULL);
2020     mutex_init(&zsd_key_lock, NULL, MUTEX_DEFAULT, NULL);
2021     list_create(&zsd_registered_keys, sizeof (struct zsd_entry),
2022               offsetof(struct zsd_entry, zsd_linkage));
2023     list_create(&zone_active, sizeof (zone_t),
2024               offsetof(zone_t, zone_linkage));
2025     list_create(&zone_deathrow, sizeof (zone_t),
2026               offsetof(zone_t, zone_linkage));

2028     mutex_init(&zone0.zone_lock, NULL, MUTEX_DEFAULT, NULL);
2029     mutex_init(&zone0.zone_nlwps_lock, NULL, MUTEX_DEFAULT, NULL);
2030     mutex_init(&zone0.zone_mem_lock, NULL, MUTEX_DEFAULT, NULL);
2031     zone0.zone_shares = 1;
2032     zone0.zone_nlwps = 0;
2033     zone0.zone_nlwps_ctl = INT_MAX;
2034     zone0.zone_nprocs = 0;
2035     zone0.zone_nprocs_ctl = INT_MAX;
2036     zone0.zone_locked_mem = 0;
2037     zone0.zone_locked_mem_ctl = UINT64_MAX;
2038     ASSERT(zone0.zone_max_swap == 0);
2039     zone0.zone_max_swap_ctl = UINT64_MAX;
2040     zone0.zone_max_lofi = 0;
2041     zone0.zone_max_lofi_ctl = UINT64_MAX;
2042     zone0.zone_shmmax = 0;
2043     zone0.zone_ipc.ipcq_shmuni = 0;
2044     zone0.zone_ipc.ipcq_semuni = 0;
2045     zone0.zone_ipc.ipcq_msguni = 0;
2046     zone0.zone_name = GLOBAL_ZONENAME;
2047     zone0.zone_nodename = utsname.nodename;
2048     zone0.zone_domain = srpc_domain;
2049     zone0.zone_hostid = HW_INVALID_HOSTID;
2050     zone0.zone_fs_allowed = NULL;
2051     psecflags_default(&zone0.zone_secflags);
2052 #endif /* ! codereview */
2053     zone0.zone_ref = 1;
2054     zone0.zone_id = GLOBAL_ZONEID;
2055     zone0.zone_status = ZONE_IS_RUNNING;
2056     zone0.zone_rootpath = "/";
2057     zone0.zone_rootpathlen = 2;
2058     zone0.zone_psetid = ZONE_PS_INVAL;
2059     zone0.zone_ncpus = 0;
2060     zone0.zone_ncpus_online = 0;
2061     zone0.zone_proc_initpid = 1;
2062     zone0.zone_initname = initname;
2063     zone0.zone_lockedmem_kstat = NULL;
2064     zone0.zone_swapresv_kstat = NULL;

```

```

2065     zone0.zone_nprocs_kstat = NULL;

2067     zone0.zone_stime = 0;
2068     zone0.zone_utime = 0;
2069     zone0.zone_wtime = 0;

2071     list_create(&zone0.zone_ref_list, sizeof (zone_ref_t),
2072               offsetof(zone_ref_t, zref_linkage));
2073     list_create(&zone0.zone_zsd, sizeof (struct zsd_entry),
2074               offsetof(struct zsd_entry, zsd_linkage));
2075     list_insert_head(&zone_active, &zone0);

2077     /*
2078     * The root filesystem is not mounted yet, so zone_rootvp cannot be set
2079     * to anything meaningful. It is assigned to be 'rootdir' in
2080     * vfs_mountroot().
2081     */
2082     zone0.zone_rootvp = NULL;
2083     zone0.zone_vfslist = NULL;
2084     zone0.zone_bootargs = initargs;
2085     zone0.zone_privset = kmem_alloc(sizeof (priv_set_t), KM_SLEEP);
2086     /*
2087     * The global zone has all privileges
2088     */
2089     priv_fillset(zone0.zone_privset);
2090     /*
2091     * Add p0 to the global zone
2092     */
2093     zone0.zone_zsched = &p0;
2094     p0.p_zone = &zone0;
2095 }

2097 /*
2098 * Compute a hash value based on the contents of the label and the DOI. The
2099 * hash algorithm is somewhat arbitrary, but is based on the observation that
2100 * humans will likely pick labels that differ by amounts that work out to be
2101 * multiples of the number of hash chains, and thus stirring in some primes
2102 * should help.
2103 */
2104 static uint_t
2105 hash_bylabel(void *hdata, mod_hash_key_t key)
2106 {
2107     const ts_label_t *lab = (ts_label_t *)key;
2108     const uint32_t *up, *ue;
2109     uint_t hash;
2110     int i;

2112     _NOTE(ARGUNUSED(hdata));

2114     hash = lab->tsl_doi + (lab->tsl_doi << 1);
2115     /* we depend on alignment of label, but not representation */
2116     up = (const uint32_t *)&lab->tsl_label;
2117     ue = up + sizeof (lab->tsl_label) / sizeof (*up);
2118     i = 1;
2119     while (up < ue) {
2120         /* using 2^n + 1, 1 <= n <= 16 as source of many primes */
2121         hash += *up + (*up << ((i % 16) + 1));
2122         up++;
2123         i++;
2124     }
2125     return (hash);
2126 }

2128 /*
2129 * All that mod_hash cares about here is zero (equal) versus non-zero (not
2130 * equal). This may need to be changed if less than / greater than is ever

```

```

2131 * needed.
2132 */
2133 static int
2134 hash_labelkey_cmp(mod_hash_key_t key1, mod_hash_key_t key2)
2135 {
2136     ts_label_t *lab1 = (ts_label_t *)key1;
2137     ts_label_t *lab2 = (ts_label_t *)key2;
2138
2139     return (label_equal(lab1, lab2) ? 0 : 1);
2140 }
2141
2142 /*
2143  * Called by main() to initialize the zones framework.
2144  */
2145 void
2146 zone_init(void)
2147 {
2148     rctl_dict_entry_t *rde;
2149     rctl_val_t *dval;
2150     rctl_set_t *set;
2151     rctl_alloc_gp_t *gp;
2152     rctl_entity_p_t e;
2153     int res;
2154
2155     ASSERT(curproc == &p0);
2156
2157     /*
2158      * Create ID space for zone IDs. ID 0 is reserved for the
2159      * global zone.
2160      */
2161     zoneid_space = id_space_create("zoneid_space", 1, MAX_ZONEID);
2162
2163     /*
2164      * Initialize generic zone resource controls, if any.
2165      */
2166     rc_zone_cpu_shares = rctl_register("zone.cpu-shares",
2167         RCENTITY_ZONE, RCTL_GLOBAL_SIGNAL_NEVER | RCTL_GLOBAL_DENY_NEVER |
2168         RCTL_GLOBAL_NOBASIC | RCTL_GLOBAL_COUNT | RCTL_GLOBAL_SYSLOG_NEVER,
2169         FSS_MAXSHARES, FSS_MAXSHARES, &zone_cpu_shares_ops);
2170
2171     rc_zone_cpu_cap = rctl_register("zone.cpu-cap",
2172         RCENTITY_ZONE, RCTL_GLOBAL_SIGNAL_NEVER | RCTL_GLOBAL_DENY_ALWAYS |
2173         RCTL_GLOBAL_NOBASIC | RCTL_GLOBAL_COUNT | RCTL_GLOBAL_SYSLOG_NEVER |
2174         RCTL_GLOBAL_INFINITE,
2175         MAXCAP, MAXCAP, &zone_cpu_cap_ops);
2176
2177     rc_zone_nlwps = rctl_register("zone.max-lwps", RCENTITY_ZONE,
2178         RCTL_GLOBAL_NOACTION | RCTL_GLOBAL_NOBASIC | RCTL_GLOBAL_COUNT,
2179         INT_MAX, INT_MAX, &zone_lwps_ops);
2180
2181     rc_zone_nprocs = rctl_register("zone.max-processes", RCENTITY_ZONE,
2182         RCTL_GLOBAL_NOACTION | RCTL_GLOBAL_NOBASIC | RCTL_GLOBAL_COUNT,
2183         INT_MAX, INT_MAX, &zone_procs_ops);
2184
2185     /*
2186      * System V IPC resource controls
2187      */
2188     rc_zone_msgmni = rctl_register("zone.max-msg-ids",
2189         RCENTITY_ZONE, RCTL_GLOBAL_DENY_ALWAYS | RCTL_GLOBAL_NOBASIC |
2190         RCTL_GLOBAL_COUNT, IPC_IDS_MAX, IPC_IDS_MAX, &zone_msgmni_ops);
2191
2192     rc_zone_semni = rctl_register("zone.max-sem-ids",
2193         RCENTITY_ZONE, RCTL_GLOBAL_DENY_ALWAYS | RCTL_GLOBAL_NOBASIC |
2194         RCTL_GLOBAL_COUNT, IPC_IDS_MAX, IPC_IDS_MAX, &zone_semni_ops);
2195
2196     rc_zone_shmmni = rctl_register("zone.max-shm-ids",

```

```

2197         RCENTITY_ZONE, RCTL_GLOBAL_DENY_ALWAYS | RCTL_GLOBAL_NOBASIC |
2198         RCTL_GLOBAL_COUNT, IPC_IDS_MAX, IPC_IDS_MAX, &zone_shmmni_ops);
2199
2200     rc_zone_shmmax = rctl_register("zone.max-shm-memory",
2201         RCENTITY_ZONE, RCTL_GLOBAL_DENY_ALWAYS | RCTL_GLOBAL_NOBASIC |
2202         RCTL_GLOBAL_BYTES, UINT64_MAX, UINT64_MAX, &zone_shmmax_ops);
2203
2204     /*
2205      * Create a rctl_val with PRIVILEGED, NOACTION, value = 1. Then attach
2206      * this at the head of the rctl_dict_entry for 'zone.cpu-shares'.
2207      */
2208     dval = kmem_cache_alloc(rctl_val_cache, KM_SLEEP);
2209     bzero(dval, sizeof(rctl_val_t));
2210     dval->rcv_value = 1;
2211     dval->rcv_privilege = RCPRIV_PRIVILEGED;
2212     dval->rcv_flagaction = RCTL_LOCAL_NOACTION;
2213     dval->rcv_action_recip_pid = -1;
2214
2215     rde = rctl_dict_lookup("zone.cpu-shares");
2216     (void) rctl_val_list_insert(&rde->rcd_default_value, dval);
2217
2218     rc_zone_locked_mem = rctl_register("zone.max-locked-memory",
2219         RCENTITY_ZONE, RCTL_GLOBAL_NOBASIC | RCTL_GLOBAL_BYTES |
2220         RCTL_GLOBAL_DENY_ALWAYS, UINT64_MAX, UINT64_MAX,
2221         &zone_locked_mem_ops);
2222
2223     rc_zone_max_swap = rctl_register("zone.max-swap",
2224         RCENTITY_ZONE, RCTL_GLOBAL_NOBASIC | RCTL_GLOBAL_BYTES |
2225         RCTL_GLOBAL_DENY_ALWAYS, UINT64_MAX, UINT64_MAX,
2226         &zone_max_swap_ops);
2227
2228     rc_zone_max_lofi = rctl_register("zone.max-lofi",
2229         RCENTITY_ZONE, RCTL_GLOBAL_NOBASIC | RCTL_GLOBAL_COUNT |
2230         RCTL_GLOBAL_DENY_ALWAYS, UINT64_MAX, UINT64_MAX,
2231         &zone_max_lofi_ops);
2232
2233     /*
2234      * Initialize the 'global zone'.
2235      */
2236     set = rctl_set_create();
2237     gp = rctl_set_init_prealloc(RCENTITY_ZONE);
2238     mutex_enter(&p0.p_lock);
2239     e.rcep_p.zone = &zone0;
2240     e.rcep_t = RCENTITY_ZONE;
2241     zone0.zone_rctls = rctl_set_init(RCENTITY_ZONE, &p0, &e, set,
2242         gp);
2243
2244     zone0.zone_nlwps = p0.p_lwpcnt;
2245     zone0.zone_nprocs = 1;
2246     zone0.zone_ntasks = 1;
2247     mutex_exit(&p0.p_lock);
2248     zone0.zone_restart_init = B_TRUE;
2249     zone0.zone_brand = &native_brand;
2250     rctl_prealloc_destroy(gp);
2251     /*
2252      * pool_default hasn't been initialized yet, so we let pool_init()
2253      * take care of making sure the global zone is in the default pool.
2254      */
2255
2256     /*
2257      * Initialize global zone kstats
2258      */
2259     zone_kstat_create(&zone0);
2260
2261     /*
2262      * Initialize zone label.

```

```

2263  * mlp are initialized when tnzonecfg is loaded.
2264  */
2265  zone0.zone_slab = 1_admin_low;
2266  rw_init(&zone0.zone_mlps.mlpl_rwlock, NULL, RW_DEFAULT, NULL);
2267  label_hold(1_admin_low);

2269  /*
2270  * Initialise the lock for the database structure used by mntfs.
2271  */
2272  rw_init(&zone0.zone_mntfs_db_lock, NULL, RW_DEFAULT, NULL);

2274  mutex_enter(&zonehash_lock);
2275  zone_uniqid(&zone0);
2276  ASSERT(zone0.zone_uniqid == GLOBAL_ZONEUNIQID);

2278  zonehashbyid = mod_hash_create_idhash("zone_by_id", zone_hash_size,
2279  mod_hash_null_valdtor);
2280  zonehashbyname = mod_hash_create_strhash("zone_by_name",
2281  zone_hash_size, mod_hash_null_valdtor);
2282  /*
2283  * maintain zonehashbylabel only for labeled systems
2284  */
2285  if (is_system_labeled())
2286  zonehashbylabel = mod_hash_create_extended("zone_by_label",
2287  zone_hash_size, mod_hash_null_keydtor,
2288  mod_hash_null_valdtor, hash_bylabel, NULL,
2289  hash_labelkey_cmp, KM_SLEEP);
2290  zonecount = 1;

2292  (void) mod_hash_insert(zonehashbyid, (mod_hash_key_t)GLOBAL_ZONEID,
2293  (mod_hash_val_t)&zone0);
2294  (void) mod_hash_insert(zonehashbyname, (mod_hash_key_t)zone0.zone_name,
2295  (mod_hash_val_t)&zone0);
2296  if (is_system_labeled()) {
2297  zone0.zone_flags |= ZF_HASHED_LABEL;
2298  (void) mod_hash_insert(zonehashbylabel,
2299  (mod_hash_key_t)zone0.zone_slab, (mod_hash_val_t)&zone0);
2300  }
2301  mutex_exit(&zonehash_lock);

2303  /*
2304  * We avoid setting zone_kcred until now, since kcred is initialized
2305  * sometime after zone_zsd_init() and before zone_init().
2306  */
2307  zone0.zone_kcred = kcred;
2308  /*
2309  * The global zone is fully initialized (except for zone_rootvp which
2310  * will be set when the root filesystem is mounted).
2311  */
2312  global_zone = &zone0;

2314  /*
2315  * Setup an event channel to send zone status change notifications on
2316  */
2317  res = sysevent_evcb_bind(ZONE_EVENT_CHANNEL, &zone_event_chan,
2318  EVCH_CREAT);

2320  if (res)
2321  panic("Sysevent_evcb_bind failed during zone setup.\n");

2323 }

2325 static void
2326 zone_free(zone_t *zone)
2327 {
2328  ASSERT(zone != global_zone);

```

```

2329  ASSERT(zone->zone_ntasks == 0);
2330  ASSERT(zone->zone_nlwps == 0);
2331  ASSERT(zone->zone_nprocs == 0);
2332  ASSERT(zone->zone_cred_ref == 0);
2333  ASSERT(zone->zone_kcred == NULL);
2334  ASSERT(zone_status_get(zone) == ZONE_IS_DEAD ||
2335  zone_status_get(zone) == ZONE_IS_UNINITIALIZED);
2336  ASSERT(list_is_empty(&zone->zone_ref_list));

2338  /*
2339  * Remove any zone caps.
2340  */
2341  cpucaps_zone_remove(zone);

2343  ASSERT(zone->zone_cpucap == NULL);

2345  /* remove from deathrow list */
2346  if (zone_status_get(zone) == ZONE_IS_DEAD) {
2347  ASSERT(zone->zone_ref == 0);
2348  mutex_enter(&zone_deathrow_lock);
2349  list_remove(&zone_deathrow, zone);
2350  mutex_exit(&zone_deathrow_lock);
2351  }

2353  list_destroy(&zone->zone_ref_list);
2354  zone_free_zsd(zone);
2355  zone_free_datasets(zone);
2356  list_destroy(&zone->zone_dl_list);

2358  if (zone->zone_rootvp != NULL)
2359  VN_RELE(zone->zone_rootvp);
2360  if (zone->zone_rootpath)
2361  kmem_free(zone->zone_rootpath, zone->zone_rootpathlen);
2362  if (zone->zone_name != NULL)
2363  kmem_free(zone->zone_name, ZONENAME_MAX);
2364  if (zone->zone_slab != NULL)
2365  label_rele(zone->zone_slab);
2366  if (zone->zone_nodename != NULL)
2367  kmem_free(zone->zone_nodename, _SYS_NMLN);
2368  if (zone->zone_domain != NULL)
2369  kmem_free(zone->zone_domain, _SYS_NMLN);
2370  if (zone->zone_privset != NULL)
2371  kmem_free(zone->zone_privset, sizeof(priv_set_t));
2372  if (zone->zone_rctl != NULL)
2373  rctl_set_free(zone->zone_rctl);
2374  if (zone->zone_bootargs != NULL)
2375  strfree(zone->zone_bootargs);
2376  if (zone->zone_initname != NULL)
2377  strfree(zone->zone_initname);
2378  if (zone->zone_fs_allowed != NULL)
2379  strfree(zone->zone_fs_allowed);
2380  if (zone->zone_pfexecd != NULL)
2381  klpd_freelist(&zone->zone_pfexecd);
2382  id_free(zoneid_space, zone->zone_id);
2383  mutex_destroy(&zone->zone_lock);
2384  cv_destroy(&zone->zone_cv);
2385  rw_destroy(&zone->zone_mlps.mlpl_rwlock);
2386  rw_destroy(&zone->zone_mntfs_db_lock);
2387  kmem_free(zone, sizeof(zone_t));
2388  }

2390  /*
2391  * See block comment at the top of this file for information about zone
2392  * status values.
2393  */
2394  /*

```

```

2395 * Convenience function for setting zone status.
2396 */
2397 static void
2398 zone_status_set(zone_t *zone, zone_status_t status)
2399 {
2401     nvlist_t *nvl = NULL;
2402     ASSERT(MUTEX_HELD(&zone_status_lock));
2403     ASSERT(status > ZONE_MIN_STATE && status <= ZONE_MAX_STATE &&
2404            status >= zone_status_get(zone));
2406     if (nvlist_alloc(&nvl, NV_UNIQUE_NAME, KM_SLEEP) ||
2407         nvlist_add_string(nvl, ZONE_CB_NAME, zone->zone_name) ||
2408         nvlist_add_string(nvl, ZONE_CB_NEWSTATE,
2409                          zone_status_table[status]) ||
2410         nvlist_add_string(nvl, ZONE_CB_OLDSTATE,
2411                          zone_status_table[zone->zone_status]) ||
2412         nvlist_add_int32(nvl, ZONE_CB_ZONEID, zone->zone_id) ||
2413         nvlist_add_uint64(nvl, ZONE_CB_TIMESTAMP, (uint64_t)gethrtime()) ||
2414         sysevent_evt_publish(zone_event_chan, ZONE_EVENT_STATUS_CLASS,
2415                             ZONE_EVENT_STATUS_SUBCLASS, "sun.com", "kernel", nvl, EVCH_SLEEP)) {
2416 #ifdef DEBUG
2417         (void) printf(
2418             "Failed to allocate and send zone state change event.\n");
2419 #endif
2420     }
2421     nvlist_free(nvl);
2423     zone->zone_status = status;
2425     cv_broadcast(&zone->zone_cv);
2426 }
2428 /*
2429 * Public function to retrieve the zone status. The zone status may
2430 * change after it is retrieved.
2431 */
2432 zone_status_t
2433 zone_status_get(zone_t *zone)
2434 {
2435     return (zone->zone_status);
2436 }
2438 static int
2439 zone_set_bootargs(zone_t *zone, const char *zone_bootargs)
2440 {
2441     char *buf = kmem_zalloc(BOOTARGS_MAX, KM_SLEEP);
2442     int err = 0;
2444     ASSERT(zone != global_zone);
2445     if ((err = copyinstr(zone_bootargs, buf, BOOTARGS_MAX, NULL)) != 0)
2446         goto done; /* EFAULT or ENAMETOOLONG */
2448     if (zone->zone_bootargs != NULL)
2449         strfree(zone->zone_bootargs);
2451     zone->zone_bootargs = strdup(buf);
2453 done:
2454     kmem_free(buf, BOOTARGS_MAX);
2455     return (err);
2456 }
2458 static int
2459 zone_set_brand(zone_t *zone, const char *brand)
2460 {

```

```

2461     struct brand_attr *attrp;
2462     brand_t *bp;
2464     attrp = kmem_alloc(sizeof (struct brand_attr), KM_SLEEP);
2465     if (copyin(brand, attrp, sizeof (struct brand_attr)) != 0) {
2466         kmem_free(attrp, sizeof (struct brand_attr));
2467         return (EFAULT);
2468     }
2470     bp = brand_register_zone(attrp);
2471     kmem_free(attrp, sizeof (struct brand_attr));
2472     if (bp == NULL)
2473         return (EINVAL);
2475     /*
2476     * This is the only place where a zone can change it's brand.
2477     * We already need to hold zone_status_lock to check the zone
2478     * status, so we'll just use that lock to serialize zone
2479     * branding requests as well.
2480     */
2481     mutex_enter(&zone_status_lock);
2483     /* Re-Branding is not allowed and the zone can't be booted yet */
2484     if ((ZONE_IS_BRANDED(zone) ||
2485         (zone_status_get(zone) >= ZONE_IS_BOOTING)) {
2486         mutex_exit(&zone_status_lock);
2487         brand_unregister_zone(bp);
2488         return (EINVAL);
2489     }
2491     /* set up the brand specific data */
2492     zone->zone_brand = bp;
2493     ZBROP(zone)->b_init_brand_data(zone);
2495     mutex_exit(&zone_status_lock);
2496     return (0);
2497 }
2499 static int
2500 zone_set_secflags(zone_t *zone, const psecflags_t *zone_secflags)
2501 {
2502     int err = 0;
2503     psecflags_t psf;
2505     ASSERT(zone != global_zone);
2507     if ((err = copyin(zone_secflags, &psf, sizeof (psf)) != 0)
2508         return (err);
2510     if (zone_status_get(zone) > ZONE_IS_READY)
2511         return (EINVAL);
2513     if (!psecflags_validate(&psf))
2514         return (EINVAL);
2516     (void) memcpy(&zone->zone_secflags, &psf, sizeof (psf));
2518     /* Set security flags on the zone's zsched */
2519     (void) memcpy(&zone->zone_zsched->p_secflags, &zone->zone_secflags,
2520                sizeof (zone->zone_zsched->p_secflags));
2522     return (0);
2523 }
2525 static int
2526 #endif /* ! codereview */

```

```

2527 zone_set_fs_allowed(zone_t *zone, const char *zone_fs_allowed)
2528 {
2529     char *buf = kmem_zalloc(ZONE_FS_ALLOWED_MAX, KM_SLEEP);
2530     int err = 0;
2531
2532     ASSERT(zone != global_zone);
2533     if ((err = copyinstr(zone_fs_allowed, buf,
2534         ZONE_FS_ALLOWED_MAX, NULL)) != 0)
2535         goto done;
2536
2537     if (zone->zone_fs_allowed != NULL)
2538         strfree(zone->zone_fs_allowed);
2539
2540     zone->zone_fs_allowed = strdup(buf);
2541
2542 done:
2543     kmem_free(buf, ZONE_FS_ALLOWED_MAX);
2544     return (err);
2545 }
2546
2547 static int
2548 zone_set_initname(zone_t *zone, const char *zone_initname)
2549 {
2550     char initname[INITNAME_SZ];
2551     size_t len;
2552     int err = 0;
2553
2554     ASSERT(zone != global_zone);
2555     if ((err = copyinstr(zone_initname, initname, INITNAME_SZ, &len)) != 0)
2556         return (err); /* EFAULT or ENAMETOOLONG */
2557
2558     if (zone->zone_initname != NULL)
2559         strfree(zone->zone_initname);
2560
2561     zone->zone_initname = kmem_alloc(strlen(initname) + 1, KM_SLEEP);
2562     (void) strcpy(zone->zone_initname, initname);
2563     return (0);
2564 }
2565
2566 static int
2567 zone_set_phys_mcap(zone_t *zone, const uint64_t *zone_mcap)
2568 {
2569     uint64_t mcap;
2570     int err = 0;
2571
2572     if ((err = copyin(zone_mcap, &mcap, sizeof (uint64_t))) == 0)
2573         zone->zone_phys_mcap = mcap;
2574
2575     return (err);
2576 }
2577
2578 static int
2579 zone_set_sched_class(zone_t *zone, const char *new_class)
2580 {
2581     char sched_class[PC_CLNMSZ];
2582     id_t classid;
2583     int err;
2584
2585     ASSERT(zone != global_zone);
2586     if ((err = copyinstr(new_class, sched_class, PC_CLNMSZ, NULL)) != 0)
2587         return (err); /* EFAULT or ENAMETOOLONG */
2588
2589     if (getcid(sched_class, &classid) != 0 || CLASS_KERNEL(classid))
2590         return (set_errno(EINVAL));
2591     zone->zone_defaultcid = classid;
2592     ASSERT(zone->zone_defaultcid > 0 &&

```

```

2593         zone->zone_defaultcid < loaded_classes);
2594
2595     return (0);
2596 }
2597
2598 /*
2599  * Block indefinitely waiting for (zone_status >= status)
2600  */
2601 void
2602 zone_status_wait(zone_t *zone, zone_status_t status)
2603 {
2604     ASSERT(status > ZONE_MIN_STATE && status <= ZONE_MAX_STATE);
2605
2606     mutex_enter(&zone_status_lock);
2607     while (zone->zone_status < status) {
2608         cv_wait(&zone->zone_cv, &zone_status_lock);
2609     }
2610     mutex_exit(&zone_status_lock);
2611 }
2612
2613 /*
2614  * Private CPR-safe version of zone_status_wait().
2615  */
2616 static void
2617 zone_status_wait_cpr(zone_t *zone, zone_status_t status, char *str)
2618 {
2619     callb_cpr_t cprinfo;
2620
2621     ASSERT(status > ZONE_MIN_STATE && status <= ZONE_MAX_STATE);
2622
2623     CALLB_CPR_INIT(&cprinfo, &zone_status_lock, callb_generic_cpr,
2624         str);
2625     mutex_enter(&zone_status_lock);
2626     while (zone->zone_status < status) {
2627         CALLB_CPR_SAFE_BEGIN(&cprinfo);
2628         cv_wait(&zone->zone_cv, &zone_status_lock);
2629         CALLB_CPR_SAFE_END(&cprinfo, &zone_status_lock);
2630     }
2631     /*
2632      * zone_status_lock is implicitly released by the following.
2633      */
2634     CALLB_CPR_EXIT(&cprinfo);
2635 }
2636
2637 /*
2638  * Block until zone enters requested state or signal is received. Return (0)
2639  * if signaled, non-zero otherwise.
2640  */
2641 int
2642 zone_status_wait_sig(zone_t *zone, zone_status_t status)
2643 {
2644     ASSERT(status > ZONE_MIN_STATE && status <= ZONE_MAX_STATE);
2645
2646     mutex_enter(&zone_status_lock);
2647     while (zone->zone_status < status) {
2648         if (!cv_wait_sig(&zone->zone_cv, &zone_status_lock)) {
2649             mutex_exit(&zone_status_lock);
2650             return (0);
2651         }
2652     }
2653     mutex_exit(&zone_status_lock);
2654     return (1);
2655 }
2656
2657 /*
2658  * Block until the zone enters the requested state or the timeout expires,

```



```

2659 * whichever happens first. Return (-1) if operation timed out, time remaining
2660 * otherwise.
2661 */
2662 clock_t
2663 zone_status_timedwait(zone_t *zone, clock_t tim, zone_status_t status)
2664 {
2665     clock_t timeleft = 0;

2667     ASSERT(status > ZONE_MIN_STATE && status <= ZONE_MAX_STATE);

2669     mutex_enter(&zone_status_lock);
2670     while (zone->zone_status < status && timeleft != -1) {
2671         timeleft = cv_timedwait(&zone->zone_cv, &zone_status_lock, tim);
2672     }
2673     mutex_exit(&zone_status_lock);
2674     return (timeleft);
2675 }

2677 /*
2678 * Block until the zone enters the requested state, the current process is
2679 * signaled, or the timeout expires, whichever happens first. Return (-1) if
2680 * operation timed out, 0 if signaled, time remaining otherwise.
2681 */
2682 clock_t
2683 zone_status_timedwait_sig(zone_t *zone, clock_t tim, zone_status_t status)
2684 {
2685     clock_t timeleft = tim - ddi_get_lbolt();

2687     ASSERT(status > ZONE_MIN_STATE && status <= ZONE_MAX_STATE);

2689     mutex_enter(&zone_status_lock);
2690     while (zone->zone_status < status) {
2691         timeleft = cv_timedwait_sig(&zone->zone_cv, &zone_status_lock,
2692             tim);
2693         if (timeleft <= 0)
2694             break;
2695     }
2696     mutex_exit(&zone_status_lock);
2697     return (timeleft);
2698 }

2700 /*
2701 * Zones have two reference counts: one for references from credential
2702 * structures (zone_cred_ref), and one (zone_ref) for everything else.
2703 * This is so we can allow a zone to be rebooted while there are still
2704 * outstanding cred references, since certain drivers cache dblks (which
2705 * implicitly results in cached creds). We wait for zone_ref to drop to
2706 * 0 (actually 1), but not zone_cred_ref. The zone structure itself is
2707 * later freed when the zone_cred_ref drops to 0, though nothing other
2708 * than the zone id and privilege set should be accessed once the zone
2709 * is "dead".
2710 *
2711 * A debugging flag, zone_wait_for_cred, can be set to a non-zero value
2712 * to force halt/reboot to block waiting for the zone_cred_ref to drop
2713 * to 0. This can be useful to flush out other sources of cached creds
2714 * that may be less innocuous than the driver case.
2715 *
2716 * Zones also provide a tracked reference counting mechanism in which zone
2717 * references are represented by "crumbs" (zone_ref structures). Crumbs help
2718 * debuggers determine the sources of leaked zone references. See
2719 * zone_hold_ref() and zone_rele_ref() below for more information.
2720 */

2722 int zone_wait_for_cred = 0;

2724 static void

```

```

2725 zone_hold_locked(zone_t *z)
2726 {
2727     ASSERT(MUTEX_HELD(&z->zone_lock));
2728     z->zone_ref++;
2729     ASSERT(z->zone_ref != 0);
2730 }

2732 /*
2733 * Increment the specified zone's reference count. The zone's zone_t structure
2734 * will not be freed as long as the zone's reference count is nonzero.
2735 * Decrement the zone's reference count via zone_rele().
2736 *
2737 * NOTE: This function should only be used to hold zones for short periods of
2738 * time. Use zone_hold_ref() if the zone must be held for a long time.
2739 */
2740 void
2741 zone_hold(zone_t *z)
2742 {
2743     mutex_enter(&z->zone_lock);
2744     zone_hold_locked(z);
2745     mutex_exit(&z->zone_lock);
2746 }

2748 /*
2749 * If the non-cred ref count drops to 1 and either the cred ref count
2750 * is 0 or we aren't waiting for cred references, the zone is ready to
2751 * be destroyed.
2752 */
2753 #define ZONE_IS_UNREF(zone) ((zone)->zone_ref == 1 && \
2754     (!zone_wait_for_cred || (zone)->zone_cred_ref == 0))

2756 /*
2757 * Common zone reference release function invoked by zone_rele() and
2758 * zone_rele_ref(). If subsys is ZONE_REF_NUM_SUBSYS, then the specified
2759 * zone's subsystem-specific reference counters are not affected by the
2760 * release. If ref is not NULL, then the zone_ref_t to which it refers is
2761 * removed from the specified zone's reference list. ref must be non-NULL iff
2762 * subsys is not ZONE_REF_NUM_SUBSYS.
2763 */
2764 static void
2765 zone_rele_common(zone_t *z, zone_ref_t *ref, zone_ref_subsys_t subsys)
2766 {
2767     boolean_t wakeup;

2769     mutex_enter(&z->zone_lock);
2770     ASSERT(z->zone_ref != 0);
2771     z->zone_ref--;
2772     if (subsys != ZONE_REF_NUM_SUBSYS) {
2773         ASSERT(z->zone_subsys_ref[subsys] != 0);
2774         z->zone_subsys_ref[subsys]--;
2775         list_remove(&z->zone_ref_list, ref);
2776     }
2777     if (z->zone_ref == 0 && z->zone_cred_ref == 0) {
2778         /* no more refs, free the structure */
2779         mutex_exit(&z->zone_lock);
2780         zone_free(z);
2781         return;
2782     }
2783     /* signal zone_destroy so the zone can finish halting */
2784     wakeup = (ZONE_IS_UNREF(z) && zone_status_get(z) >= ZONE_IS_DEAD);
2785     mutex_exit(&z->zone_lock);

2787     if (wakeup) {
2788         /*
2789          * Grabbing zonehash_lock here effectively synchronizes with
2790          * zone_destroy() to avoid missed signals.

```

```

2791     */
2792     mutex_enter(&zonehash_lock);
2793     cv_broadcast(&zone_destroy_cv);
2794     mutex_exit(&zonehash_lock);
2795 }
2796 }

2798 /*
2799  * Decrement the specified zone's reference count. The specified zone will
2800  * cease to exist after this function returns if the reference count drops to
2801  * zero. This function should be paired with zone_hold().
2802  */
2803 void
2804 zone_rele(zone_t *z)
2805 {
2806     zone_rele_common(z, NULL, ZONE_REF_NUM_SUBSYS);
2807 }

2809 /*
2810  * Initialize a zone reference structure. This function must be invoked for
2811  * a reference structure before the structure is passed to zone_hold_ref().
2812  */
2813 void
2814 zone_init_ref(zone_ref_t *ref)
2815 {
2816     ref->zref_zone = NULL;
2817     list_link_init(&ref->zref_linkage);
2818 }

2820 /*
2821  * Acquire a reference to zone z. The caller must specify the
2822  * zone_ref_subsys_t constant associated with its subsystem. The specified
2823  * zone_ref_t structure will represent a reference to the specified zone. Use
2824  * zone_rele_ref() to release the reference.
2825  *
2826  * The referenced zone_t structure will not be freed as long as the zone_t's
2827  * zone_status field is not ZONE_IS_DEAD and the zone has outstanding
2828  * references.
2829  *
2830  * NOTE: The zone_ref_t structure must be initialized before it is used.
2831  * See zone_init_ref() above.
2832  */
2833 void
2834 zone_hold_ref(zone_t *z, zone_ref_t *ref, zone_ref_subsys_t subsys)
2835 {
2836     ASSERT(subsys >= 0 && subsys < ZONE_REF_NUM_SUBSYS);

2838     /*
2839      * Prevent consumers from reusing a reference structure before
2840      * releasing it.
2841      */
2842     VERIFY(ref->zref_zone == NULL);

2844     ref->zref_zone = z;
2845     mutex_enter(&z->zone_lock);
2846     zone_hold_locked(z);
2847     z->zone_subsys_ref[subsys]++;
2848     ASSERT(z->zone_subsys_ref[subsys] != 0);
2849     list_insert_head(&z->zone_ref_list, ref);
2850     mutex_exit(&z->zone_lock);
2851 }

2853 /*
2854  * Release the zone reference represented by the specified zone_ref_t.
2855  * The reference is invalid after it's released; however, the zone_ref_t
2856  * structure can be reused without having to invoke zone_init_ref().

```

```

2857  * subsys should be the same value that was passed to zone_hold_ref()
2858  * when the reference was acquired.
2859  */
2860 void
2861 zone_rele_ref(zone_ref_t *ref, zone_ref_subsys_t subsys)
2862 {
2863     zone_rele_common(ref->zref_zone, ref, subsys);

2865     /*
2866      * Set the zone_ref_t's zref_zone field to NULL to generate panics
2867      * when consumers dereference the reference. This helps us catch
2868      * consumers who use released references. Furthermore, this lets
2869      * consumers reuse the zone_ref_t structure without having to
2870      * invoke zone_init_ref().
2871      */
2872     ref->zref_zone = NULL;
2873 }

2875 void
2876 zone_cred_hold(zone_t *z)
2877 {
2878     mutex_enter(&z->zone_lock);
2879     z->zone_cred_ref++;
2880     ASSERT(z->zone_cred_ref != 0);
2881     mutex_exit(&z->zone_lock);
2882 }

2884 void
2885 zone_cred_rele(zone_t *z)
2886 {
2887     boolean_t wakeup;

2889     mutex_enter(&z->zone_lock);
2890     ASSERT(z->zone_cred_ref != 0);
2891     z->zone_cred_ref--;
2892     if (z->zone_cred_ref == 0 && z->zone_cred_ref == 0) {
2893         /* no more refs, free the structure */
2894         mutex_exit(&z->zone_lock);
2895         zone_free(z);
2896         return;
2897     }
2898     /*
2899      * If zone_destroy is waiting for the cred references to drain
2900      * out, and they have, signal it.
2901      */
2902     wakeup = (zone_wait_for_cred && ZONE_IS_UNREF(z) &&
2903             zone_status_get(z) >= ZONE_IS_DEAD);
2904     mutex_exit(&z->zone_lock);

2906     if (wakeup) {
2907         /*
2908          * Grabbing zonehash_lock here effectively synchronizes with
2909          * zone_destroy() to avoid missed signals.
2910          */
2911         mutex_enter(&zonehash_lock);
2912         cv_broadcast(&zone_destroy_cv);
2913         mutex_exit(&zonehash_lock);
2914     }
2915 }

2917 void
2918 zone_task_hold(zone_t *z)
2919 {
2920     mutex_enter(&z->zone_lock);
2921     z->zone_ntasks++;
2922     ASSERT(z->zone_ntasks != 0);

```

```

2923     mutex_exit(&z->zone_lock);
2924 }

2926 void
2927 zone_task_rele(zone_t *zone)
2928 {
2929     uint_t refcnt;

2931     mutex_enter(&zone->zone_lock);
2932     ASSERT(zone->zone_ntasks != 0);
2933     refcnt = --zone->zone_ntasks;
2934     if (refcnt > 1) { /* Common case */
2935         mutex_exit(&zone->zone_lock);
2936         return;
2937     }
2938     zone_hold_locked(zone); /* so we can use the zone_t later */
2939     mutex_exit(&zone->zone_lock);
2940     if (refcnt == 1) {
2941         /*
2942          * See if the zone is shutting down.
2943          */
2944         mutex_enter(&zone_status_lock);
2945         if (zone_status_get(zone) != ZONE_IS_SHUTTING_DOWN) {
2946             goto out;
2947         }

2949         /*
2950          * Make sure the ntasks didn't change since we
2951          * dropped zone_lock.
2952          */
2953         mutex_enter(&zone->zone_lock);
2954         if (refcnt != zone->zone_ntasks) {
2955             mutex_exit(&zone->zone_lock);
2956             goto out;
2957         }
2958         mutex_exit(&zone->zone_lock);

2960         /*
2961          * No more user processes in the zone. The zone is empty.
2962          */
2963         zone_status_set(zone, ZONE_IS_EMPTY);
2964         goto out;
2965     }

2967     ASSERT(refcnt == 0);
2968     /*
2969      * zsched has exited; the zone is dead.
2970      */
2971     zone->zone_zsched = NULL; /* paranoia */
2972     mutex_enter(&zone_status_lock);
2973     zone_status_set(zone, ZONE_IS_DEAD);
2974 out:
2975     mutex_exit(&zone_status_lock);
2976     zone_rele(zone);
2977 }

2979 zoneid_t
2980 getzoneid(void)
2981 {
2982     return (curproc->p_zone->zone_id);
2983 }

2985 /*
2986  * Internal versions of zone_find_by_*. These don't zone_hold() or
2987  * check the validity of a zone's state.
2988  */

```

```

2989 static zone_t *
2990 zone_find_all_by_id(zoneid_t zoneid)
2991 {
2992     mod_hash_val_t hv;
2993     zone_t *zone = NULL;

2995     ASSERT(MUTEX_HELD(&zonehash_lock));

2997     if (mod_hash_find(zonehashbyid,
2998         (mod_hash_key_t)(uintptr_t)zoneid, &hv) == 0)
2999         zone = (zone_t *)hv;
3000     return (zone);
3001 }

3003 static zone_t *
3004 zone_find_all_by_label(const ts_label_t *label)
3005 {
3006     mod_hash_val_t hv;
3007     zone_t *zone = NULL;

3009     ASSERT(MUTEX_HELD(&zonehash_lock));

3011     /*
3012      * zonehashbylabel is not maintained for unlabeled systems
3013      */
3014     if (!is_system_labeled())
3015         return (NULL);
3016     if (mod_hash_find(zonehashbylabel, (mod_hash_key_t)label, &hv) == 0)
3017         zone = (zone_t *)hv;
3018     return (zone);
3019 }

3021 static zone_t *
3022 zone_find_all_by_name(char *name)
3023 {
3024     mod_hash_val_t hv;
3025     zone_t *zone = NULL;

3027     ASSERT(MUTEX_HELD(&zonehash_lock));

3029     if (mod_hash_find(zonehashbyname, (mod_hash_key_t)name, &hv) == 0)
3030         zone = (zone_t *)hv;
3031     return (zone);
3032 }

3034 /*
3035  * Public interface for looking up a zone by zoneid. Only returns the zone if
3036  * it is fully initialized, and has not yet begun the zone_destroy() sequence.
3037  * Caller must call zone_rele() once it is done with the zone.
3038  *
3039  * The zone may begin the zone_destroy() sequence immediately after this
3040  * function returns, but may be safely used until zone_rele() is called.
3041  */
3042 zone_t *
3043 zone_find_by_id(zoneid_t zoneid)
3044 {
3045     zone_t *zone;
3046     zone_status_t status;

3048     mutex_enter(&zonehash_lock);
3049     if ((zone = zone_find_all_by_id(zoneid)) == NULL) {
3050         mutex_exit(&zonehash_lock);
3051         return (NULL);
3052     }
3053     status = zone_status_get(zone);
3054     if (status < ZONE_IS_READY || status > ZONE_IS_DOWN) {

```

```

3055     /*
3056     * For all practical purposes the zone doesn't exist.
3057     */
3058     mutex_exit(&zonehash_lock);
3059     return (NULL);
3060 }
3061 zone_hold(zone);
3062 mutex_exit(&zonehash_lock);
3063 return (zone);
3064 }

3066 /*
3067 * Similar to zone_find_by_id, but using zone label as the key.
3068 */
3069 zone_t *
3070 zone_find_by_label(const ts_label_t *label)
3071 {
3072     zone_t *zone;
3073     zone_status_t status;

3075     mutex_enter(&zonehash_lock);
3076     if ((zone = zone_find_all_by_label(label)) == NULL) {
3077         mutex_exit(&zonehash_lock);
3078         return (NULL);
3079     }

3081     status = zone_status_get(zone);
3082     if (status > ZONE_IS_DOWN) {
3083         /*
3084         * For all practical purposes the zone doesn't exist.
3085         */
3086         mutex_exit(&zonehash_lock);
3087         return (NULL);
3088     }
3089     zone_hold(zone);
3090     mutex_exit(&zonehash_lock);
3091     return (zone);
3092 }

3094 /*
3095 * Similar to zone_find_by_id, but using zone name as the key.
3096 */
3097 zone_t *
3098 zone_find_by_name(char *name)
3099 {
3100     zone_t *zone;
3101     zone_status_t status;

3103     mutex_enter(&zonehash_lock);
3104     if ((zone = zone_find_all_by_name(name)) == NULL) {
3105         mutex_exit(&zonehash_lock);
3106         return (NULL);
3107     }
3108     status = zone_status_get(zone);
3109     if (status < ZONE_IS_READY || status > ZONE_IS_DOWN) {
3110         /*
3111         * For all practical purposes the zone doesn't exist.
3112         */
3113         mutex_exit(&zonehash_lock);
3114         return (NULL);
3115     }
3116     zone_hold(zone);
3117     mutex_exit(&zonehash_lock);
3118     return (zone);
3119 }

```

```

3121 /*
3122 * Similar to zone_find_by_id(), using the path as a key. For instance,
3123 * if there is a zone "foo" rooted at /foo/root, and the path argument
3124 * is "/foo/root/proc", it will return the held zone_t corresponding to
3125 * zone "foo".
3126 *
3127 * zone_find_by_path() always returns a non-NULL value, since at the
3128 * very least every path will be contained in the global zone.
3129 *
3130 * As with the other zone_find_by_*() functions, the caller is
3131 * responsible for zone_rele()ing the return value of this function.
3132 */
3133 zone_t *
3134 zone_find_by_path(const char *path)
3135 {
3136     zone_t *zone;
3137     zone_t *zret = NULL;
3138     zone_status_t status;

3140     if (path == NULL) {
3141         /*
3142         * Call from rootconf().
3143         */
3144         zone_hold(global_zone);
3145         return (global_zone);
3146     }
3147     ASSERT(*path == '/');
3148     mutex_enter(&zonehash_lock);
3149     for (zone = list_head(&zone_active); zone != NULL;
3150          zone = list_next(&zone_active, zone)) {
3151         if (ZONE_PATH_VISIBLE(path, zone))
3152             zret = zone;
3153     }
3154     ASSERT(zret != NULL);
3155     status = zone_status_get(zret);
3156     if (status < ZONE_IS_READY || status > ZONE_IS_DOWN) {
3157         /*
3158         * Zone practically doesn't exist.
3159         */
3160         zret = global_zone;
3161     }
3162     zone_hold(zret);
3163     mutex_exit(&zonehash_lock);
3164     return (zret);
3165 }

3167 /*
3168 * Public interface for updating per-zone load averages. Called once per
3169 * second.
3170 *
3171 * Based on loadavg_update(), genloadavg() and calclloadavg() from clock.c.
3172 */
3173 void
3174 zone_loadavg_update()
3175 {
3176     zone_t *zp;
3177     zone_status_t status;
3178     struct loadavg_s *lavg;
3179     hrtime_t zone_total;
3180     int i;
3181     hrtime_t hr_avg;
3182     int nrun;
3183     static int64_t f[3] = { 135, 27, 9 };
3184     int64_t q, r;

3186     mutex_enter(&zonehash_lock);

```

```

3187     for (zp = list_head(&zone_active); zp != NULL;
3188          zp = list_next(&zone_active, zp)) {
3189         mutex_enter(&zp->zone_lock);

3191         /* Skip zones that are on the way down or not yet up */
3192         status = zone_status_get(zp);
3193         if (status < ZONE_IS_READY || status >= ZONE_IS_DOWN) {
3194             /* For all practical purposes the zone doesn't exist. */
3195             mutex_exit(&zp->zone_lock);
3196             continue;
3197         }

3199         /*
3200          * Update the 10 second moving average data in zone_loadavg.
3201          */
3202         lavg = &zp->zone_loadavg;

3204         zone_total = zp->zone_untime + zp->zone_stime + zp->zone_wtime;
3205         scalehrtime(&zone_total);

3207         /* The zone_total should always be increasing. */
3208         lavg->lg_loads[lavg->lg_cur] = (zone_total > lavg->lg_total) ?
3209             zone_total - lavg->lg_total : 0;
3210         lavg->lg_cur = (lavg->lg_cur + 1) % S_LOADAVG_SZ;
3211         /* lg_total holds the prev. 1 sec. total */
3212         lavg->lg_total = zone_total;

3214         /*
3215          * To simplify the calculation, we don't calculate the load avg.
3216          * until the zone has been up for at least 10 seconds and our
3217          * moving average is thus full.
3218          */
3219         if ((lavg->lg_len + 1) < S_LOADAVG_SZ) {
3220             lavg->lg_len++;
3221             mutex_exit(&zp->zone_lock);
3222             continue;
3223         }

3225         /* Now calculate the lmin, 5min, 15 min load avg. */
3226         hr_avg = 0;
3227         for (i = 0; i < S_LOADAVG_SZ; i++)
3228             hr_avg += lavg->lg_loads[i];
3229         hr_avg = hr_avg / S_LOADAVG_SZ;
3230         nrun = hr_avg / (NANOSEC / LGRP_LOADAVG_IN_THREAD_MAX);

3232         /* Compute load avg. See comment in calcloadavg() */
3233         for (i = 0; i < 3; i++) {
3234             q = (zp->zone_hp_avenrun[i] >> 16) << 7;
3235             r = (zp->zone_hp_avenrun[i] & 0xffff) << 7;
3236             zp->zone_hp_avenrun[i] +=
3237                 ((nrun - q) * f[i] - ((r * f[i]) >> 16)) >> 4;

3239             /* avenrun[] can only hold 31 bits of load avg. */
3240             if (zp->zone_hp_avenrun[i] <
3241                 ((uint64_t)1<<(31+16-FSHIFT)))
3242                 zp->zone_avenrun[i] = (int32_t)
3243                     (zp->zone_hp_avenrun[i] >> (16 - FSHIFT));
3244             else
3245                 zp->zone_avenrun[i] = 0x7fffffff;
3246         }

3248         mutex_exit(&zp->zone_lock);
3249     }
3250     mutex_exit(&zonehash_lock);
3251 }

```

```

3253 /*
3254  * Get the number of cpus visible to this zone. The system-wide global
3255  * 'ncpus' is returned if pools are disabled, the caller is in the
3256  * global zone, or a NULL zone argument is passed in.
3257  */
3258 int
3259 zone_ncpus_get(zone_t *zone)
3260 {
3261     int myncpus = zone == NULL ? 0 : zone->zone_ncpus;

3263     return (myncpus != 0 ? myncpus : ncpus);
3264 }

3266 /*
3267  * Get the number of online cpus visible to this zone. The system-wide
3268  * global 'ncpus_online' is returned if pools are disabled, the caller
3269  * is in the global zone, or a NULL zone argument is passed in.
3270  */
3271 int
3272 zone_ncpus_online_get(zone_t *zone)
3273 {
3274     int myncpus_online = zone == NULL ? 0 : zone->zone_ncpus_online;

3276     return (myncpus_online != 0 ? myncpus_online : ncpus_online);
3277 }

3279 /*
3280  * Return the pool to which the zone is currently bound.
3281  */
3282 pool_t *
3283 zone_pool_get(zone_t *zone)
3284 {
3285     ASSERT(pool_lock_held());

3287     return (zone->zone_pool);
3288 }

3290 /*
3291  * Set the zone's pool pointer and update the zone's visibility to match
3292  * the resources in the new pool.
3293  */
3294 void
3295 zone_pool_set(zone_t *zone, pool_t *pool)
3296 {
3297     ASSERT(pool_lock_held());
3298     ASSERT(MUTEX_HELD(&cpu_lock));

3300     zone->zone_pool = pool;
3301     zone_pset_set(zone, pool->pool_pset->pset_id);
3302 }

3304 /*
3305  * Return the cached value of the id of the processor set to which the
3306  * zone is currently bound. The value will be ZONE_PS_INVALID if the pools
3307  * facility is disabled.
3308  */
3309 psetid_t
3310 zone_pset_get(zone_t *zone)
3311 {
3312     ASSERT(MUTEX_HELD(&cpu_lock));

3314     return (zone->zone_psetid);
3315 }

3317 /*
3318  * Set the cached value of the id of the processor set to which the zone

```

```

3319 * is currently bound. Also update the zone's visibility to match the
3320 * resources in the new processor set.
3321 */
3322 void
3323 zone_pset_set(zone_t *zone, psetid_t newpsetid)
3324 {
3325     psetid_t oldpsetid;
3326
3327     ASSERT(MUTEX_HELD(&cpu_lock));
3328     oldpsetid = zone_pset_get(zone);
3329
3330     if (oldpsetid == newpsetid)
3331         return;
3332     /*
3333      * Global zone sees all.
3334      */
3335     if (zone != global_zone) {
3336         zone->zone_psetid = newpsetid;
3337         if (newpsetid != ZONE_PS_INVALID)
3338             pool_pset_visibility_add(newpsetid, zone);
3339         if (oldpsetid != ZONE_PS_INVALID)
3340             pool_pset_visibility_remove(oldpsetid, zone);
3341     }
3342     /*
3343      * Disabling pools, so we should start using the global values
3344      * for ncpus and ncpus_online.
3345      */
3346     if (newpsetid == ZONE_PS_INVALID) {
3347         zone->zone_ncpus = 0;
3348         zone->zone_ncpus_online = 0;
3349     }
3350 }
3351
3352 /*
3353  * Walk the list of active zones and issue the provided callback for
3354  * each of them.
3355  *
3356  * Caller must not be holding any locks that may be acquired under
3357  * zonehash_lock. See comment at the beginning of the file for a list of
3358  * common locks and their interactions with zones.
3359  */
3360 int
3361 zone_walk(int (*cb)(zone_t *, void *), void *data)
3362 {
3363     zone_t *zone;
3364     int ret = 0;
3365     zone_status_t status;
3366
3367     mutex_enter(&zonehash_lock);
3368     for (zone = list_head(&zone_active); zone != NULL;
3369          zone = list_next(&zone_active, zone)) {
3370         /*
3371          * Skip zones that shouldn't be externally visible.
3372          */
3373         status = zone_status_get(zone);
3374         if (status < ZONE_IS_READY || status > ZONE_IS_DOWN)
3375             continue;
3376         /*
3377          * Bail immediately if any callback invocation returns a
3378          * non-zero value.
3379          */
3380         ret = (*cb)(zone, data);
3381         if (ret != 0)
3382             break;
3383     }
3384     mutex_exit(&zonehash_lock);

```

```

3385         return (ret);
3386     }
3387
3388     static int
3389     zone_set_root(zone_t *zone, const char *upath)
3390     {
3391         vnode_t *vp;
3392         int trycount;
3393         int error = 0;
3394         char *path;
3395         struct pathname upn, pn;
3396         size_t pathlen;
3397
3398         if ((error = pn_get((char *)upath, UIO_USERSPACE, &upn)) != 0)
3399             return (error);
3400
3401         pn_alloc(&pn);
3402
3403         /* prevent infinite loop */
3404         trycount = 10;
3405         for (;;) {
3406             if (--trycount <= 0) {
3407                 error = ESTALE;
3408                 goto out;
3409             }
3410
3411             if ((error = lookupppn(&upn, &pn, FOLLOW, NULLVPP, &vp)) == 0) {
3412                 /*
3413                  * VOP_ACCESS() may cover 'vp' with a new
3414                  * filesystem, if 'vp' is an autoFS vnode.
3415                  * Get the new 'vp' if so.
3416                  */
3417                 if ((error =
3418                      VOP_ACCESS(vp, VEXEC, 0, CRED(), NULL)) == 0 &&
3419                     (!vn_ismntpt(vp) ||
3420                      (error = traverse(&vp)) == 0)) {
3421                     pathlen = pn.pn_pathlen + 2;
3422                     path = kmem_alloc(pathlen, KM_SLEEP);
3423                     (void) strncpy(path, pn.pn_path,
3424                                   pn.pn_pathlen + 1);
3425                     path[pathlen - 2] = '/';
3426                     path[pathlen - 1] = '\0';
3427                     pn_free(&pn);
3428                     pn_free(&upn);
3429
3430                     /* Success! */
3431                     break;
3432                 }
3433                 VN_RELE(vp);
3434             }
3435             if (error != ESTALE)
3436                 goto out;
3437         }
3438
3439         ASSERT(error == 0);
3440         zone->zone_rootvp = vp;          /* we hold a reference to vp */
3441         zone->zone_rootpath = path;
3442         zone->zone_rootpathlen = pathlen;
3443         if (pathlen > 5 && strcmp(path + pathlen - 5, "/lu/") == 0)
3444             zone->zone_flags |= ZF_IS_SCRATCH;
3445         return (0);
3446     out:
3447         pn_free(&pn);
3448         pn_free(&upn);
3449         return (error);
3450     }

```

```

3451 }
3453 #define isalnum(c)      (((c) >= '0' && (c) <= '9') || \
3454                       ((c) >= 'a' && (c) <= 'z') || \
3455                       ((c) >= 'A' && (c) <= 'Z'))
3457 static int
3458 zone_set_name(zone_t *zone, const char *uname)
3459 {
3460     char *kname = kmem_zalloc(ZONENAME_MAX, KM_SLEEP);
3461     size_t len;
3462     int i, err;
3464     if ((err = copyinstr(uname, kname, ZONENAME_MAX, &len)) != 0) {
3465         kmem_free(kname, ZONENAME_MAX);
3466         return (err); /* EFAULT or ENAMETOOLONG */
3467     }
3469     /* must be less than ZONENAME_MAX */
3470     if (len == ZONENAME_MAX && kname[ZONENAME_MAX - 1] != '\0') {
3471         kmem_free(kname, ZONENAME_MAX);
3472         return (EINVAL);
3473     }
3475     /*
3476      * Name must start with an alphanumeric and must contain only
3477      * alphanumerics, '-', '_' and '.'.
3478      */
3479     if (!isalnum(kname[0])) {
3480         kmem_free(kname, ZONENAME_MAX);
3481         return (EINVAL);
3482     }
3483     for (i = 1; i < len - 1; i++) {
3484         if (!isalnum(kname[i]) && kname[i] != '-' && kname[i] != '_' &&
3485             kname[i] != '.') {
3486             kmem_free(kname, ZONENAME_MAX);
3487             return (EINVAL);
3488         }
3489     }
3491     zone->zone_name = kname;
3492     return (0);
3493 }
3495 /*
3496  * Gets the 32-bit hostid of the specified zone as an unsigned int. If 'zonep'
3497  * is NULL or it points to a zone with no hostid emulation, then the machine's
3498  * hostid (i.e., the global zone's hostid) is returned. This function returns
3499  * zero if neither the zone nor the host machine (global zone) have hostids. It
3500  * returns HW_INVALID_HOSTID if the function attempts to return the machine's
3501  * hostid and the machine's hostid is invalid.
3502  */
3503 uint32_t
3504 zone_get_hostid(zone_t *zonep)
3505 {
3506     unsigned long machine_hostid;
3508     if (zonep == NULL || zonep->zone_hostid == HW_INVALID_HOSTID) {
3509         if (ddi_strtoul(hw_serial, NULL, 10, &machine_hostid) != 0)
3510             return (HW_INVALID_HOSTID);
3511         return ((uint32_t)machine_hostid);
3512     }
3513     return (zonep->zone_hostid);
3514 }
3516 /*

```

```

3517  * Similar to thread_create(), but makes sure the thread is in the appropriate
3518  * zone's zsched process (curproc->p_zone->zone_zsched) before returning.
3519  */
3520 /*ARGSUSED*/
3521 kthread_t *
3522 zthread_create(
3523     caddr_t stk,
3524     size_t stksize,
3525     void (*proc)(),
3526     void *arg,
3527     size_t len,
3528     pri_t pri)
3529 {
3530     kthread_t *t;
3531     zone_t *zone = curproc->p_zone;
3532     proc_t *pp = zone->zone_zsched;
3534     zone_hold(zone); /* Reference to be dropped when thread exits */
3536     /*
3537      * No-one should be trying to create threads if the zone is shutting
3538      * down and there aren't any kernel threads around. See comment
3539      * in zthread_exit().
3540      */
3541     ASSERT(!(zone->zone_kthreads == NULL &&
3542         zone_status_get(zone) >= ZONE_IS_EMPTY));
3543     /*
3544      * Create a thread, but don't let it run until we've finished setting
3545      * things up.
3546      */
3547     t = thread_create(stk, stksize, proc, arg, len, pp, TS_STOPPED, pri);
3548     ASSERT(t->t_forw == NULL);
3549     mutex_enter(&zone_status_lock);
3550     if (zone->zone_kthreads == NULL) {
3551         t->t_forw = t->t_back = t;
3552     } else {
3553         kthread_t *tx = zone->zone_kthreads;
3555         t->t_forw = tx;
3556         t->t_back = tx->t_back;
3557         tx->t_back->t_forw = t;
3558         tx->t_back = t;
3559     }
3560     zone->zone_kthreads = t;
3561     mutex_exit(&zone_status_lock);
3563     mutex_enter(&pp->p_lock);
3564     t->t_proc_flag |= TP_ZTHREAD;
3565     project_rele(t->t_proj);
3566     t->t_proj = project_hold(pp->p_task->tk_proj);
3568     /*
3569      * Setup complete, let it run.
3570      */
3571     thread_lock(t);
3572     t->t_schedflag |= TS_ALLSTART;
3573     setrun_locked(t);
3574     thread_unlock(t);
3576     mutex_exit(&pp->p_lock);
3578     return (t);
3579 }
3581 /*
3582  * Similar to thread_exit(). Must be called by threads created via

```

```

3583 * zthread_exit().
3584 */
3585 void
3586 zthread_exit(void)
3587 {
3588     kthread_t *t = curthread;
3589     proc_t *pp = curproc;
3590     zone_t *zone = pp->p_zone;
3592     mutex_enter(&zone_status_lock);
3594     /*
3595      * Reparent to p0
3596      */
3597     kpreempt_disable();
3598     mutex_enter(&pp->p_lock);
3599     t->t_proc_flag &= -TP_ZTHREAD;
3600     t->t_procp = &p0;
3601     hat_thread_exit(t);
3602     mutex_exit(&pp->p_lock);
3603     kpreempt_enable();
3605     if (t->t_back == t) {
3606         ASSERT(t->t_forw == t);
3607         /*
3608          * If the zone is empty, once the thread count
3609          * goes to zero no further kernel threads can be
3610          * created. This is because if the creator is a process
3611          * in the zone, then it must have exited before the zone
3612          * state could be set to ZONE_IS_EMPTY.
3613          * Otherwise, if the creator is a kernel thread in the
3614          * zone, the thread count is non-zero.
3615          *
3616          * This really means that non-zone kernel threads should
3617          * not create zone kernel threads.
3618          */
3619         zone->zone_kthreads = NULL;
3620         if (zone_status_get(zone) == ZONE_IS_EMPTY) {
3621             zone_status_set(zone, ZONE_IS_DOWN);
3622             /*
3623              * Remove any CPU caps on this zone.
3624              */
3625             cpucaps_zone_remove(zone);
3626         }
3627     } else {
3628         t->t_forw->t_back = t->t_back;
3629         t->t_back->t_forw = t->t_forw;
3630         if (zone->zone_kthreads == t)
3631             zone->zone_kthreads = t->t_forw;
3632     }
3633     mutex_exit(&zone_status_lock);
3634     zone_rele(zone);
3635     thread_exit();
3636     /* NOTREACHED */
3637 }
3639 static void
3640 zone_chdir(vnode_t *vp, vnode_t **vpp, proc_t *pp)
3641 {
3642     vnode_t *oldvp;
3644     /* we're going to hold a reference here to the directory */
3645     VN_HOLD(vp);
3647     /* update abs cwd/root path see c2/audit.c */
3648     if (AU_AUDITING())

```

```

3649         audit_chdirec(vp, vpp);
3651         mutex_enter(&pp->p_lock);
3652         oldvp = *vpp;
3653         *vpp = vp;
3654         mutex_exit(&pp->p_lock);
3655         if (oldvp != NULL)
3656             VN_RELE(oldvp);
3657     }
3659     /*
3660      * Convert an rctl value represented by an nvlist_t into an rctl_val_t.
3661      */
3662     static int
3663     nvlist2rctlval(nvlist_t *nvl, rctl_val_t *rv)
3664     {
3665         nvpair_t *nvp = NULL;
3666         boolean_t priv_set = B_FALSE;
3667         boolean_t limit_set = B_FALSE;
3668         boolean_t action_set = B_FALSE;
3670         while ((nvp = nvlist_next_nvpair(nvl, nvp)) != NULL) {
3671             const char *name;
3672             uint64_t ui64;
3674             name = nvpair_name(nvp);
3675             if (nvpair_type(nvp) != DATA_TYPE_UINT64)
3676                 return (EINVAL);
3677             (void) nvpair_value_uint64(nvp, &ui64);
3678             if (strcmp(name, "privilege") == 0) {
3679                 /*
3680                  * Currently only privileged values are allowed, but
3681                  * this may change in the future.
3682                  */
3683                 if (ui64 != RCPRIV_PRIVILEGED)
3684                     return (EINVAL);
3685                 rv->rcv_privilege = ui64;
3686                 priv_set = B_TRUE;
3687             } else if (strcmp(name, "limit") == 0) {
3688                 rv->rcv_value = ui64;
3689                 limit_set = B_TRUE;
3690             } else if (strcmp(name, "action") == 0) {
3691                 if (ui64 != RCTL_LOCAL_NOACTION &&
3692                     ui64 != RCTL_LOCAL_DENY)
3693                     return (EINVAL);
3694                 rv->rcv_flagaction = ui64;
3695                 action_set = B_TRUE;
3696             } else {
3697                 return (EINVAL);
3698             }
3699         }
3701         if (!(priv_set && limit_set && action_set))
3702             return (EINVAL);
3703         rv->rcv_action_signal = 0;
3704         rv->rcv_action_recipient = NULL;
3705         rv->rcv_action_recip_pid = -1;
3706         rv->rcv_firing_time = 0;
3708         return (0);
3709     }
3711     /*
3712      * Non-global zone version of start_init.
3713      */
3714     void

```



```

3715 zone_start_init(void)
3716 {
3717     proc_t *p = ttoproc(curthread);
3718     zone_t *z = p->p_zone;
3720     ASSERT(!INGLOBALZONE(curproc));
3722     /*
3723      * For all purposes (ZONE_ATTR_INITPID and restart_init),
3724      * storing just the pid of init is sufficient.
3725      */
3726     z->zone_proc_initpid = p->p_pid;
3728     /*
3729      * We maintain zone_boot_err so that we can return the cause of the
3730      * failure back to the caller of the zone_boot syscall.
3731      */
3732     p->p_zone->zone_boot_err = start_init_common();
3734     /*
3735      * We will prevent booting zones from becoming running zones if the
3736      * global zone is shutting down.
3737      */
3738     mutex_enter(&zone_status_lock);
3739     if (z->zone_boot_err != 0 || zone_status_get(global_zone) >=
3740         ZONE_IS_SHUTTING_DOWN) {
3741         /*
3742          * Make sure we are still in the booting state-- we could have
3743          * raced and already be shutting down, or even further along.
3744          */
3745         if (zone_status_get(z) == ZONE_IS_BOOTING) {
3746             zone_status_set(z, ZONE_IS_SHUTTING_DOWN);
3747         }
3748         mutex_exit(&zone_status_lock);
3749         /* It's gone bad, dispose of the process */
3750         if (proc_exit(CLD_EXITED, z->zone_boot_err) != 0) {
3751             mutex_enter(&p->p_lock);
3752             ASSERT(p->p_flag & SEXITLWPS);
3753             lwp_exit();
3754         }
3755     } else {
3756         if (zone_status_get(z) == ZONE_IS_BOOTING)
3757             zone_status_set(z, ZONE_IS_RUNNING);
3758         mutex_exit(&zone_status_lock);
3759         /* cause the process to return to userland. */
3760         lwp_rtt();
3761     }
3762 }
3764 struct zsched_arg {
3765     zone_t *zone;
3766     nvlist_t *nvlist;
3767 };
3769 /*
3770 * Per-zone "sched" workalike. The similarity to "sched" doesn't have
3771 * anything to do with scheduling, but rather with the fact that
3772 * per-zone kernel threads are parented to zsched, just like regular
3773 * kernel threads are parented to sched (p0).
3774 *
3775 * zsched is also responsible for launching init for the zone.
3776 */
3777 static void
3778 zsched(void *arg)
3779 {
3780     struct zsched_arg *za = arg;

```

```

3781     proc_t *pp = curproc;
3782     proc_t *initp = proc_init;
3783     zone_t *zone = za->zone;
3784     cred_t *cr, *oldcred;
3785     rctl_set_t *set;
3786     rctl_alloc_gp_t *gp;
3787     contract_t *ct = NULL;
3788     task_t *tk, *oldtk;
3789     rctl_entity_p_t e;
3790     kproject_t *pj;
3792     nvlist_t *nvl = za->nvlist;
3793     nvpair_t *nvp = NULL;
3795     bcopy("zsched", PTOU(pp)->u_psargs, sizeof("zsched"));
3796     bcopy("zsched", PTOU(pp)->u_comm, sizeof("zsched"));
3797     PTOU(pp)->u_argc = 0;
3798     PTOU(pp)->u_argv = NULL;
3799     PTOU(pp)->u_envp = NULL;
3800     closeall(P_FINFO(pp));
3802     /*
3803      * We are this zone's "zsched" process. As the zone isn't generally
3804      * visible yet we don't need to grab any locks before initializing its
3805      * zone_proc pointer.
3806      */
3807     zone_hold(zone); /* this hold is released by zone_destroy() */
3808     zone->zone_zsched = pp;
3809     mutex_enter(&pp->p_lock);
3810     pp->p_zone = zone;
3811     mutex_exit(&pp->p_lock);
3813     /*
3814      * Disassociate process from its 'parent'; parent ourselves to init
3815      * (pid 1) and change other values as needed.
3816      */
3817     sess_create();
3819     mutex_enter(&pidlock);
3820     proc_detach(pp);
3821     pp->p_ppid = 1;
3822     pp->p_flag |= SZONETOP;
3823     pp->p_ancpid = 1;
3824     pp->p_parent = initp;
3825     pp->p_psibling = NULL;
3826     if (initp->p_child)
3827         initp->p_child->p_psibling = pp;
3828     pp->p_sibling = initp->p_child;
3829     initp->p_child = pp;
3831     /* Decrement what newproc() incremented. */
3832     upcount_dec(crgetruid(CRED()), GLOBAL_ZONEID);
3833     /*
3834      * Our credentials are about to become kcred-like, so we don't care
3835      * about the caller's ruid.
3836      */
3837     upcount_inc(crgetruid(kcred), zone->zone_id);
3838     mutex_exit(&pidlock);
3840     /*
3841      * getting out of global zone, so decrement lwp and process counts
3842      */
3843     pj = pp->p_task->tk_proj;
3844     mutex_enter(&global_zone->zone_nlwps_lock);
3845     pj->kpj_nlwps -= pp->p_lwpcnt;
3846     global_zone->zone_nlwps -= pp->p_lwpcnt;

```

```

3847     pj->kpj_nprocs--;
3848     global_zone->zone_nprocs--;
3849     mutex_exit(&global_zone->zone_nlwps_lock);

3851     /*
3852      * Decrement locked memory counts on old zone and project.
3853      */
3854     mutex_enter(&global_zone->zone_mem_lock);
3855     global_zone->zone_locked_mem -= pp->p_locked_mem;
3856     pj->kpj_data.kpd_locked_mem -= pp->p_locked_mem;
3857     mutex_exit(&global_zone->zone_mem_lock);

3859     /*
3860      * Create and join a new task in project '0' of this zone.
3861      *
3862      * We don't need to call holdlwps() since we know we're the only lwp in
3863      * this process.
3864      *
3865      * task_join() returns with p_lock held.
3866      */
3867     tk = task_create(0, zone);
3868     mutex_enter(&cpu_lock);
3869     oldtk = task_join(tk, 0);

3871     pj = pp->p_task->tk_proj;

3873     mutex_enter(&zone->zone_mem_lock);
3874     zone->zone_locked_mem += pp->p_locked_mem;
3875     pj->kpj_data.kpd_locked_mem += pp->p_locked_mem;
3876     mutex_exit(&zone->zone_mem_lock);

3878     /*
3879      * add lwp and process counts to zsched's zone, and increment
3880      * project's task and process count due to the task created in
3881      * the above task_create.
3882      */
3883     mutex_enter(&zone->zone_nlwps_lock);
3884     pj->kpj_nlwps += pp->p_lwpcnt;
3885     pj->kpj_nprocs += 1;
3886     zone->zone_nlwps += pp->p_lwpcnt;
3887     pj->kpj_nprocs++;
3888     zone->zone_nprocs++;
3889     mutex_exit(&zone->zone_nlwps_lock);

3891     mutex_exit(&curproc->p_lock);
3892     mutex_exit(&cpu_lock);
3893     task_rele(oldtk);

3895     /*
3896      * The process was created by a process in the global zone, hence the
3897      * credentials are wrong. We might as well have kcred-ish credentials.
3898      */
3899     cr = zone->zone_kcred;
3900     crhold(cr);
3901     mutex_enter(&pp->p_crlock);
3902     oldcred = pp->p_cred;
3903     pp->p_cred = cr;
3904     mutex_exit(&pp->p_crlock);
3905     crfree(oldcred);

3907     /*
3908      * Hold credentials again (for thread)
3909      */
3910     crhold(cr);

3912     /*

```

```

3913      * p_lwpcnt can't change since this is a kernel process.
3914      */
3915     crset(pp, cr);

3917     /*
3918      * Chroot
3919      */
3920     zone_chdir(zone->zone_rootvp, &PTOU(pp)->u_cdir, pp);
3921     zone_chdir(zone->zone_rootvp, &PTOU(pp)->u_rdir, pp);

3923     /*
3924      * Initialize zone's rctl set.
3925      */
3926     set = rctl_set_create();
3927     gp = rctl_set_init_prealloc(RCENTITY_ZONE);
3928     mutex_enter(&pp->p_lock);
3929     e.rcep_p.zone = zone;
3930     e.rcep_t = RCENTITY_ZONE;
3931     zone->zone_rctls = rctl_set_init(RCENTITY_ZONE, pp, &e, set, gp);
3932     mutex_exit(&pp->p_lock);
3933     rctl_prealloc_destroy(gp);

3935     /*
3936      * Apply the rctls passed in to zone_create(). This is basically a list
3937      * assignment: all of the old values are removed and the new ones
3938      * inserted. That is, if an empty list is passed in, all values are
3939      * removed.
3940      */
3941     while ((nvp = nvlist_next_nvpair(nvl, nvp)) != NULL) {
3942         rctl_dict_entry_t *rde;
3943         rctl_hdl_t hndl;
3944         char *name;
3945         nvlist_t **nvlarray;
3946         uint_t i, nelem;
3947         int error; /* For ASSERT()s */

3949         name = nvpair_name(nvp);
3950         hndl = rctl_hdl_lookup(name);
3951         ASSERT(hndl != -1);
3952         rde = rctl_dict_lookup_hdl(hndl);
3953         ASSERT(rde != NULL);

3955         for (; /* ever */; ) {
3956             rctl_val_t oval;

3958             mutex_enter(&pp->p_lock);
3959             error = rctl_local_get(hndl, NULL, &oval, pp);
3960             mutex_exit(&pp->p_lock);
3961             ASSERT(error == 0); /* Can't fail for RCTL_FIRST */
3962             ASSERT(oval.rcv_privilege != RCPRIV_BASIC);
3963             if (oval.rcv_privilege == RCPRIV_SYSTEM)
3964                 break;
3965             mutex_enter(&pp->p_lock);
3966             error = rctl_local_delete(hndl, &oval, pp);
3967             mutex_exit(&pp->p_lock);
3968             ASSERT(error == 0);
3969         }
3970         error = nvpair_value_nvlist_array(nvp, &nvlarray, &nelem);
3971         ASSERT(error == 0);
3972         for (i = 0; i < nelem; i++) {
3973             rctl_val_t *nvalp;

3975             nvalp = kmem_cache_alloc(rctl_val_cache, KM_SLEEP);
3976             error = nvlist2rctlval(nvlarray[i], nvalp);
3977             ASSERT(error == 0);
3978             /*

```

```

3979     * rctl_local_insert can fail if the value being
3980     * inserted is a duplicate; this is OK.
3981     */
3982     mutex_enter(&pp->p_lock);
3983     if (rctl_local_insert(hndl, nvalp, pp) != 0)
3984         kmem_cache_free(rctl_val_cache, nvalp);
3985     mutex_exit(&pp->p_lock);
3986 }
3987
3989 #endif /* ! codereview */
3990 /*
3991  * Tell the world that we're done setting up.
3992  */
3993  * At this point we want to set the zone status to ZONE_IS_INITIALIZED
3994  * and atomically set the zone's processor set visibility. Once
3995  * we drop pool_lock() this zone will automatically get updated
3996  * to reflect any future changes to the pools configuration.
3997  */
3998  * Note that after we drop the locks below (zonehash_lock in
3999  * particular) other operations such as a zone_getattr call can
4000  * now proceed and observe the zone. That is the reason for doing a
4001  * state transition to the INITIALIZED state.
4002  */
4003  pool_lock();
4004  mutex_enter(&cpu_lock);
4005  mutex_enter(&zonehash_lock);
4006  zone_uniqid(zone);
4007  zone_zsd_configure(zone);
4008  if (pool_state == POOL_ENABLED)
4009      zone_pset_set(zone, pool_default->pool_pset->pset_id);
4010  mutex_enter(&zone_status_lock);
4011  ASSERT(zone_status_get(zone) == ZONE_IS_UNINITIALIZED);
4012  zone_status_set(zone, ZONE_IS_INITIALIZED);
4013  mutex_exit(&zone_status_lock);
4014  mutex_exit(&zonehash_lock);
4015  mutex_exit(&cpu_lock);
4016  pool_unlock();
4017
4018  /* Now call the create callback for this key */
4019  zsd_apply_all_keys(zsd_apply_create, zone);
4020
4021  /* The callbacks are complete. Mark ZONE_IS_READY */
4022  mutex_enter(&zone_status_lock);
4023  ASSERT(zone_status_get(zone) == ZONE_IS_INITIALIZED);
4024  zone_status_set(zone, ZONE_IS_READY);
4025  mutex_exit(&zone_status_lock);
4026
4027  /*
4028  * Once we see the zone transition to the ZONE_IS_BOOTING state,
4029  * we launch init, and set the state to running.
4030  */
4031  zone_status_wait_cpr(zone, ZONE_IS_BOOTING, "zsched");
4032
4033  if (zone_status_get(zone) == ZONE_IS_BOOTING) {
4034      id_t cid;
4035
4036      /*
4037       * Ok, this is a little complicated. We need to grab the
4038       * zone's pool's scheduling class ID; note that by now, we
4039       * are already bound to a pool if we need to be (zoneadm
4040       * will have done that to us while we're in the READY
4041       * state). *But* the scheduling class for the zone's 'init'
4042       * must be explicitly passed to newproc, which doesn't
4043       * respect pool bindings.
4044       */

```

```

4045     * We hold the pool_lock across the call to newproc() to
4046     * close the obvious race: the pool's scheduling class
4047     * could change before we manage to create the LWP with
4048     * classid 'cid'.
4049     */
4050     pool_lock();
4051     if (zone->zone_defaultcid > 0)
4052         cid = zone->zone_defaultcid;
4053     else
4054         cid = pool_get_class(zone->zone_pool);
4055     if (cid == -1)
4056         cid = defaultcid;
4057
4058     /*
4059     * If this fails, zone_boot will ultimately fail. The
4060     * state of the zone will be set to SHUTTING_DOWN-- userland
4061     * will have to tear down the zone, and fail, or try again.
4062     */
4063     if ((zone->zone_boot_err = newproc(zone_start_init, NULL, cid,
4064         minclsyspri - 1, &ct, 0)) != 0) {
4065         mutex_enter(&zone_status_lock);
4066         zone_status_set(zone, ZONE_IS_SHUTTING_DOWN);
4067         mutex_exit(&zone_status_lock);
4068     } else {
4069         zone->zone_boot_time = gethrestime_sec();
4070     }
4071
4072     pool_unlock();
4073 }
4074
4075 /*
4076  * Wait for zone_destroy() to be called. This is what we spend
4077  * most of our life doing.
4078  */
4079  zone_status_wait_cpr(zone, ZONE_IS_DYING, "zsched");
4080
4081  if (ct)
4082      /*
4083       * At this point the process contract should be empty.
4084       * (Though if it isn't, it's not the end of the world.)
4085       */
4086      VERIFY(contract_abandon(ct, curproc, B_TRUE) == 0);
4087
4088  /*
4089  * Allow kcred to be freed when all referring processes
4090  * (including this one) go away. We can't just do this in
4091  * zone_free because we need to wait for the zone_cred_ref to
4092  * drop to 0 before calling zone_free, and the existence of
4093  * zone_kcred will prevent that. Thus, we call crfree here to
4094  * balance the crdup in zone_create. The crhold calls earlier
4095  * in zsched will be dropped when the thread and process exit.
4096  */
4097  crfree(zone->zone_kcred);
4098  zone->zone_kcred = NULL;
4099
4100  exit(CLD_EXITED, 0);
4101 }
4102
4103 /*
4104  * Helper function to determine if there are any submounts of the
4105  * provided path. Used to make sure the zone doesn't "inherit" any
4106  * mounts from before it is created.
4107  */
4108  static uint_t
4109  zone_mount_count(const char *rootpath)
4110  {

```

```

4111     vfs_t *vfsp;
4112     uint_t count = 0;
4113     size_t rootpathlen = strlen(rootpath);

4115     /*
4116      * Holding zonehash_lock prevents race conditions with
4117      * vfs_list_add()/vfs_list_remove() since we serialize with
4118      * zone_find_by_path().
4119      */
4120     ASSERT(MUTEX_HELD(&zonehash_lock));
4121     /*
4122      * The rootpath must end with a '/'
4123      */
4124     ASSERT(rootpath[rootpathlen - 1] == '/');

4126     /*
4127      * This intentionally does not count the rootpath itself if that
4128      * happens to be a mount point.
4129      */
4130     vfs_list_read_lock();
4131     vfsp = rootvfs;
4132     do {
4133         if (strncmp(rootpath, refstr_value(vfsp->vfs_mntpt),
4134                 rootpathlen) == 0)
4135             count++;
4136         vfsp = vfsp->vfs_next;
4137     } while (vfsp != rootvfs);
4138     vfs_list_unlock();
4139     return (count);
4140 }

4142 /*
4143  * Helper function to make sure that a zone created on 'rootpath'
4144  * wouldn't end up containing other zones' rootpaths.
4145  */
4146 static boolean_t
4147 zone_is_nested(const char *rootpath)
4148 {
4149     zone_t *zone;
4150     size_t rootpathlen = strlen(rootpath);
4151     size_t len;

4153     ASSERT(MUTEX_HELD(&zonehash_lock));

4155     /*
4156      * zone_set_root() appended '/' and '\0' at the end of rootpath
4157      */
4158     if ((rootpathlen <= 3) && (rootpath[0] == '/') &&
4159         (rootpath[1] == '/') && (rootpath[2] == '\0'))
4160         return (B_TRUE);

4162     for (zone = list_head(&zone_active); zone != NULL;
4163          zone = list_next(&zone_active, zone)) {
4164         if (zone == global_zone)
4165             continue;
4166         len = strlen(zone->zone_rootpath);
4167         if (strncmp(rootpath, zone->zone_rootpath,
4168                 MIN(rootpathlen, len)) == 0)
4169             return (B_TRUE);
4170     }
4171     return (B_FALSE);
4172 }

4174 static int
4175 zone_set_privset(zone_t *zone, const priv_set_t *zone_privs,
4176                 size_t zone_privssz)

```

```

4177 {
4178     priv_set_t *privs;

4180     if (zone_privssz < sizeof (priv_set_t))
4181         return (ENOMEM);

4183     privs = kmem_alloc(sizeof (priv_set_t), KM_SLEEP);

4185     if (copyin(zone_privs, privs, sizeof (priv_set_t))) {
4186         kmem_free(privs, sizeof (priv_set_t));
4187         return (EFAULT);
4188     }

4190     zone->zone_privset = privs;
4191     return (0);
4192 }

4194 /*
4195  * We make creative use of nvlists to pass in rctls from userland. The list is
4196  * a list of the following structures:
4197  *
4198  * (name = rctl_name, value = nvpair_list_array)
4199  *
4200  * Where each element of the nvpair_list_array is of the form:
4201  *
4202  * [(name = "privilege", value = RCPRIV_PRIVILEGED),
4203  *  (name = "limit", value = uint64_t),
4204  *  (name = "action", value = (RCTL_LOCAL_NOACTION || RCTL_LOCAL_DENY))]
4205  */
4206 static int
4207 parse_rctls(caddr_t ubuf, size_t buflen, nvlist_t **nvlp)
4208 {
4209     nvpair_t *nvp = NULL;
4210     nvlist_t *nvl = NULL;
4211     char *kbuf;
4212     int error;
4213     rctl_val_t rv;

4215     *nvlp = NULL;

4217     if (buflen == 0)
4218         return (0);

4220     if ((kbuf = kmem_alloc(buflen, KM_NOSLEEP)) == NULL)
4221         return (ENOMEM);
4222     if (copyin(ubuf, kbuf, buflen)) {
4223         error = EFAULT;
4224         goto out;
4225     }
4226     if (nvlist_unpack(kbuf, buflen, &nvl, KM_SLEEP) != 0) {
4227         /*
4228          * nvl may have been allocated/free'd, but the value set to
4229          * non-NULL, so we reset it here.
4230          */
4231         nvl = NULL;
4232         error = EINVAL;
4233         goto out;
4234     }
4235     while ((nvp = nvlist_next_nvpair(nvl, nvp)) != NULL) {
4236         rctl_dict_entry_t *rde;
4237         rctl_hdl_t hndl;
4238         nvlist_t **nvlarray;
4239         uint_t i, nelem;
4240         char *name;

4242         error = EINVAL;

```

```

4243     name = nvpair_name(nvp);
4244     if (strcmp(nvpair_name(nvp), "zone.", sizeof ("zone.") - 1)
4245         != 0 || nvpair_type(nvp) != DATA_TYPE_NVLIST_ARRAY) {
4246         goto out;
4247     }
4248     if ((hndl = rctl_hdl_lookup(name)) == -1) {
4249         goto out;
4250     }
4251     rde = rctl_dict_lookup_hdl(hndl);
4252     error = nvpair_value_nvlist_array(nvp, &nvlarray, &nelem);
4253     ASSERT(error == 0);
4254     for (i = 0; i < nelem; i++) {
4255         if (error = nvlist2rctlval(nvlarray[i], &rv))
4256             goto out;
4257     }
4258     if (rctl_invalid_value(rde, &rv)) {
4259         error = EINVAL;
4260         goto out;
4261     }
4262 }
4263 error = 0;
4264 *nvlp = nvl;
4265 out:
4266     kmem_free(kbuf, buflen);
4267     if (error && nvl != NULL)
4268         nvlist_free(nvl);
4269     return (error);
4270 }

4272 int
4273 zone_create_error(int er_error, int er_ext, int *er_out)
4274 {
4275     zone_create_error(int er_error, int er_ext, int *er_out) {
4276         if (er_out != NULL) {
4277             if (copyout(&er_ext, er_out, sizeof (int))) {
4278                 return (set_errno(EFAULT));
4279             }
4280         }
4281     }
4282     return (set_errno(er_error));
4281 }

```

unchanged portion omitted

```

4349 /*
4350  * System call to create/initialize a new zone named 'zone_name', rooted
4351  * at 'zone_root', with a zone-wide privilege limit set of 'zone_privs',
4352  * and initialized with the zone-wide rctls described in 'rctlbuf', and
4353  * with labeling set by 'match', 'doi', and 'label'.
4354  *
4355  * If extended error is non-null, we may use it to return more detailed
4356  * error information.
4357  */
4358 static zoneid_t
4359 zone_create(const char *zone_name, const char *zone_root,
4360             const priv_set_t *zone_privs, size_t zone_privssz,
4361             caddr_t rctlbuf, size_t rctlbufsz,
4362             caddr_t zfsbuf, size_t zfsbufsz, int *extended_error,
4363             int match, uint32_t doi, const bslabel_t *label,
4364             int flags)
4365 {
4366     struct zsched_arg zarg;
4367     nvlist_t *rctls = NULL;
4368     proc_t *pp = curproc;
4369     zone_t *zone, *ztmp;
4370     zoneid_t zoneid;
4371     int error;
4372     int error2 = 0;

```

```

4373     char *str;
4374     cred_t *zkcr;
4375     boolean_t insert_label_hash;

4377     if (secpolicy_zone_config(CRED()) != 0)
4378         return (set_errno(EPERM));

4380     /* can't boot zone from within chroot environment */
4381     if (PTOU(pp)->u_rdir != NULL && PTOU(pp)->u_rdir != rootdir)
4382         return (zone_create_error(ENOTSUP, ZE_CHROOTED,
4383             extended_error));

4385     zone = kmem_zalloc(sizeof (zone_t), KM_SLEEP);
4386     zoneid = zone->zone_id = id_alloc(zoneid_space);
4387     zone->zone_status = ZONE_IS_UNINITIALIZED;
4388     zone->zone_pool = pool_default;
4389     zone->zone_pool_mod = gethrtime();
4390     zone->zone_psetid = ZONE_PS_INVALID;
4391     zone->zone_ncpus = 0;
4392     zone->zone_ncpus_online = 0;
4393     zone->zone_restart_init = B_TRUE;
4394     zone->zone_brand = &native_brand;
4395     zone->zone_initname = NULL;
4396     mutex_init(&zone->zone_lock, NULL, MUTEX_DEFAULT, NULL);
4397     mutex_init(&zone->zone_nlws_lock, NULL, MUTEX_DEFAULT, NULL);
4398     mutex_init(&zone->zone_mem_lock, NULL, MUTEX_DEFAULT, NULL);
4399     cv_init(&zone->zone_cv, NULL, CV_DEFAULT, NULL);
4400     list_create(&zone->zone_ref_list, sizeof (zone_ref_t),
4401         offsetof(zone_ref_t, zref_linkage));
4402     list_create(&zone->zone_zsd, sizeof (struct zsd_entry),
4403         offsetof(struct zsd_entry, zsd_linkage));
4404     list_create(&zone->zone_datasets, sizeof (zone_dataset_t),
4405         offsetof(zone_dataset_t, zd_linkage));
4406     list_create(&zone->zone_dl_list, sizeof (zone_dl_t),
4407         offsetof(zone_dl_t, zd_linkage));
4408     rw_init(&zone->zone_mlps.mpl_rwlock, NULL, RW_DEFAULT, NULL);
4409     rw_init(&zone->zone_mntfs_db_lock, NULL, RW_DEFAULT, NULL);

4411     if (flags & ZCF_NET_EXCL) {
4412         zone->zone_flags |= ZF_NET_EXCL;
4413     }

4415     if ((error = zone_set_name(zone, zone_name)) != 0) {
4416         zone_free(zone);
4417         return (zone_create_error(error, 0, extended_error));
4418     }

4420     if ((error = zone_set_root(zone, zone_root)) != 0) {
4421         zone_free(zone);
4422         return (zone_create_error(error, 0, extended_error));
4423     }
4424     if ((error = zone_set_privset(zone, zone_privs, zone_privssz)) != 0) {
4425         zone_free(zone);
4426         return (zone_create_error(error, 0, extended_error));
4427     }

4429     /* initialize node name to be the same as zone name */
4430     zone->zone_nodename = kmem_alloc(_SYS_NMLN, KM_SLEEP);
4431     (void) strncpy(zone->zone_nodename, zone->zone_name, _SYS_NMLN);
4432     zone->zone_nodename[_SYS_NMLN - 1] = '\0';

4434     zone->zone_domain = kmem_alloc(_SYS_NMLN, KM_SLEEP);
4435     zone->zone_domain[0] = '\0';
4436     zone->zone_hostid = HW_INVALID_HOSTID;
4437     zone->zone_shares = 1;
4438     zone->zone_shmmax = 0;

```

```

4439     zone->zone_ipc.ipcq_shmmni = 0;
4440     zone->zone_ipc.ipcq_semmni = 0;
4441     zone->zone_ipc.ipcq_msgmni = 0;
4442     zone->zone_bootargs = NULL;
4443     zone->zone_fs_allowed = NULL;

4445     secflags_zero(&zone0.zone_secflags.psf_lower);
4446     secflags_zero(&zone0.zone_secflags.psf_effective);
4447     secflags_zero(&zone0.zone_secflags.psf_inherit);
4448     secflags_fullset(&zone0.zone_secflags.psf_upper);

4450 #endif /* ! codereview */
4451     zone->zone_initname =
4452         kmem_alloc(strlen(zone_default_initname) + 1, KM_SLEEP);
4453     (void) strcpy(zone->zone_initname, zone_default_initname);
4454     zone->zone_nlwps = 0;
4455     zone->zone_nlwps_ctl = INT_MAX;
4456     zone->zone_nprocs = 0;
4457     zone->zone_nprocs_ctl = INT_MAX;
4458     zone->zone_locked_mem = 0;
4459     zone->zone_locked_mem_ctl = UINT64_MAX;
4460     zone->zone_max_swap = 0;
4461     zone->zone_max_swap_ctl = UINT64_MAX;
4462     zone->zone_max_lofi = 0;
4463     zone->zone_max_lofi_ctl = UINT64_MAX;
4464     zone0.zone_lockedmem_kstat = NULL;
4465     zone0.zone_swapresv_kstat = NULL;

4467     /*
4468      * Zsched initializes the rctls.
4469      */
4470     zone->zone_rctls = NULL;

4472     if ((error = parse_rctls(rctlbuf, rctlbufsz, &rctls)) != 0) {
4473         zone_free(zone);
4474         return (zone_create_error(error, 0, extended_error));
4475     }

4477     if ((error = parse_zfs(zone, zfsbuf, zfsbufsz)) != 0) {
4478         zone_free(zone);
4479         return (set_errno(error));
4480     }

4482     /*
4483      * Read in the trusted system parameters:
4484      * match flag and sensitivity label.
4485      */
4486     zone->zone_match = match;
4487     if (is_system_labeled() && !(zone->zone_flags & ZF_IS_SCRATCH)) {
4488         /* Fail if requested to set doi to anything but system's doi */
4489         if (doi != 0 && doi != default_doi) {
4490             zone_free(zone);
4491             return (set_errno(EINVAL));
4492         }
4493         /* Always apply system's doi to the zone */
4494         error = zone_set_label(zone, label, default_doi);
4495         if (error != 0) {
4496             zone_free(zone);
4497             return (set_errno(error));
4498         }
4499         insert_label_hash = B_TRUE;
4500     } else {
4501         /* all zones get an admin_low label if system is not labeled */
4502         zone->zone_slabel = l_admin_low;
4503         label_hold(l_admin_low);
4504         insert_label_hash = B_FALSE;

```

```

4505     }

4507     /*
4508      * Stop all lwps since that's what normally happens as part of fork().
4509      * This needs to happen before we grab any locks to avoid deadlock
4510      * (another lwp in the process could be waiting for the held lock).
4511      */
4512     if (curthread != pp->p_agenttp && !holdlwps(SHOLDFORK)) {
4513         zone_free(zone);
4514         nvlist_free(rctls);
4515         return (zone_create_error(error, 0, extended_error));
4516     }

4518     if (block_mounts(zone) == 0) {
4519         mutex_enter(&pp->p_lock);
4520         if (curthread != pp->p_agenttp)
4521             continuelwps(pp);
4522         mutex_exit(&pp->p_lock);
4523         zone_free(zone);
4524         nvlist_free(rctls);
4525         return (zone_create_error(error, 0, extended_error));
4526     }

4528     /*
4529      * Set up credential for kernel access. After this, any errors
4530      * should go through the dance in errout rather than calling
4531      * zone_free directly.
4532      */
4533     zone->zone_kcred = crdup(kcred);
4534     crsetzone(zone->zone_kcred, zone);
4535     priv_intersect(zone->zone_privset, &CR_PPRIV(zone->zone_kcred));
4536     priv_intersect(zone->zone_privset, &CR_EPRIV(zone->zone_kcred));
4537     priv_intersect(zone->zone_privset, &CR_IPRIV(zone->zone_kcred));
4538     priv_intersect(zone->zone_privset, &CR_LPRIV(zone->zone_kcred));

4540     mutex_enter(&zonehash_lock);
4541     /*
4542      * Make sure zone doesn't already exist.
4543      *
4544      * If the system and zone are labeled,
4545      * make sure no other zone exists that has the same label.
4546      */
4547     if ((ztmp = zone_find_all_by_name(zone->zone_name)) != NULL ||
4548         (insert_label_hash &&
4549          (ztmp = zone_find_all_by_label(zone->zone_slabel)) != NULL)) {
4550         zone_status_t status;

4552         status = zone_status_get(ztmp);
4553         if (status == ZONE_IS_READY || status == ZONE_IS_RUNNING)
4554             error = EEXIST;
4555         else
4556             error = EBUSY;

4558         if (insert_label_hash)
4559             error2 = ZE_LABELINUSE;

4561         goto errout;
4562     }

4564     /*
4565      * Don't allow zone creations which would cause one zone's rootpath to
4566      * be accessible from that of another (non-global) zone.
4567      */
4568     if (zone_is_nested(zone->zone_rootpath)) {
4569         error = EBUSY;
4570         goto errout;

```

```

4571     }
4573     ASSERT(zonecount != 0); /* check for leaks */
4574     if (zonecount + 1 > maxzones) {
4575         error = ENOMEM;
4576         goto errout;
4577     }
4579     if (zone_mount_count(zone->zone_rootpath) != 0) {
4580         error = EBUSY;
4581         error2 = ZE_AREMOUNTS;
4582         goto errout;
4583     }
4585     /*
4586     * Zone is still incomplete, but we need to drop all locks while
4587     * zsched() initializes this zone's kernel process. We
4588     * optimistically add the zone to the hashtable and associated
4589     * lists so a parallel zone_create() doesn't try to create the
4590     * same zone.
4591     */
4592     zonecount++;
4593     (void) mod_hash_insert(zonehashbyid,
4594         (mod_hash_key_t)(uintptr_t)zone->zone_id,
4595         (mod_hash_val_t)(uintptr_t)zone);
4596     str = kmem_alloc(strlen(zone->zone_name) + 1, KM_SLEEP);
4597     (void) strcpy(str, zone->zone_name);
4598     (void) mod_hash_insert(zonehashbyname, (mod_hash_key_t)str,
4599         (mod_hash_val_t)(uintptr_t)zone);
4600     if (insert_label_hash) {
4601         (void) mod_hash_insert(zonehashbylabel,
4602             (mod_hash_key_t)zone->zone_slab, (mod_hash_val_t)zone);
4603         zone->zone_flags |= ZF_HASHED_LABEL;
4604     }
4606     /*
4607     * Insert into active list. At this point there are no 'hold's
4608     * on the zone, but everyone else knows not to use it, so we can
4609     * continue to use it. zsched() will do a zone_hold() if the
4610     * newproc() is successful.
4611     */
4612     list_insert_tail(&zone_active, zone);
4613     mutex_exit(&zonehash_lock);
4615     zarg.zone = zone;
4616     zarg.nvlist = rctls;
4617     /*
4618     * The process, task, and project rctls are probably wrong;
4619     * we need an interface to get the default values of all rctls,
4620     * and initialize zsched appropriately. I'm not sure that that
4621     * makes much of a difference, though.
4622     */
4623     error = newproc(zsched, (void *)&zarg, syscid, minclsyspri, NULL, 0);
4624     if (error != 0) {
4625         /*
4626         * We need to undo all globally visible state.
4627         */
4628         mutex_enter(&zonehash_lock);
4629         list_remove(&zone_active, zone);
4630         if (zone->zone_flags & ZF_HASHED_LABEL) {
4631             ASSERT(zone->zone_slab != NULL);
4632             (void) mod_hash_destroy(zonehashbylabel,
4633                 (mod_hash_key_t)zone->zone_slab);
4634         }
4635         (void) mod_hash_destroy(zonehashbyname,
4636             (mod_hash_key_t)(uintptr_t)zone->zone_name);

```

```

4637         (void) mod_hash_destroy(zonehashbyid,
4638             (mod_hash_key_t)(uintptr_t)zone->zone_id);
4639         ASSERT(zonecount > 1);
4640         zonecount--;
4641         goto errout;
4642     }
4644     /*
4645     * Zone creation can't fail from now on.
4646     */
4648     /*
4649     * Create zone kstats
4650     */
4651     zone_kstat_create(zone);
4653     /*
4654     * Let the other lwps continue.
4655     */
4656     mutex_enter(&pp->p_lock);
4657     if (curthread != pp->p_agenttp)
4658         continuelwps(pp);
4659     mutex_exit(&pp->p_lock);
4661     /*
4662     * Wait for zsched to finish initializing the zone.
4663     */
4664     zone_status_wait(zone, ZONE_IS_READY);
4665     /*
4666     * The zone is fully visible, so we can let mounts progress.
4667     */
4668     resume_mounts(zone);
4669     nvlist_free(rctls);
4671     return (zoneid);
4673 errout:
4674     mutex_exit(&zonehash_lock);
4675     /*
4676     * Let the other lwps continue.
4677     */
4678     mutex_enter(&pp->p_lock);
4679     if (curthread != pp->p_agenttp)
4680         continuelwps(pp);
4681     mutex_exit(&pp->p_lock);
4683     resume_mounts(zone);
4684     nvlist_free(rctls);
4685     /*
4686     * There is currently one reference to the zone, a cred_ref from
4687     * zone_kcred. To free the zone, we call crfree, which will call
4688     * zone_cred_rele, which will call zone_free.
4689     */
4690     ASSERT(zone->zone_cred_ref == 1);
4691     ASSERT(zone->zone_kcred->cr_ref == 1);
4692     ASSERT(zone->zone_ref == 0);
4693     zkcr = zone->zone_kcred;
4694     zone->zone_kcred = NULL;
4695     crfree(zkcr); /* triggers call to zone_free */
4696     return (zone_create_error(error, error2, extended_error));
4697 }
4699 /*
4700 * Cause the zone to boot. This is pretty simple, since we let zoneadmd do
4701 * the heavy lifting. initname is the path to the program to launch
4702 * at the "top" of the zone; if this is NULL, we use the system default,

```

```

4703 * which is stored at zone_default_initname.
4704 */
4705 static int
4706 zone_boot(zoneid_t zoneid)
4707 {
4708     int err;
4709     zone_t *zone;

4711     if (secpolicy_zone_config(CRED()) != 0)
4712         return (set_errno(EPERM));
4713     if (zoneid < MIN_USERZONEID || zoneid > MAX_ZONEID)
4714         return (set_errno(EINVAL));

4716     mutex_enter(&zonehash_lock);
4717     /*
4718      * Look for zone under hash lock to prevent races with calls to
4719      * zone_shutdown, zone_destroy, etc.
4720      */
4721     if ((zone = zone_find_all_by_id(zoneid)) == NULL) {
4722         mutex_exit(&zonehash_lock);
4723         return (set_errno(EINVAL));
4724     }

4726     mutex_enter(&zone_status_lock);
4727     if (zone_status_get(zone) != ZONE_IS_READY) {
4728         mutex_exit(&zone_status_lock);
4729         mutex_exit(&zonehash_lock);
4730         return (set_errno(EINVAL));
4731     }
4732     zone_status_set(zone, ZONE_IS_BOOTING);
4733     mutex_exit(&zone_status_lock);

4735     zone_hold(zone);          /* so we can use the zone_t later */
4736     mutex_exit(&zonehash_lock);

4738     if (zone_status_wait_sig(zone, ZONE_IS_RUNNING) == 0) {
4739         zone_rele(zone);
4740         return (set_errno(EINTR));
4741     }

4743     /*
4744      * Boot (starting init) might have failed, in which case the zone
4745      * will go to the SHUTTING_DOWN state; an appropriate errno will
4746      * be placed in zone->zone_boot_err, and so we return that.
4747      */
4748     err = zone->zone_boot_err;
4749     zone_rele(zone);
4750     return (err ? set_errno(err) : 0);
4751 }

4753 /*
4754 * Kills all user processes in the zone, waiting for them all to exit
4755 * before returning.
4756 */
4757 static int
4758 zone_empty(zone_t *zone)
4759 {
4760     int waitstatus;

4762     /*
4763      * We need to drop zonehash_lock before killing all
4764      * processes, otherwise we'll deadlock with zone_find_*
4765      * which can be called from the exit path.
4766      */
4767     ASSERT(MUTEX_NOT_HELD(&zonehash_lock));
4768     while ((waitstatus = zone_status_timedwait_sig(zone,

```

```

4769         ddi_get_lbolt() + hz, ZONE_IS_EMPTY)) == -1) {
4770         killall(zone->zone_id);
4771     }
4772     /*
4773      * return EINTR if we were signaled
4774      */
4775     if (waitstatus == 0)
4776         return (EINTR);
4777     return (0);
4778 }

4780 /*
4781 * This function implements the policy for zone visibility.
4782 *
4783 * In standard Solaris, a non-global zone can only see itself.
4784 *
4785 * In Trusted Extensions, a labeled zone can lookup any zone whose label
4786 * it dominates. For this test, the label of the global zone is treated as
4787 * admin_high so it is special-cased instead of being checked for dominance.
4788 *
4789 * Returns true if zone attributes are viewable, false otherwise.
4790 */
4791 static boolean_t
4792 zone_list_access(zone_t *zone)
4793 {
4795     if (curproc->p_zone == global_zone ||
4796         curproc->p_zone == zone) {
4797         return (B_TRUE);
4798     } else if (is_system_labeled() && !(zone->zone_flags & ZF_IS_SCRATCH)) {
4799         bslabel_t *curproc_label;
4800         bslabel_t *zone_label;

4802         curproc_label = label2bslabel(curproc->p_zone->zone_slablel);
4803         zone_label = label2bslabel(zone->zone_slablel);

4805         if (zone->zone_id != GLOBAL_ZONEID &&
4806             bldominates(curproc_label, zone_label)) {
4807             return (B_TRUE);
4808         } else {
4809             return (B_FALSE);
4810         }
4811     } else {
4812         return (B_FALSE);
4813     }
4814 }

4816 /*
4817 * Systemcall to start the zone's halt sequence. By the time this
4818 * function successfully returns, all user processes and kernel threads
4819 * executing in it will have exited, ZSD shutdown callbacks executed,
4820 * and the zone status set to ZONE_IS_DOWN.
4821 *
4822 * It is possible that the call will interrupt itself if the caller is the
4823 * parent of any process running in the zone, and doesn't have SIGCHLD blocked.
4824 */
4825 static int
4826 zone_shutdown(zoneid_t zoneid)
4827 {
4828     int error;
4829     zone_t *zone;
4830     zone_status_t status;

4832     if (secpolicy_zone_config(CRED()) != 0)
4833         return (set_errno(EPERM));
4834     if (zoneid < MIN_USERZONEID || zoneid > MAX_ZONEID)

```



```

4835         return (set_errno(EINVAL));
4837     mutex_enter(&zonehash_lock);
4838     /*
4839      * Look for zone under hash lock to prevent races with other
4840      * calls to zone_shutdown and zone_destroy.
4841      */
4842     if ((zone = zone_find_all_by_id(zoneid)) == NULL) {
4843         mutex_exit(&zonehash_lock);
4844         return (set_errno(EINVAL));
4845     }
4847     /*
4848      * We have to drop zonehash_lock before calling block_mounts.
4849      * Hold the zone so we can continue to use the zone_t.
4850      */
4851     zone_hold(zone);
4852     mutex_exit(&zonehash_lock);
4854     /*
4855      * Block mounts so that VFS_MOUNT() can get an accurate view of
4856      * the zone's status with regards to ZONE_IS_SHUTTING down.
4857      *
4858      * e.g. NFS can fail the mount if it determines that the zone
4859      * has already begun the shutdown sequence.
4860      */
4861     if (block_mounts(zone) == 0) {
4862         zone_rele(zone);
4863         return (set_errno(EINTR));
4864     }
4865
4867     mutex_enter(&zonehash_lock);
4868     mutex_enter(&zone_status_lock);
4869     status = zone_status_get(zone);
4870     /*
4871      * Fail if the zone isn't fully initialized yet.
4872      */
4873     if (status < ZONE_IS_READY) {
4874         mutex_exit(&zone_status_lock);
4875         mutex_exit(&zonehash_lock);
4876         resume_mounts(zone);
4877         zone_rele(zone);
4878         return (set_errno(EINVAL));
4879     }
4880     /*
4881      * If conditions required for zone_shutdown() to return have been met,
4882      * return success.
4883      */
4884     if (status >= ZONE_IS_DOWN) {
4885         mutex_exit(&zone_status_lock);
4886         mutex_exit(&zonehash_lock);
4887         resume_mounts(zone);
4888         zone_rele(zone);
4889         return (0);
4890     }
4891     /*
4892      * If zone_shutdown() hasn't been called before, go through the motions.
4893      * If it has, there's nothing to do but wait for the kernel threads to
4894      * drain.
4895      */
4896     if (status < ZONE_IS_EMPTY) {
4897         uint_t ntasks;
4899         mutex_enter(&zone->zone_lock);
4900         if ((ntasks = zone->zone_ntasks) != 1) {

```

```

4901         /*
4902          * There's still stuff running.
4903          */
4904         zone_status_set(zone, ZONE_IS_SHUTTING_DOWN);
4905     }
4906     mutex_exit(&zone->zone_lock);
4907     if (ntasks == 1) {
4908         /*
4909          * The only way to create another task is through
4910          * zone_enter(), which will block until we drop
4911          * zonehash_lock. The zone is empty.
4912          */
4913         if (zone->zone_kthreads == NULL) {
4914             /*
4915              * Skip ahead to ZONE_IS_DOWN
4916              */
4917             zone_status_set(zone, ZONE_IS_DOWN);
4918         } else {
4919             zone_status_set(zone, ZONE_IS_EMPTY);
4920         }
4921     }
4922     }
4923     mutex_exit(&zone_status_lock);
4924     mutex_exit(&zonehash_lock);
4925     resume_mounts(zone);
4927     if (error = zone_empty(zone)) {
4928         zone_rele(zone);
4929         return (set_errno(error));
4930     }
4931     /*
4932      * After the zone status goes to ZONE_IS_DOWN this zone will no
4933      * longer be notified of changes to the pools configuration, so
4934      * in order to not end up with a stale pool pointer, we point
4935      * ourselves at the default pool and remove all resource
4936      * visibility. This is especially important as the zone_t may
4937      * languish on the deathrow for a very long time waiting for
4938      * cred's to drain out.
4939      *
4940      * This rebinding of the zone can happen multiple times
4941      * (presumably due to interrupted or parallel systemcalls)
4942      * without any adverse effects.
4943      */
4944     if (pool_lock_intr() != 0) {
4945         zone_rele(zone);
4946         return (set_errno(EINTR));
4947     }
4948     if (pool_state == POOL_ENABLED) {
4949         mutex_enter(&cpu_lock);
4950         zone_pool_set(zone, pool_default);
4951         /*
4952          * The zone no longer needs to be able to see any cpus.
4953          */
4954         zone_pset_set(zone, ZONE_PS_INVALID);
4955         mutex_exit(&cpu_lock);
4956     }
4957     pool_unlock();
4959     /*
4960      * ZSD shutdown callbacks can be executed multiple times, hence
4961      * it is safe to not be holding any locks across this call.
4962      */
4963     zone_zsd_callbacks(zone, ZSD_SHUTDOWN);
4965     mutex_enter(&zone_status_lock);
4966     if (zone->zone_kthreads == NULL && zone_status_get(zone) < ZONE_IS_DOWN)

```

```

4967     zone_status_set(zone, ZONE_IS_DOWN);
4968     mutex_exit(&zone_status_lock);

4970     /*
4971     * Wait for kernel threads to drain.
4972     */
4973     if (!zone_status_wait_sig(zone, ZONE_IS_DOWN)) {
4974         zone_rele(zone);
4975         return (set_errno(EINTR));
4976     }

4978     /*
4979     * Zone can be become down/destroyable even if the above wait
4980     * returns EINTR, so any code added here may never execute.
4981     * (i.e. don't add code here)
4982     */

4984     zone_rele(zone);
4985     return (0);
4986 }

4988 /*
4989 * Log the specified zone's reference counts. The caller should not be
4990 * holding the zone's zone_lock.
4991 */
4992 static void
4993 zone_log_refcounts(zone_t *zone)
4994 {
4995     char *buffer;
4996     char *buffer_position;
4997     uint32_t buffer_size;
4998     uint32_t index;
4999     uint_t ref;
5000     uint_t cred_ref;

5002     /*
5003     * Construct a string representing the subsystem-specific reference
5004     * counts. The counts are printed in ascending order by index into the
5005     * zone_t::zone_subsys_ref array. The list will be surrounded by
5006     * square brackets [] and will only contain nonzero reference counts.
5007     *
5008     * The buffer will hold two square bracket characters plus ten digits,
5009     * one colon, one space, one comma, and some characters for a
5010     * subsystem name per subsystem-specific reference count. (Unsigned 32-
5011     * bit integers have at most ten decimal digits.) The last
5012     * reference count's comma is replaced by the closing square
5013     * bracket and a NULL character to terminate the string.
5014     *
5015     * NOTE: We have to grab the zone's zone_lock to create a consistent
5016     * snapshot of the zone's reference counters.
5017     *
5018     * First, figure out how much space the string buffer will need.
5019     * The buffer's size is stored in buffer_size.
5020     */
5021     buffer_size = 2; /* for the square brackets */
5022     mutex_enter(&zone->zone_lock);
5023     zone->zone_flags |= ZF_REFCOUNTS_LOGGED;
5024     ref = zone->zone_ref;
5025     cred_ref = zone->zone_cred_ref;
5026     for (index = 0; index < ZONE_REF_NUM_SUBSYS; ++index)
5027         if (zone->zone_subsys_ref[index] != 0)
5028             buffer_size += strlen(zone_ref_subsys_names[index]) +
5029                 13;
5030     if (buffer_size == 2) {
5031         /*
5032         * No subsystems had nonzero reference counts. Don't bother

```

```

5033     * with allocating a buffer; just log the general-purpose and
5034     * credential reference counts.
5035     */
5036     mutex_exit(&zone->zone_lock);
5037     (void) strlog(0, 0, 1, SL_CONSOLE | SL_NOTE,
5038     "Zone '%s' (ID: %d) is shutting down, but %u zone "
5039     "references and %u credential references are still extant",
5040     zone->zone_name, zone->zone_id, ref, cred_ref);
5041     return;
5042 }

5044     /*
5045     * buffer_size contains the exact number of characters that the
5046     * buffer will need. Allocate the buffer and fill it with nonzero
5047     * subsystem-specific reference counts. Surround the results with
5048     * square brackets afterwards.
5049     */
5050     buffer = kmem_alloc(buffer_size, KM_SLEEP);
5051     buffer_position = &buffer[1];
5052     for (index = 0; index < ZONE_REF_NUM_SUBSYS; ++index) {
5053         /*
5054         * NOTE: The DDI's version of sprintf() returns a pointer to
5055         * the modified buffer rather than the number of bytes written
5056         * (as in snprintf(3C)). This is unfortunate and annoying.
5057         * Therefore, we'll use snprintf() with INT_MAX to get the
5058         * number of bytes written. Using INT_MAX is safe because
5059         * the buffer is perfectly sized for the data: we'll never
5060         * overrun the buffer.
5061         */
5062         if (zone->zone_subsys_ref[index] != 0)
5063             buffer_position += snprintf(buffer_position, INT_MAX,
5064             "%s: %u,", zone_ref_subsys_names[index],
5065             zone->zone_subsys_ref[index]);
5066     }
5067     mutex_exit(&zone->zone_lock);
5068     buffer[0] = '[';
5069     ASSERT((uintptr_t)(buffer_position - buffer) < buffer_size);
5070     ASSERT(buffer_position[0] == '\0' && buffer_position[-1] == ',');
5071     buffer_position[-1] = ']';

5073     /*
5074     * Log the reference counts and free the message buffer.
5075     */
5076     (void) strlog(0, 0, 1, SL_CONSOLE | SL_NOTE,
5077     "Zone '%s' (ID: %d) is shutting down, but %u zone references and "
5078     "%u credential references are still extant %s", zone->zone_name,
5079     zone->zone_id, ref, cred_ref, buffer);
5080     kmem_free(buffer, buffer_size);
5081 }

5083 /*
5084 * Systemcall entry point to finalize the zone halt process. The caller
5085 * must have already successfully called zone_shutdown().
5086 *
5087 * Upon successful completion, the zone will have been fully destroyed:
5088 * zsched will have exited, destructor callbacks executed, and the zone
5089 * removed from the list of active zones.
5090 */
5091 static int
5092 zone_destroy(zoneid_t zoneid)
5093 {
5094     uint64_t uniqid;
5095     zone_t *zone;
5096     zone_status_t status;
5097     clock_t wait_time;
5098     boolean_t log_refcounts;

```

```

5100     if (secpolicy_zone_config(CRED()) != 0)
5101         return (set_errno(EPERM));
5102     if (zoneid < MIN_USERZONEID || zoneid > MAX_ZONEID)
5103         return (set_errno(EINVAL));

5105     mutex_enter(&zonehash_lock);
5106     /*
5107      * Look for zone under hash lock to prevent races with other
5108      * calls to zone_destroy.
5109      */
5110     if ((zone = zone_find_all_by_id(zoneid)) == NULL) {
5111         mutex_exit(&zonehash_lock);
5112         return (set_errno(EINVAL));
5113     }

5115     if (zone_mount_count(zone->zone_rootpath) != 0) {
5116         mutex_exit(&zonehash_lock);
5117         return (set_errno(EBUSY));
5118     }
5119     mutex_enter(&zone_status_lock);
5120     status = zone_status_get(zone);
5121     if (status < ZONE_IS_DOWN) {
5122         mutex_exit(&zone_status_lock);
5123         mutex_exit(&zonehash_lock);
5124         return (set_errno(EBUSY));
5125     } else if (status == ZONE_IS_DOWN) {
5126         zone_status_set(zone, ZONE_IS_DYING); /* Tell zsched to exit */
5127     }
5128     mutex_exit(&zone_status_lock);
5129     zone_hold(zone);
5130     mutex_exit(&zonehash_lock);

5132     /*
5133      * wait for zsched to exit
5134      */
5135     zone_status_wait(zone, ZONE_IS_DEAD);
5136     zone_zsd_callbacks(zone, ZSD_DESTROY);
5137     zone->zone_netstack = NULL;
5138     unqid = zone->zone_unqid;
5139     zone_rele(zone);
5140     zone = NULL; /* potentially free'd */

5142     log_refcounts = B_FALSE;
5143     wait_time = SEC_TO_TICK(ZONE_DESTROY_TIMEOUT_SECS);
5144     mutex_enter(&zonehash_lock);
5145     for (; /* ever */; ) {
5146         boolean_t unref;
5147         boolean_t refs_have_been_logged;

5149         if ((zone = zone_find_all_by_id(zoneid)) == NULL ||
5150             zone->zone_unqid != unqid) {
5151             /*
5152              * The zone has gone away. Necessary conditions
5153              * are met, so we return success.
5154              */
5155             mutex_exit(&zonehash_lock);
5156             return (0);
5157         }
5158         mutex_enter(&zone->zone_lock);
5159         unref = ZONE_IS_UNREF(zone);
5160         refs_have_been_logged = (zone->zone_flags &
5161             ZF_REFCOUNTS_LOGGED);
5162         mutex_exit(&zone->zone_lock);
5163         if (unref) {
5164             /*

```

```

5165         * There is only one reference to the zone -- that
5166         * added when the zone was added to the hashtables --
5167         * and things will remain this way until we drop
5168         * zonehash_lock... we can go ahead and cleanup the
5169         * zone.
5170         */
5171         break;
5172     }

5174     /*
5175      * Wait for zone_rele_common() or zone_cred_rele() to signal
5176      * zone_destroy_cv. zone_destroy_cv is signaled only when
5177      * some zone's general-purpose reference count reaches one.
5178      * If ZONE_DESTROY_TIMEOUT_SECS seconds elapse while waiting
5179      * on zone_destroy_cv, then log the zone's reference counts and
5180      * continue to wait for zone_rele() and zone_cred_rele().
5181      */
5182     if (!refs_have_been_logged) {
5183         if (!log_refcounts) {
5184             /*
5185              * This thread hasn't timed out waiting on
5186              * zone_destroy_cv yet. Wait wait_time clock
5187              * ticks (initially ZONE_DESTROY_TIMEOUT_SECS
5188              * seconds) for the zone's references to clear.
5189              */
5190             ASSERT(wait_time > 0);
5191             wait_time = cv_reltimedwait_sig(
5192                 &zone_destroy_cv, &zonehash_lock, wait_time,
5193                 TR_SEC);
5194             if (wait_time > 0) {
5195                 /*
5196                  * A thread in zone_rele() or
5197                  * zone_cred_rele() signaled
5198                  * zone_destroy_cv before this thread's
5199                  * wait timed out. The zone might have
5200                  * only one reference left; find out!
5201                  */
5202                 continue;
5203             } else if (wait_time == 0) {
5204                 /* The thread's process was signaled. */
5205                 mutex_exit(&zonehash_lock);
5206                 return (set_errno(EINTR));
5207             }
5209             /*
5210              * The thread timed out while waiting on
5211              * zone_destroy_cv. Even though the thread
5212              * timed out, it has to check whether another
5213              * thread woke up from zone_destroy_cv and
5214              * destroyed the zone.
5215              *
5216              * If the zone still exists and has more than
5217              * one unreleased general-purpose reference,
5218              * then log the zone's reference counts.
5219              */
5220             log_refcounts = B_TRUE;
5221             continue;
5222         }
5224         /*
5225          * The thread already timed out on zone_destroy_cv while
5226          * waiting for subsystems to release the zone's last
5227          * general-purpose references. Log the zone's reference
5228          * counts and wait indefinitely on zone_destroy_cv.
5229          */
5230         zone_log_refcounts(zone);

```

```

5231     }
5232     if (cv_wait_sig(&zone_destroy_cv, &zonehash_lock) == 0) {
5233         /* The thread's process was signaled. */
5234         mutex_exit(&zonehash_lock);
5235         return (set_errno(EINTR));
5236     }
5237 }

5239 /*
5240  * Remove CPU cap for this zone now since we're not going to
5241  * fail below this point.
5242  */
5243 cpucaps_zone_remove(zone);

5245 /* Get rid of the zone's kstats */
5246 zone_kstat_delete(zone);

5248 /* remove the pfexecd doors */
5249 if (zone->zone_pfexecd != NULL) {
5250     klpd_freelist(&zone->zone_pfexecd);
5251     zone->zone_pfexecd = NULL;
5252 }

5254 /* free brand specific data */
5255 if (ZONE_IS_BRAUNDED(zone))
5256     ZBROP(zone)->b_free_brand_data(zone);

5258 /* Say goodbye to brand framework. */
5259 brand_unregister_zone(zone->zone_brand);

5261 /*
5262  * It is now safe to let the zone be recreated; remove it from the
5263  * lists. The memory will not be freed until the last cred
5264  * reference goes away.
5265  */
5266 ASSERT(zonecount > 1); /* must be > 1; can't destroy global zone */
5267 zonecount--;
5268 /* remove from active list and hash tables */
5269 list_remove(&zone_active, zone);
5270 (void) mod_hash_destroy(zonehashbyname,
5271     (mod_hash_key_t)zone->zone_name);
5272 (void) mod_hash_destroy(zonehashbyid,
5273     (mod_hash_key_t)(uintptr_t)zone->zone_id);
5274 if (zone->zone_flags & ZF_HASHED_LABEL)
5275     (void) mod_hash_destroy(zonehashbylabel,
5276     (mod_hash_key_t)zone->zone_slabel);
5277 mutex_exit(&zonehash_lock);

5279 /*
5280  * Release the root vnode; we're not using it anymore. Nor should any
5281  * other thread that might access it exist.
5282  */
5283 if (zone->zone_rootvp != NULL) {
5284     VN_RELE(zone->zone_rootvp);
5285     zone->zone_rootvp = NULL;
5286 }

5288 /* add to deathrow list */
5289 mutex_enter(&zone_deathrow_lock);
5290 list_insert_tail(&zone_deathrow, zone);
5291 mutex_exit(&zone_deathrow_lock);

5293 /*
5294  * Drop last reference (which was added by zsched()), this will
5295  * free the zone unless there are outstanding cred references.
5296  */

```

```

5297     zone_rele(zone);
5298     return (0);
5299 }

5301 /*
5302  * Systemcall entry point for zone_getattr(2).
5303  */
5304 static ssize_t
5305 zone_getattr(zoneid_t zoneid, int attr, void *buf, size_t bufsize)
5306 {
5307     size_t size;
5308     int error = 0, err;
5309     zone_t *zone;
5310     char *zonepath;
5311     char *outstr;
5312     zone_status_t zone_status;
5313     pid_t initpid;
5314     boolean_t global = (curzone == global_zone);
5315     boolean_t inzone = (curzone->zone_id == zoneid);
5316     ushort_t flags;
5317     zone_net_data_t *zbuf;

5319     mutex_enter(&zonehash_lock);
5320     if ((zone = zone_find_all_by_id(zoneid)) == NULL) {
5321         mutex_exit(&zonehash_lock);
5322         return (set_errno(EINVAL));
5323     }
5324     zone_status = zone_status_get(zone);
5325     if (zone_status < ZONE_IS_INITIALIZED) {
5326         mutex_exit(&zonehash_lock);
5327         return (set_errno(EINVAL));
5328     }
5329     zone_hold(zone);
5330     mutex_exit(&zonehash_lock);

5332     /*
5333      * If not in the global zone, don't show information about other zones,
5334      * unless the system is labeled and the local zone's label dominates
5335      * the other zone.
5336      */
5337     if (!zone_list_access(zone)) {
5338         zone_rele(zone);
5339         return (set_errno(EINVAL));
5340     }

5342     switch (attr) {
5343     case ZONE_ATTR_ROOT:
5344         if (global) {
5345             /*
5346              * Copy the path to trim the trailing "/" (except for
5347              * the global zone).
5348              */
5349             if (zone != global_zone)
5350                 size = zone->zone_rootpathlen - 1;
5351             else
5352                 size = zone->zone_rootpathlen;
5353             zonepath = kmem_alloc(size, KM_SLEEP);
5354             bcopy(zone->zone_rootpath, zonepath, size);
5355             zonepath[size - 1] = '\0';
5356         } else {
5357             if (inzone || !is_system_labeled()) {
5358                 /*
5359                  * Caller is not in the global zone.
5360                  * if the query is on the current zone
5361                  * or the system is not labeled,
5362                  * just return faked-up path for current zone.

```

```

5363     */
5364     zonepath = "/";
5365     size = 2;
5366 } else {
5367     /*
5368     * Return related path for current zone.
5369     */
5370     int prefix_len = strlen(zone_prefix);
5371     int zname_len = strlen(zone->zone_name);

5373     size = prefix_len + zname_len + 1;
5374     zonepath = kmem_alloc(size, KM_SLEEP);
5375     bcopy(zone_prefix, zonepath, prefix_len);
5376     bcopy(zone->zone_name, zonepath +
5377           prefix_len, zname_len);
5378     zonepath[size - 1] = '\0';
5379 }
5380 }
5381 if (bufsize > size)
5382     bufsize = size;
5383 if (buf != NULL) {
5384     err = copyoutstr(zonepath, buf, bufsize, NULL);
5385     if (err != 0 && err != ENAMETOOLONG)
5386         error = EFAULT;
5387 }
5388 if (global || (is_system_labeled() && !inzone))
5389     kmem_free(zonepath, size);
5390 break;

5392 case ZONE_ATTR_NAME:
5393     size = strlen(zone->zone_name) + 1;
5394     if (bufsize > size)
5395         bufsize = size;
5396     if (buf != NULL) {
5397         err = copyoutstr(zone->zone_name, buf, bufsize, NULL);
5398         if (err != 0 && err != ENAMETOOLONG)
5399             error = EFAULT;
5400     }
5401     break;

5403 case ZONE_ATTR_STATUS:
5404     /*
5405     * Since we're not holding zonehash_lock, the zone status
5406     * may be anything; leave it up to userland to sort it out.
5407     */
5408     size = sizeof (zone_status);
5409     if (bufsize > size)
5410         bufsize = size;
5411     zone_status = zone_status_get(zone);
5412     if (buf != NULL &&
5413         copyout(&zone_status, buf, bufsize) != 0)
5414         error = EFAULT;
5415     break;

5416 case ZONE_ATTR_FLAGS:
5417     size = sizeof (zone->zone_flags);
5418     if (bufsize > size)
5419         bufsize = size;
5420     flags = zone->zone_flags;
5421     if (buf != NULL &&
5422         copyout(&flags, buf, bufsize) != 0)
5423         error = EFAULT;
5424     break;

5425 case ZONE_ATTR_PRIVSET:
5426     size = sizeof (priv_set_t);
5427     if (bufsize > size)
5428         bufsize = size;

```

```

5429     if (buf != NULL &&
5430         copyout(zone->zone_privset, buf, bufsize) != 0)
5431         error = EFAULT;
5432     break;
5433 case ZONE_ATTR_UNIQID:
5434     size = sizeof (zone->zone_uniqid);
5435     if (bufsize > size)
5436         bufsize = size;
5437     if (buf != NULL &&
5438         copyout(&zone->zone_uniqid, buf, bufsize) != 0)
5439         error = EFAULT;
5440     break;
5441 case ZONE_ATTR_POOLID:
5442     {
5443         pool_t *pool;
5444         poolid_t poolid;

5446         if (pool_lock_intr() != 0) {
5447             error = EINTR;
5448             break;
5449         }
5450         pool = zone_pool_get(zone);
5451         poolid = pool->pool_id;
5452         pool_unlock();
5453         size = sizeof (poolid);
5454         if (bufsize > size)
5455             bufsize = size;
5456         if (buf != NULL && copyout(&poolid, buf, size) != 0)
5457             error = EFAULT;
5458     }
5459     break;
5460 case ZONE_ATTR_SLBL:
5461     size = sizeof (bslabel_t);
5462     if (bufsize > size)
5463         bufsize = size;
5464     if (zone->zone_slab == NULL)
5465         error = EINVAL;
5466     else if (buf != NULL &&
5467         copyout(label2bslabel(zone->zone_slab), buf,
5468                 bufsize) != 0)
5469         error = EFAULT;
5470     break;
5471 case ZONE_ATTR_INITPID:
5472     size = sizeof (initpid);
5473     if (bufsize > size)
5474         bufsize = size;
5475     initpid = zone->zone_proc_initpid;
5476     if (initpid == -1) {
5477         error = ESRCH;
5478         break;
5479     }
5480     if (buf != NULL &&
5481         copyout(&initpid, buf, bufsize) != 0)
5482         error = EFAULT;
5483     break;
5484 case ZONE_ATTR_BRAND:
5485     size = strlen(zone->zone_brand->b_name) + 1;

5487     if (bufsize > size)
5488         bufsize = size;
5489     if (buf != NULL) {
5490         err = copyoutstr(zone->zone_brand->b_name, buf,
5491                         bufsize, NULL);
5492         if (err != 0 && err != ENAMETOOLONG)
5493             error = EFAULT;
5494     }

```

```

5495         break;
5496     case ZONE_ATTR_INITNAME:
5497         size = strlen(zone->zone_initname) + 1;
5498         if (bufsize > size)
5499             bufsize = size;
5500         if (buf != NULL) {
5501             err = copyoutstr(zone->zone_initname, buf, bufsize,
5502                 NULL);
5503             if (err != 0 && err != ENAMETOOLONG)
5504                 error = EFAULT;
5505         }
5506         break;
5507     case ZONE_ATTR_BOOTARGS:
5508         if (zone->zone_bootargs == NULL)
5509             outstr = "";
5510         else
5511             outstr = zone->zone_bootargs;
5512         size = strlen(outstr) + 1;
5513         if (bufsize > size)
5514             bufsize = size;
5515         if (buf != NULL) {
5516             err = copyoutstr(outstr, buf, bufsize, NULL);
5517             if (err != 0 && err != ENAMETOOLONG)
5518                 error = EFAULT;
5519         }
5520         break;
5521     case ZONE_ATTR_PHYS_MCAP:
5522         size = sizeof (zone->zone_phys_mcap);
5523         if (bufsize > size)
5524             bufsize = size;
5525         if (buf != NULL &&
5526             copyout(&zone->zone_phys_mcap, buf, bufsize) != 0)
5527             error = EFAULT;
5528         break;
5529     case ZONE_ATTR_SCHED_CLASS:
5530         mutex_enter(&class_lock);
5532         if (zone->zone_defaultcid >= loaded_classes)
5533             outstr = "";
5534         else
5535             outstr = sclass[zone->zone_defaultcid].cl_name;
5536         size = strlen(outstr) + 1;
5537         if (bufsize > size)
5538             bufsize = size;
5539         if (buf != NULL) {
5540             err = copyoutstr(outstr, buf, bufsize, NULL);
5541             if (err != 0 && err != ENAMETOOLONG)
5542                 error = EFAULT;
5543         }
5545         mutex_exit(&class_lock);
5546         break;
5547     case ZONE_ATTR_HOSTID:
5548         if (zone->zone_hostid != HW_INVALID_HOSTID &&
5549             bufsize == sizeof (zone->zone_hostid) {
5550             size = sizeof (zone->zone_hostid);
5551             if (buf != NULL && copyout(&zone->zone_hostid, buf,
5552                 bufsize) != 0)
5553                 error = EFAULT;
5554         } else {
5555             error = EINVAL;
5556         }
5557         break;
5558     case ZONE_ATTR_FS_ALLOWED:
5559         if (zone->zone_fs_allowed == NULL)
5560             outstr = "";

```

```

5561         else
5562             outstr = zone->zone_fs_allowed;
5563         size = strlen(outstr) + 1;
5564         if (bufsize > size)
5565             bufsize = size;
5566         if (buf != NULL) {
5567             err = copyoutstr(outstr, buf, bufsize, NULL);
5568             if (err != 0 && err != ENAMETOOLONG)
5569                 error = EFAULT;
5570         }
5571         break;
5572     case ZONE_ATTR_SECFLAGS:
5573         size = sizeof (zone->zone_secflags);
5574         if (bufsize > size)
5575             bufsize = size;
5576         if ((err = copyout(&zone->zone_secflags, buf, bufsize)) != 0)
5577             error = EFAULT;
5578         break;
5579     #endif /* ! codereview */
5580     case ZONE_ATTR_NETWORK:
5581         zbuf = kmem_alloc(bufsize, KM_SLEEP);
5582         if (copyin(buf, zbuf, bufsize) != 0) {
5583             error = EFAULT;
5584         } else {
5585             error = zone_get_network(zoneid, zbuf);
5586             if (error == 0 && copyout(zbuf, buf, bufsize) != 0)
5587                 error = EFAULT;
5588         }
5589         kmem_free(zbuf, bufsize);
5590         break;
5591     default:
5592         if ((attr >= ZONE_ATTR_BRAND_ATTRS) && ZONE_IS_BRAINED(zone)) {
5593             size = bufsize;
5594             error = ZBROP(zone)->b_getattr(zone, attr, buf, &size);
5595         } else {
5596             error = EINVAL;
5597         }
5598     }
5599     zone_rele(zone);
5601     if (error)
5602         return (set_errno(error));
5603     return ((ssize_t)size);
5604 }
5606 /*
5607  * Systemcall entry point for zone_setattr(2).
5608  */
5609 /*ARGSUSED*/
5610 static int
5611 zone_setattr(zoneid_t zoneid, int attr, void *buf, size_t bufsize)
5612 {
5613     zone_t *zone;
5614     zone_status_t zone_status;
5615     int err = -1;
5616     zone_net_data_t *zbuf;
5618     if (secpolicy_zone_config(CRED()) != 0)
5619         return (set_errno(EPERM));
5621     /*
5622      * Only the ZONE_ATTR_PHYS_MCAP attribute can be set on the
5623      * global zone.
5624      */
5625     if (zoneid == GLOBAL_ZONEID && attr != ZONE_ATTR_PHYS_MCAP) {
5626         return (set_errno(EINVAL));

```

```

5627     }
5629     mutex_enter(&zonehash_lock);
5630     if ((zone = zone_find_all_by_id(zoneid)) == NULL) {
5631         mutex_exit(&zonehash_lock);
5632         return (set_errno(EINVAL));
5633     }
5634     zone_hold(zone);
5635     mutex_exit(&zonehash_lock);

5637 /*
5638  * At present most attributes can only be set on non-running,
5639  * non-global zones.
5640  */
5641     zone_status = zone_status_get(zone);
5642     if (attr != ZONE_ATTR_PHYS_MCAP && zone_status > ZONE_IS_READY) {
5643         err = EINVAL;
5644         goto done;
5645     }

5647     switch (attr) {
5648     case ZONE_ATTR_INITNAME:
5649         err = zone_set_initname(zone, (const char *)buf);
5650         break;
5651     case ZONE_ATTR_INITNORESTART:
5652         zone->zone_restart_init = B_FALSE;
5653         err = 0;
5654         break;
5655     case ZONE_ATTR_BOOTARGS:
5656         err = zone_set_bootargs(zone, (const char *)buf);
5657         break;
5658     case ZONE_ATTR_BRAND:
5659         err = zone_set_brand(zone, (const char *)buf);
5660         break;
5661     case ZONE_ATTR_FS_ALLOWED:
5662         err = zone_set_fs_allowed(zone, (const char *)buf);
5663         break;
5664     case ZONE_ATTR_SECFLAGS:
5665         err = zone_set_secflags(zone, (psecflags_t *)buf);
5666         break;
5667 #endif /* ! codereview */
5668     case ZONE_ATTR_PHYS_MCAP:
5669         err = zone_set_phys_mcap(zone, (const uint64_t *)buf);
5670         break;
5671     case ZONE_ATTR_SCHED_CLASS:
5672         err = zone_set_sched_class(zone, (const char *)buf);
5673         break;
5674     case ZONE_ATTR_HOSTID:
5675         if (bufsize == sizeof (zone->zone_hostid)) {
5676             if (copyin(buf, &zone->zone_hostid, bufsize) == 0)
5677                 err = 0;
5678             else
5679                 err = EFAULT;
5680         } else {
5681             err = EINVAL;
5682         }
5683         break;
5684     case ZONE_ATTR_NETWORK:
5685         if (bufsize > (PIPE_BUF + sizeof (zone_net_data_t))) {
5686             err = EINVAL;
5687             break;
5688         }
5689         zbuf = kmem_alloc(bufsize, KM_SLEEP);
5690         if (copyin(buf, zbuf, bufsize) != 0) {
5691             kmem_free(zbuf, bufsize);
5692             err = EFAULT;

```

```

5693         break;
5694     }
5695     err = zone_set_network(zoneid, zbuf);
5696     kmem_free(zbuf, bufsize);
5697     break;
5698     default:
5699         if ((attr >= ZONE_ATTR_BRAND_ATTRS) && ZONE_IS_BRANDED(zone))
5700             err = ZBROP(zone)->b_setattr(zone, attr, buf, bufsize);
5701         else
5702             err = EINVAL;
5703     }

5705 done:
5706     zone_rele(zone);
5707     ASSERT(err != -1);
5708     return (err != 0 ? set_errno(err) : 0);
5709 }

5711 /*
5712  * Return zero if the process has at least one vnode mapped in to its
5713  * address space which shouldn't be allowed to change zones.
5714  *
5715  * Also return zero if the process has any shared mappings which reserve
5716  * swap. This is because the counting for zone.max-swap does not allow swap
5717  * reservation to be shared between zones. zone swap reservation is counted
5718  * on zone->zone_max_swap.
5719  */
5720 static int
5721 as_can_change_zones(void)
5722 {
5723     proc_t *pp = curproc;
5724     struct seg *seg;
5725     struct as *as = pp->p_as;
5726     vnode_t *vp;
5727     int allow = 1;

5729     ASSERT(pp->p_as != &kas);
5730     AS_LOCK_ENTER(as, RW_READER);
5731     for (seg = AS_SEGFIRST(as); seg != NULL; seg = AS_SEGNEXT(as, seg)) {

5733         /*
5734          * Cannot enter zone with shared anon memory which
5735          * reserves swap. See comment above.
5736          */
5737         if (seg_can_change_zones(seg) == B_FALSE) {
5738             allow = 0;
5739             break;
5740         }
5741         /*
5742          * if we can't get a backing vnode for this segment then skip
5743          * it.
5744          */
5745         vp = NULL;
5746         if (SEGOP_GETVP(seg, seg->s_base, &vp) != 0 || vp == NULL)
5747             continue;
5748         if (!vn_can_change_zones(vp)) { /* bail on first match */
5749             allow = 0;
5750             break;
5751         }
5752     }
5753     AS_LOCK_EXIT(as);
5754     return (allow);
5755 }

5757 /*
5758  * Count swap reserved by curproc's address space

```

```

5759 */
5760 static size_t
5761 as_swresv(void)
5762 {
5763     proc_t *pp = curproc;
5764     struct seg *seg;
5765     struct as *as = pp->p_as;
5766     size_t swap = 0;

5768     ASSERT(pp->p_as != &kas);
5769     ASSERT(AS_WRITE_HELD(as));
5770     for (seg = AS_SEGFIRST(as); seg != NULL; seg = AS_SEGNEXT(as, seg))
5771         swap += seg_swresv(seg);

5773     return (swap);
5774 }

5776 /*
5777  * Systemcall entry point for zone_enter().
5778  *
5779  * The current process is injected into said zone.  In the process
5780  * it will change its project membership, privileges, rootdir/cwd,
5781  * zone-wide rctls, and pool association to match those of the zone.
5782  *
5783  * The first zone_enter() called while the zone is in the ZONE_IS_READY
5784  * state will transition it to ZONE_IS_RUNNING.  Processes may only
5785  * enter a zone that is "ready" or "running".
5786  */
5787 static int
5788 zone_enter(zoneid_t zoneid)
5789 {
5790     zone_t *zone;
5791     vnode_t *vp;
5792     proc_t *pp = curproc;
5793     contract_t *ct;
5794     cont_process_t *ctp;
5795     task_t *tk, *oldtk;
5796     kproject_t *zone_proj0;
5797     cred_t *cr, *newcr;
5798     pool_t *oldpool, *newpool;
5799     sess_t *sp;
5800     uid_t uid;
5801     zone_status_t status;
5802     int err = 0;
5803     rctl_entity_p_t e;
5804     size_t swap;
5805     kthread_id_t t;

5807     if (secpolicy_zone_config(CRED()) != 0)
5808         return (set_errno(EPERM));
5809     if (zoneid < MIN_USERZONEID || zoneid > MAX_ZONEID)
5810         return (set_errno(EINVAL));

5812     /*
5813      * Stop all lwps so we don't need to hold a lock to look at
5814      * curproc->p_zone.  This needs to happen before we grab any
5815      * locks to avoid deadlock (another lwp in the process could
5816      * be waiting for the held lock).
5817      */
5818     if (curthread != pp->p_agenttp && !holdlwps(SHOLDFORK))
5819         return (set_errno(EINTR));

5821     /*
5822      * Make sure we're not changing zones with files open or mapped in
5823      * to our address space which shouldn't be changing zones.
5824      */

```

```

5825     if (!files_can_change_zones()) {
5826         err = EBADF;
5827         goto out;
5828     }
5829     if (!as_can_change_zones()) {
5830         err = EFAULT;
5831         goto out;
5832     }

5834     mutex_enter(&zonehash_lock);
5835     if (pp->p_zone != global_zone) {
5836         mutex_exit(&zonehash_lock);
5837         err = EINVAL;
5838         goto out;
5839     }

5841     zone = zone_find_all_by_id(zoneid);
5842     if (zone == NULL) {
5843         mutex_exit(&zonehash_lock);
5844         err = EINVAL;
5845         goto out;
5846     }

5848     /*
5849      * To prevent processes in a zone from holding contracts on
5850      * extrazonal resources, and to avoid process contract
5851      * memberships which span zones, contract holders and processes
5852      * which aren't the sole members of their encapsulating process
5853      * contracts are not allowed to zone_enter.
5854      */
5855     ctp = pp->p_ct_process;
5856     ct = &ctp->conp_contract;
5857     mutex_enter(&ct->ct_lock);
5858     mutex_enter(&pp->p_lock);
5859     if ((avl_numnodes(&pp->p_ct_held) != 0) || (ctp->conp_nmembers != 1)) {
5860         mutex_exit(&pp->p_lock);
5861         mutex_exit(&ct->ct_lock);
5862         mutex_exit(&zonehash_lock);
5863         err = EINVAL;
5864         goto out;
5865     }

5867     /*
5868      * Moreover, we don't allow processes whose encapsulating
5869      * process contracts have inherited extrazonal contracts.
5870      * While it would be easier to eliminate all process contracts
5871      * with inherited contracts, we need to be able to give a
5872      * restarted init (or other zone-penetrating process) its
5873      * predecessor's contracts.
5874      */
5875     if (ctp->conp_ninherited != 0) {
5876         contract_t *next;
5877         for (next = list_head(&ctp->conp_inherited); next;
5878              next = list_next(&ctp->conp_inherited, next)) {
5879             if (contract_getzuniqid(next) != zone->zone_uniqid) {
5880                 mutex_exit(&pp->p_lock);
5881                 mutex_exit(&ct->ct_lock);
5882                 mutex_exit(&zonehash_lock);
5883                 err = EINVAL;
5884                 goto out;
5885             }
5886         }
5887     }

5889     mutex_exit(&pp->p_lock);
5890     mutex_exit(&ct->ct_lock);

```



```

5892     status = zone_status_get(zone);
5893     if (status < ZONE_IS_READY || status >= ZONE_IS_SHUTTING_DOWN) {
5894         /*
5895          * Can't join
5896          */
5897         mutex_exit(&zonehash_lock);
5898         err = EINVAL;
5899         goto out;
5900     }

5902     /*
5903     * Make sure new priv set is within the permitted set for caller
5904     */
5905     if (!priv_issubset(zone->zone_privset, &CR_OPPRIV(CRED()))) {
5906         mutex_exit(&zonehash_lock);
5907         err = EPERM;
5908         goto out;
5909     }
5910     /*
5911     * We want to momentarily drop zonehash_lock while we optimistically
5912     * bind curproc to the pool it should be running in. This is safe
5913     * since the zone can't disappear (we have a hold on it).
5914     */
5915     zone_hold(zone);
5916     mutex_exit(&zonehash_lock);

5918     /*
5919     * Grab pool_lock to keep the pools configuration from changing
5920     * and to stop ourselves from getting rebound to another pool
5921     * until we join the zone.
5922     */
5923     if (pool_lock_intr() != 0) {
5924         zone_rele(zone);
5925         err = EINTR;
5926         goto out;
5927     }
5928     ASSERT(secpolicy_pool(CRED()) == 0);
5929     /*
5930     * Bind ourselves to the pool currently associated with the zone.
5931     */
5932     oldpool = curproc->p_pool;
5933     newpool = zone_pool_get(zone);
5934     if (pool_state == POOL_ENABLED && newpool != oldpool &&
5935         (err = pool_do_bind(newpool, P_PID, P_MYID,
5936             POOL_BIND_ALL)) != 0) {
5937         pool_unlock();
5938         zone_rele(zone);
5939         goto out;
5940     }

5942     /*
5943     * Grab cpu_lock now; we'll need it later when we call
5944     * task_join().
5945     */
5946     mutex_enter(&cpu_lock);
5947     mutex_enter(&zonehash_lock);
5948     /*
5949     * Make sure the zone hasn't moved on since we dropped zonehash_lock.
5950     */
5951     if (zone_status_get(zone) >= ZONE_IS_SHUTTING_DOWN) {
5952         /*
5953          * Can't join anymore.
5954          */
5955         mutex_exit(&zonehash_lock);
5956         mutex_exit(&cpu_lock);

```

```

5957         if (pool_state == POOL_ENABLED &&
5958             newpool != oldpool)
5959             (void) pool_do_bind(oldpool, P_PID, P_MYID,
5960                 POOL_BIND_ALL);
5961         pool_unlock();
5962         zone_rele(zone);
5963         err = EINVAL;
5964         goto out;
5965     }

5967     /*
5968     * a_lock must be held while transferring locked memory and swap
5969     * reservation from the global zone to the non global zone because
5970     * asynchronous faults on the processes' address space can lock
5971     * memory and reserve swap via MCL_FUTURE and MAP_NORESERVE
5972     * segments respectively.
5973     */
5974     AS_LOCK_ENTER(pp->p_as, RW_WRITER);
5975     swap = as_swresv();
5976     mutex_enter(&pp->p_lock);
5977     zone_proj0 = zone->zone_zsched->p_task->tk_proj;
5978     /* verify that we do not exceed and task or lwp limits */
5979     mutex_enter(&zone->zone_nlwps_lock);
5980     /* add new lwps to zone and zone's proj0 */
5981     zone_proj0->kpj_nlwps += pp->p_lwpcnt;
5982     zone->zone_nlwps += pp->p_lwpcnt;
5983     /* add 1 task to zone's proj0 */
5984     zone_proj0->kpj_nprocs += 1;

5986     zone_proj0->kpj_nprocs++;
5987     zone->zone_nprocs++;
5988     mutex_exit(&zone->zone_nlwps_lock);

5990     mutex_enter(&zone->zone_mem_lock);
5991     zone->zone_locked_mem += pp->p_locked_mem;
5992     zone_proj0->kpj_data.kpd_locked_mem += pp->p_locked_mem;
5993     zone->zone_max_swap += swap;
5994     mutex_exit(&zone->zone_mem_lock);

5996     mutex_enter(&(zone_proj0->kpj_data.kpd_crypto_lock));
5997     zone_proj0->kpj_data.kpd_crypto_mem += pp->p_crypto_mem;
5998     mutex_exit(&(zone_proj0->kpj_data.kpd_crypto_lock));

6000     /* remove lwps and process from proc's old zone and old project */
6001     mutex_enter(&pp->p_zone->zone_nlwps_lock);
6002     pp->p_zone->zone_nlwps -= pp->p_lwpcnt;
6003     pp->p_task->tk_proj->kpj_nlwps -= pp->p_lwpcnt;
6004     pp->p_task->tk_proj->kpj_nprocs--;
6005     pp->p_zone->zone_nprocs--;
6006     mutex_exit(&pp->p_zone->zone_nlwps_lock);

6008     mutex_enter(&pp->p_zone->zone_mem_lock);
6009     pp->p_zone->zone_locked_mem -= pp->p_locked_mem;
6010     pp->p_task->tk_proj->kpj_data.kpd_locked_mem -= pp->p_locked_mem;
6011     pp->p_zone->zone_max_swap -= swap;
6012     mutex_exit(&pp->p_zone->zone_mem_lock);

6014     mutex_enter(&(pp->p_task->tk_proj->kpj_data.kpd_crypto_lock));
6015     pp->p_task->tk_proj->kpj_data.kpd_crypto_mem -= pp->p_crypto_mem;
6016     mutex_exit(&(pp->p_task->tk_proj->kpj_data.kpd_crypto_lock));

6018     pp->p_flag |= SZONETOP;
6019     pp->p_zone = zone;
6020     mutex_exit(&pp->p_lock);
6021     AS_LOCK_EXIT(pp->p_as);

```

```

6023  /*
6024  * Joining the zone cannot fail from now on.
6025  *
6026  * This means that a lot of the following code can be commonized and
6027  * shared with zsched().
6028  */
6030  /*
6031  * If the process contract fmri was inherited, we need to
6032  * flag this so that any contract status will not leak
6033  * extra zone information, svc_fmri in this case
6034  */
6035  if (ctp->comp_svc_ctid != ct->ct_id) {
6036      mutex_enter(&ct->ct_lock);
6037      ctp->comp_svc_zone_enter = ct->ct_id;
6038      mutex_exit(&ct->ct_lock);
6039  }
6041  /*
6042  * Reset the encapsulating process contract's zone.
6043  */
6044  ASSERT(ct->ct_mzuniqid == GLOBAL_ZONEUNIQID);
6045  contract_setzuniqid(ct, zone->zone_uniqid);
6047  /*
6048  * Create a new task and associate the process with the project keyed
6049  * by (projid,zoneid).
6050  *
6051  * We might as well be in project 0; the global zone's projid doesn't
6052  * make much sense in a zone anyhow.
6053  *
6054  * This also increments zone_ntasks, and returns with p_lock held.
6055  */
6056  tk = task_create(0, zone);
6057  oldtk = task_join(tk, 0);
6058  mutex_exit(&cpu_lock);
6060  /*
6061  * call RCTLOP_SET functions on this proc
6062  */
6063  e.rcep_p.zone = zone;
6064  e.rcep_t = RCENTITY_ZONE;
6065  (void) rctl_set_dup(NULL, NULL, pp, &e, zone->zone_rctls, NULL,
6066      RCD_CALLBACK);
6067  mutex_exit(&pp->p_lock);
6069  /*
6070  * We don't need to hold any of zsched's locks here; not only do we know
6071  * the process and zone aren't going away, we know its session isn't
6072  * changing either.
6073  *
6074  * By joining zsched's session here, we mimic the behavior in the
6075  * global zone of init's sid being the pid of sched. We extend this
6076  * to all zlogin-like zone_enter()'ing processes as well.
6077  */
6078  mutex_enter(&pidlock);
6079  sp = zone->zone_zsched->p_sessp;
6080  sess_hold(zone->zone_zsched);
6081  mutex_enter(&pp->p_lock);
6082  pgsplit(pp);
6083  sess_rele(pp->p_sessp, B_TRUE);
6084  pp->p_sessp = sp;
6085  pgjoin(pp, zone->zone_zsched->p_pidp);
6087  /*
6088  * If any threads are scheduled to be placed on zone wait queue they

```

```

6089  * should abandon the idea since the wait queue is changing.
6090  * We need to be holding pidlock & p_lock to do this.
6091  */
6092  if ((t = pp->p_tlist) != NULL) {
6093      do {
6094          thread_lock(t);
6095          /*
6096           * Kick this thread so that he doesn't sit
6097           * on a wrong wait queue.
6098           */
6099          if (ISWAITING(t))
6100              setrun_locked(t);
6102          if (t->t_schedflag & TS_ANYWAITQ)
6103              t->t_schedflag &= ~ TS_ANYWAITQ;
6105          thread_unlock(t);
6106          } while ((t = t->t_forw) != pp->p_tlist);
6107  }
6109  /*
6110  * If there is a default scheduling class for the zone and it is not
6111  * the class we are currently in, change all of the threads in the
6112  * process to the new class. We need to be holding pidlock & p_lock
6113  * when we call parmsset so this is a good place to do it.
6114  */
6115  if (zone->zone_defaultcid > 0 &&
6116      zone->zone_defaultcid != curthread->t_cid) {
6117      pcparms_t pcparms;
6119      pcparms.pc_cid = zone->zone_defaultcid;
6120      pcparms.pc_clparms[0] = 0;
6122      /*
6123       * If setting the class fails, we still want to enter the zone.
6124       */
6125      if ((t = pp->p_tlist) != NULL) {
6126          do {
6127              (void) parmsset(&pcparms, t);
6128              } while ((t = t->t_forw) != pp->p_tlist);
6129          }
6130  }
6132  mutex_exit(&pp->p_lock);
6133  mutex_exit(&pidlock);
6135  mutex_exit(&zonehash_lock);
6136  /*
6137  * We're firmly in the zone; let pools progress.
6138  */
6139  pool_unlock();
6140  task_rele(oldtk);
6141  /*
6142  * We don't need to retain a hold on the zone since we already
6143  * incremented zone_ntasks, so the zone isn't going anywhere.
6144  */
6145  zone_rele(zone);
6147  /*
6148  * Chroot
6149  */
6150  vp = zone->zone_rootvp;
6151  zone_chdir(vp, &PTOU(pp)->u_cdir, pp);
6152  zone_chdir(vp, &PTOU(pp)->u_rdir, pp);
6154  /*

```

```

6155     * Change process credentials
6156     */
6157     newcr = cralloc();
6158     mutex_enter(&pp->p_crlock);
6159     cr = pp->p_cred;
6160     crcopy_to(cr, newcr);
6161     crsetzone(newcr, zone);
6162     pp->p_cred = newcr;

6164     /*
6165     * Restrict all process privilege sets to zone limit
6166     */
6167     priv_intersect(zone->zone_privset, &CR_PPRIV(newcr));
6168     priv_intersect(zone->zone_privset, &CR_EPRIV(newcr));
6169     priv_intersect(zone->zone_privset, &CR_IPRIV(newcr));
6170     priv_intersect(zone->zone_privset, &CR_LPRIV(newcr));
6171     mutex_exit(&pp->p_crlock);
6172     crset(pp, newcr);

6174     /*
6175     * Adjust upcount to reflect zone entry.
6176     */
6177     uid = crgetruid(newcr);
6178     mutex_enter(&pidlock);
6179     upcount_dec(uid, GLOBAL_ZONEID);
6180     upcount_inc(uid, zoneid);
6181     mutex_exit(&pidlock);

6183     /*
6184     * Set up core file path and content.
6185     */
6186     set_core_defaults();

6188 out:
6189     /*
6190     * Let the other lwps continue.
6191     */
6192     mutex_enter(&pp->p_lock);
6193     if (curthread != pp->p_agenttp)
6194         continuelwps(pp);
6195     mutex_exit(&pp->p_lock);

6197     return (err != 0 ? set_errno(err) : 0);
6198 }

6200 /*
6201 * Systemcall entry point for zone_list(2).
6202 *
6203 * Processes running in a (non-global) zone only see themselves.
6204 * On labeled systems, they see all zones whose label they dominate.
6205 */
6206 static int
6207 zone_list(zoneid_t *zoneidlist, uint_t *numzones)
6208 {
6209     zoneid_t *zoneids;
6210     zone_t *zone, *myzone;
6211     uint_t user_nzones, real_nzones;
6212     uint_t domi_nzones;
6213     int error;

6215     if (copyin(numzones, &user_nzones, sizeof (uint_t)) != 0)
6216         return (set_errno(EFAULT));

6218     myzone = curproc->p_zone;
6219     if (myzone != global_zone) {
6220         bslab_t *mybslab;

```

```

6222         if (!is_system_labeled()) {
6223             /* just return current zone */
6224             real_nzones = domi_nzones = 1;
6225             zoneids = kmem_alloc(sizeof (zoneid_t), KM_SLEEP);
6226             zoneids[0] = myzone->zone_id;
6227         } else {
6228             /* return all zones that are dominated */
6229             mutex_enter(&zonehash_lock);
6230             real_nzones = zonecount;
6231             domi_nzones = 0;
6232             if (real_nzones > 0) {
6233                 zoneids = kmem_alloc(real_nzones *
6234                     sizeof (zoneid_t), KM_SLEEP);
6235                 mybslab = label2bslabel(myzone->zone_label);
6236                 for (zone = list_head(&zone_active);
6237                     zone != NULL;
6238                     zone = list_next(&zone_active, zone)) {
6239                     if (zone->zone_id == GLOBAL_ZONEID)
6240                         continue;
6241                     if (zone != myzone &&
6242                         (zone->zone_flags & ZF_IS_SCRATCH))
6243                         continue;
6244                     /*
6245                     * Note that a label always dominates
6246                     * itself, so myzone is always included
6247                     * in the list.
6248                     */
6249                     if (bldominates(mybslab,
6250                         label2bslabel(zone->zone_label))) {
6251                         zoneids[domi_nzones++] =
6252                             zone->zone_id;
6253                     }
6254                 }
6255             }
6256             mutex_exit(&zonehash_lock);
6257         }
6258     } else {
6259         mutex_enter(&zonehash_lock);
6260         real_nzones = zonecount;
6261         domi_nzones = 0;
6262         if (real_nzones > 0) {
6263             zoneids = kmem_alloc(real_nzones * sizeof (zoneid_t),
6264                 KM_SLEEP);
6265             for (zone = list_head(&zone_active); zone != NULL;
6266                 zone = list_next(&zone_active, zone))
6267                 zoneids[domi_nzones++] = zone->zone_id;
6268             ASSERT(domi_nzones == real_nzones);
6269         }
6270         mutex_exit(&zonehash_lock);
6271     }

6273     /*
6274     * If user has allocated space for fewer entries than we found, then
6275     * return only up to his limit. Either way, tell him exactly how many
6276     * we found.
6277     */
6278     if (domi_nzones < user_nzones)
6279         user_nzones = domi_nzones;
6280     error = 0;
6281     if (copyout(&domi_nzones, numzones, sizeof (uint_t)) != 0) {
6282         error = EFAULT;
6283     } else if (zoneidlist != NULL && user_nzones != 0) {
6284         if (copyout(zoneids, zoneidlist,
6285             user_nzones * sizeof (zoneid_t)) != 0)
6286             error = EFAULT;

```

```

6287     }
6289     if (real_nzones > 0)
6290         kmem_free(zoneids, real_nzones * sizeof (zoneid_t));
6292     if (error != 0)
6293         return (set_errno(error));
6294     else
6295         return (0);
6296 }
6298 /*
6299  * Systemcall entry point for zone_lookup(2).
6300  * Non-global zones are only able to see themselves and (on labeled systems)
6301  * the zones they dominate.
6302  */
6303 */
6304 static zoneid_t
6305 zone_lookup(const char *zone_name)
6306 {
6307     char *kname;
6308     zone_t *zone;
6309     zoneid_t zoneid;
6310     int err;
6312     if (zone_name == NULL) {
6313         /* return caller's zone id */
6314         return (getzoneid());
6315     }
6317     kname = kmem_zalloc(ZONENAME_MAX, KM_SLEEP);
6318     if ((err = copyinstr(zone_name, kname, ZONENAME_MAX, NULL)) != 0) {
6319         kmem_free(kname, ZONENAME_MAX);
6320         return (set_errno(err));
6321     }
6323     mutex_enter(&zonehash_lock);
6324     zone = zone_find_all_by_name(kname);
6325     kmem_free(kname, ZONENAME_MAX);
6326     /*
6327      * In a non-global zone, can only lookup global and own name.
6328      * In Trusted Extensions zone label dominance rules apply.
6329      */
6330     if (zone == NULL ||
6331         zone_status_get(zone) < ZONE_IS_READY ||
6332         !zone_list_access(zone)) {
6333         mutex_exit(&zonehash_lock);
6334         return (set_errno(EINVAL));
6335     } else {
6336         zoneid = zone->zone_id;
6337         mutex_exit(&zonehash_lock);
6338         return (zoneid);
6339     }
6340 }
6342 static int
6343 zone_version(int *version_arg)
6344 {
6345     int version = ZONE_SYSCALL_API_VERSION;
6347     if (copyout(&version, version_arg, sizeof (int)) != 0)
6348         return (set_errno(EFAULT));
6349     return (0);
6350 }
6352 /* ARGSUSED */

```

```

6353 long
6354 zone(int cmd, void *arg1, void *arg2, void *arg3, void *arg4)
6355 {
6356     zone_def zs;
6357     int err;
6359     switch (cmd) {
6360     case ZONE_CREATE:
6361         if (get_udatamodel() == DATAMODEL_NATIVE) {
6362             if (copyin(arg1, &zs, sizeof (zone_def))) {
6363                 return (set_errno(EFAULT));
6364             }
6365         } else {
6366             #ifdef _SYSCALL32_IMPL
6367                 zone_def32 zs32;
6369                 if (copyin(arg1, &zs32, sizeof (zone_def32))) {
6370                     return (set_errno(EFAULT));
6371                 }
6372                 zs.zone_name =
6373                     (const char *) (unsigned long) zs32.zone_name;
6374                 zs.zone_root =
6375                     (const char *) (unsigned long) zs32.zone_root;
6376                 zs.zone_privs =
6377                     (const struct priv_set *)
6378                     (unsigned long) zs32.zone_privs;
6379                 zs.zone_privssz = zs32.zone_privssz;
6380                 zs.rctlbuf = (caddr_t) (unsigned long) zs32.rctlbuf;
6381                 zs.rctlbufsz = zs32.rctlbufsz;
6382                 zs.zfsbuf = (caddr_t) (unsigned long) zs32.zfsbuf;
6383                 zs.zfsbufsz = zs32.zfsbufsz;
6384                 zs.extended_error =
6385                     (int *) (unsigned long) zs32.extended_error;
6386                 zs.match = zs32.match;
6387                 zs.doi = zs32.doi;
6388                 zs.label = (const bslabel_t *) (uintptr_t) zs32.label;
6389                 zs.flags = zs32.flags;
6390             #else
6391                 panic("get_udatamodel() returned bogus result\n");
6392             #endif
6393         }
6395         return (zone_create(zs.zone_name, zs.zone_root,
6396             zs.zone_privs, zs.zone_privssz,
6397             (caddr_t) zs.rctlbuf, zs.rctlbufsz,
6398             (caddr_t) zs.zfsbuf, zs.zfsbufsz,
6399             zs.extended_error, zs.match, zs.doi,
6400             zs.label, zs.flags));
6401     case ZONE_BOOT:
6402         return (zone_boot((zoneid_t) (uintptr_t) arg1));
6403     case ZONE_DESTROY:
6404         return (zone_destroy((zoneid_t) (uintptr_t) arg1));
6405     case ZONE_GETATTR:
6406         return (zone_getattr((zoneid_t) (uintptr_t) arg1,
6407             (int) (uintptr_t) arg2, arg3, (size_t) arg4));
6408     case ZONE_SETATTR:
6409         return (zone_setattr((zoneid_t) (uintptr_t) arg1,
6410             (int) (uintptr_t) arg2, arg3, (size_t) arg4));
6411     case ZONE_ENTER:
6412         return (zone_enter((zoneid_t) (uintptr_t) arg1));
6413     case ZONE_LIST:
6414         return (zone_list((zoneid_t *) arg1, (uint_t *) arg2));
6415     case ZONE_SHUTDOWN:
6416         return (zone_shutdown((zoneid_t) (uintptr_t) arg1));
6417     case ZONE_LOOKUP:
6418         return (zone_lookup((const char *) arg1));

```

```

6419     case ZONE_VERSION:
6420         return (zone_version((int *)arg1));
6421     case ZONE_ADD_DATALINK:
6422         return (zone_add_datalink((zoneid_t)(uintptr_t)arg1,
6423             (datalink_id_t)(uintptr_t)arg2));
6424     case ZONE_DEL_DATALINK:
6425         return (zone_remove_datalink((zoneid_t)(uintptr_t)arg1,
6426             (datalink_id_t)(uintptr_t)arg2));
6427     case ZONE_CHECK_DATALINK: {
6428         zoneid_t     zoneid;
6429         boolean_t    need_copyout;

6431         if (copyin(arg1, &zoneid, sizeof (zoneid)) != 0)
6432             return (EFAULT);
6433         need_copyout = (zoneid == ALL_ZONES);
6434         err = zone_check_datalink(&zoneid,
6435             (datalink_id_t)(uintptr_t)arg2);
6436         if (err == 0 && need_copyout) {
6437             if (copyout(&zoneid, arg1, sizeof (zoneid)) != 0)
6438                 err = EFAULT;
6439         }
6440         return (err == 0 ? 0 : set_errno(err));
6441     }
6442     case ZONE_LIST_DATALINK:
6443         return (zone_list_datalink((zoneid_t)(uintptr_t)arg1,
6444             (int *)arg2, (datalink_id_t *) (uintptr_t)arg3));
6445     default:
6446         return (set_errno(EINVAL));
6447 }
6448 }

6450 struct zarg {
6451     zone_t *zone;
6452     zone_cmd_arg_t arg;
6453 };

6455 static int
6456 zone_lookup_door(const char *zone_name, door_handle_t *doorp)
6457 {
6458     char *buf;
6459     size_t buflen;
6460     int error;

6462     buflen = sizeof (ZONE_DOOR_PATH) + strlen(zone_name);
6463     buf = kmem_alloc(buflen, KM_SLEEP);
6464     (void) snprintf(buf, buflen, ZONE_DOOR_PATH, zone_name);
6465     error = door_ki_open(buf, doorp);
6466     kmem_free(buf, buflen);
6467     return (error);
6468 }

6470 static void
6471 zone_release_door(door_handle_t *doorp)
6472 {
6473     door_ki_rele(*doorp);
6474     *doorp = NULL;
6475 }

6477 static void
6478 zone_ki_call_zoneadmd(struct zarg *zargp)
6479 {
6480     door_handle_t door = NULL;
6481     door_arg_t darg, save_arg;
6482     char *zone_name;
6483     size_t zone_namelen;
6484     zoneid_t zoneid;

```

```

6485     zone_t *zone;
6486     zone_cmd_arg_t arg;
6487     uint64_t uniqid;
6488     size_t size;
6489     int error;
6490     int retry;

6492     zone = zargp->zone;
6493     arg = zargp->arg;
6494     kmem_free(zargp, sizeof (*zargp));

6496     zone_namelen = strlen(zone->zone_name) + 1;
6497     zone_name = kmem_alloc(zone_namelen, KM_SLEEP);
6498     bcopy(zone->zone_name, zone_name, zone_namelen);
6499     zoneid = zone->zone_id;
6500     uniqid = zone->zone_uniqid;
6501     /*
6502      * zoneadmd may be down, but at least we can empty out the zone.
6503      * We can ignore the return value of zone_empty() since we're called
6504      * from a kernel thread and know we won't be delivered any signals.
6505      */
6506     ASSERT(curproc == &p0);
6507     (void) zone_empty(zone);
6508     ASSERT(zone_status_get(zone) >= ZONE_IS_EMPTY);
6509     zone_rele(zone);

6511     size = sizeof (arg);
6512     darg.rbuf = (char *)&arg;
6513     darg.data_ptr = (char *)&arg;
6514     darg.rsize = size;
6515     darg.data_size = size;
6516     darg.desc_ptr = NULL;
6517     darg.desc_num = 0;

6519     save_arg = darg;
6520     /*
6521      * Since we're not holding a reference to the zone, any number of
6522      * things can go wrong, including the zone disappearing before we get a
6523      * chance to talk to zoneadmd.
6524      */
6525     for (retry = 0; /* forever */; retry++) {
6526         if (door == NULL &&
6527             (error = zone_lookup_door(zone_name, &door)) != 0) {
6528             goto next;
6529         }
6530         ASSERT(door != NULL);

6532         if ((error = door_ki_upcall_limited(door, &darg, NULL,
6533             SIZE_MAX, 0)) == 0) {
6534             break;
6535         }
6536         switch (error) {
6537         case EINTR:
6538             /* FALLTHROUGH */
6539         case EAGAIN: /* process may be forking */
6540             /*
6541              * Back off for a bit
6542              */
6543             break;
6544         case EBADF:
6545             zone_release_door(&door);
6546             if (zone_lookup_door(zone_name, &door) != 0) {
6547                 /*
6548                  * zoneadmd may be dead, but it may come back to
6549                  * life later.
6550                  */

```

```

6551         break;
6552     }
6553     break;
6554     default:
6555         cmn_err(CE_WARN,
6556             "zone_ki_call_zoneadmd: door_ki_upcall error %d\n",
6557             error);
6558         goto out;
6559     }
6560 next:
6561     /*
6562     * If this isn't the same zone_t that we originally had in mind,
6563     * then this is the same as if two kadmin requests come in at
6564     * the same time: the first one wins. This means we lose, so we
6565     * bail.
6566     */
6567     if ((zone = zone_find_by_id(zoneid)) == NULL) {
6568         /*
6569         * Problem is solved.
6570         */
6571         break;
6572     }
6573     if (zone->zone_uniqid != uniqid) {
6574         /*
6575         * zoneid recycled
6576         */
6577         zone_rele(zone);
6578         break;
6579     }
6580     /*
6581     * We could zone_status_timedwait(), but there doesn't seem to
6582     * be much point in doing that (plus, it would mean that
6583     * zone_free() isn't called until this thread exits).
6584     */
6585     zone_rele(zone);
6586     delay(hz);
6587     darg = save_arg;
6588 }
6589 out:
6590     if (door != NULL) {
6591         zone_release_door(&door);
6592     }
6593     kmem_free(zone_name, zone_namelen);
6594     thread_exit();
6595 }
6597 /*
6598 * Entry point for uadmin() to tell the zone to go away or reboot. Analog to
6599 * kadmin(). The caller is a process in the zone.
6600 *
6601 * In order to shutdown the zone, we will hand off control to zoneadmd
6602 * (running in the global zone) via a door. We do a half-hearted job at
6603 * killing all processes in the zone, create a kernel thread to contact
6604 * zoneadmd, and make note of the "uniqid" of the zone. The uniqid is
6605 * a form of generation number used to let zoneadmd (as well as
6606 * zone_destroy()) know exactly which zone they're re talking about.
6607 */
6608 int
6609 zone_kadmin(int cmd, int fcn, const char *mdep, cred_t *credp)
6610 {
6611     struct zarg *zargp;
6612     zone_cmd_t zcmd;
6613     zone_t *zone;
6615     zone = curproc->p_zone;
6616     ASSERT(getzoneid() != GLOBAL_ZONEID);

```

```

6618     switch (cmd) {
6619     case A_SHUTDOWN:
6620         switch (fcn) {
6621         case AD_HALT:
6622             case AD_POWEROFF:
6623                 zcmd = Z_HALT;
6624                 break;
6625         case AD_BOOT:
6626             zcmd = Z_REBOOT;
6627             break;
6628         case AD_IBOOT:
6629             case AD_SBOOT:
6630             case AD_SIBOOT:
6631             case AD_NOSYNC:
6632                 return (ENOTSUP);
6633         default:
6634             return (EINVAL);
6635         }
6636         break;
6637     case A_REBOOT:
6638         zcmd = Z_REBOOT;
6639         break;
6640     case A_FTRACE:
6641     case A_REMOUNT:
6642     case A_FREEZE:
6643     case A_DUMP:
6644     case A_CONFIG:
6645         return (ENOTSUP);
6646     default:
6647         ASSERT(cmd != A_SWAPCTL);          /* handled by uadmin() */
6648         return (EINVAL);
6649     }
6651     if (secpolicy_zone_admin(credp, B_FALSE))
6652         return (EPERM);
6653     mutex_enter(&zone_status_lock);
6655     /*
6656     * zone_status can't be ZONE_IS_EMPTY or higher since curproc
6657     * is in the zone.
6658     */
6659     ASSERT(zone_status_get(zone) < ZONE_IS_EMPTY);
6660     if (zone_status_get(zone) > ZONE_IS_RUNNING) {
6661         /*
6662         * This zone is already on its way down.
6663         */
6664         mutex_exit(&zone_status_lock);
6665         return (0);
6666     }
6667     /*
6668     * Prevent future zone_enter()s
6669     */
6670     zone_status_set(zone, ZONE_IS_SHUTTING_DOWN);
6671     mutex_exit(&zone_status_lock);
6673     /*
6674     * Kill everyone now and call zoneadmd later.
6675     * zone_ki_call_zoneadmd() will do a more thorough job of this
6676     * later.
6677     */
6678     killall(zone->zone_id);
6679     /*
6680     * Now, create the thread to contact zoneadmd and do the rest of the
6681     * work. This thread can't be created in our zone otherwise
6682     * zone_destroy() would deadlock.

```

```

6683  */
6684  zargp = kmem_zalloc(sizeof (*zargp), KM_SLEEP);
6685  zargp->arg.cmd = zcmd;
6686  zargp->arg.uniqlid = zone->zone_uniqlid;
6687  zargp->zone = zone;
6688  (void) strcpy(zargp->arg.locale, "C");
6689  /* mdep was already copied in for us by uadmin */
6690  if (mdep != NULL)
6691      (void) strcpy(zargp->arg.bootbuf, mdep,
6692                  sizeof (zargp->arg.bootbuf));
6693  zone_hold(zone);

6695  (void) thread_create(NULL, 0, zone_ki_call_zoneadmd, zargp, 0, &p0,
6696                      TS_RUN, minclsyspri);
6697  exit(CLD_EXITED, 0);

6699  return (EINVAL);
6700 }

6702 /*
6703  * Entry point so kadmind(A_SHUTDOWN, ...) can set the global zone's
6704  * status to ZONE_IS_SHUTTING_DOWN.
6705  *
6706  * This function also shuts down all running zones to ensure that they won't
6707  * fork new processes.
6708  */
6709 void
6710 zone_shutdown_global(void)
6711 {
6712     zone_t *current_zonep;

6714     ASSERT(INGLOBALZONE(curproc));
6715     mutex_enter(&zonehash_lock);
6716     mutex_enter(&zone_status_lock);

6718     /* Modify the global zone's status first. */
6719     ASSERT(zone_status_get(global_zone) == ZONE_IS_RUNNING);
6720     zone_status_set(global_zone, ZONE_IS_SHUTTING_DOWN);

6722     /*
6723      * Now change the states of all running zones to ZONE_IS_SHUTTING_DOWN.
6724      * We don't mark all zones with ZONE_IS_SHUTTING_DOWN because doing so
6725      * could cause assertions to fail (e.g., assertions about a zone's
6726      * state during initialization, readying, or booting) or produce races.
6727      * We'll let threads continue to initialize and ready new zones: they'll
6728      * fail to boot the new zones when they see that the global zone is
6729      * shutting down.
6730      */
6731     for (current_zonep = list_head(&zone_active); current_zonep != NULL;
6732          current_zonep = list_next(&zone_active, current_zonep)) {
6733         if (zone_status_get(current_zonep) == ZONE_IS_RUNNING)
6734             zone_status_set(current_zonep, ZONE_IS_SHUTTING_DOWN);
6735     }
6736     mutex_exit(&zone_status_lock);
6737     mutex_exit(&zonehash_lock);
6738 }

6740 /*
6741  * Returns true if the named dataset is visible in the current zone.
6742  * The 'write' parameter is set to 1 if the dataset is also writable.
6743  */
6744 int
6745 zone_dataset_visible(const char *dataset, int *write)
6746 {
6747     static int zfstype = -1;
6748     zone_dataset_t *zd;

```

```

6749     size_t len;
6750     zone_t *zone = curproc->p_zone;
6751     const char *name = NULL;
6752     vfs_t *vfswp = NULL;

6754     if (dataset[0] == '\0')
6755         return (0);

6757     /*
6758      * Walk the list once, looking for datasets which match exactly, or
6759      * specify a dataset underneath an exported dataset. If found, return
6760      * true and note that it is writable.
6761      */
6762     for (zd = list_head(&zone->zone_datasets); zd != NULL;
6763          zd = list_next(&zone->zone_datasets, zd)) {

6765         len = strlen(zd->zd_dataset);
6766         if (strlen(dataset) >= len &&
6767             bcmp(dataset, zd->zd_dataset, len) == 0 &&
6768             (dataset[len] == '\0' || dataset[len] == '/' ||
6769              dataset[len] == '@')) {
6770             if (write)
6771                 *write = 1;
6772             return (1);
6773         }
6774     }

6776     /*
6777      * Walk the list a second time, searching for datasets which are parents
6778      * of exported datasets. These should be visible, but read-only.
6779      *
6780      * Note that we also have to support forms such as 'pool/dataset/', with
6781      * a trailing slash.
6782      */
6783     for (zd = list_head(&zone->zone_datasets); zd != NULL;
6784          zd = list_next(&zone->zone_datasets, zd)) {

6786         len = strlen(dataset);
6787         if (dataset[len - 1] == '/')
6788             len--; /* Ignore trailing slash */
6789         if (len < strlen(zd->zd_dataset) &&
6790             bcmp(dataset, zd->zd_dataset, len) == 0 &&
6791             zd->zd_dataset[len] == '/') {
6792             if (write)
6793                 *write = 0;
6794             return (1);
6795         }
6796     }

6798     /*
6799      * We reach here if the given dataset is not found in the zone_dataset
6800      * list. Check if this dataset was added as a filesystem (ie. "add fs")
6801      * instead of delegation. For this we search for the dataset in the
6802      * zone_vfswp of this zone. If found, return true and note that it is
6803      * not writable.
6804      */

6806     /*
6807      * Initialize zfstype if it is not initialized yet.
6808      */
6809     if (zfstype == -1) {
6810         struct vfswp *vswp = vfswp_getvfswp("zfs");
6811         zfstype = vswp - vfswp;
6812         vfswp_unrefvfswp(vswp);
6813     }

```

```

6815     vfs_list_read_lock();
6816     vfsp = zone->zone_vfslst;
6817     do {
6818         ASSERT(vfsp);
6819         if (vfsp->vfs_fstype == zfstype) {
6820             name = refstr_value(vfsp->vfs_resource);
6821
6822             /*
6823              * Check if we have an exact match.
6824              */
6825             if (strcmp(dataset, name) == 0) {
6826                 vfs_list_unlock();
6827                 if (write)
6828                     *write = 0;
6829                 return (1);
6830             }
6831             /*
6832              * We need to check if we are looking for parents of
6833              * a dataset. These should be visible, but read-only.
6834              */
6835             len = strlen(dataset);
6836             if (dataset[len - 1] == '/')
6837                 len--;
6838
6839             if (len < strlen(name) &&
6840                 bcmp(dataset, name, len) == 0 && name[len] == '/') {
6841                 vfs_list_unlock();
6842                 if (write)
6843                     *write = 0;
6844                 return (1);
6845             }
6846             vfsp = vfsp->vfs_zone_next;
6847         } while (vfsp != zone->zone_vfslst);
6848
6849     } while (1);
6850     vfs_list_unlock();
6851     return (0);
6852 }
6853
6854 /*
6855  * zone_find_by_any_path() -
6856  *
6857  * kernel-private routine similar to zone_find_by_path(), but which
6858  * effectively compares against zone paths rather than zonerootpath
6859  * (i.e., the last component of zonerootpaths, which should be "root/",
6860  * are not compared.) This is done in order to accurately identify all
6861  * paths, whether zone-visible or not, including those which are parallel
6862  * to /root/, such as /dev/, /home/, etc...
6863  *
6864  * If the specified path does not fall under any zone path then global
6865  * zone is returned.
6866  *
6867  * The treat_abs parameter indicates whether the path should be treated as
6868  * an absolute path although it does not begin with "/". (This supports
6869  * nfs mount syntax such as host:any/path.)
6870  *
6871  * The caller is responsible for zone_rele of the returned zone.
6872  */
6873 zone_t *
6874 zone_find_by_any_path(const char *path, boolean_t treat_abs)
6875 {
6876     zone_t *zone;
6877     int path_offset = 0;
6878
6879     if (path == NULL) {
6880         zone_hold(global_zone);

```

```

6881         return (global_zone);
6882     }
6883
6884     if (*path != '/') {
6885         ASSERT(treat_abs);
6886         path_offset = 1;
6887     }
6888
6889     mutex_enter(&zonehash_lock);
6890     for (zone = list_head(&zone_active); zone != NULL;
6891          zone = list_next(&zone_active, zone)) {
6892         char *c;
6893         size_t pathlen;
6894         char *rootpath_start;
6895
6896         if (zone == global_zone) /* skip global zone */
6897             continue;
6898
6899         /* scan backwards to find start of last component */
6900         c = zone->zone_rootpath + zone->zone_rootpathlen - 2;
6901         do {
6902             c--;
6903         } while (*c != '/');
6904
6905         pathlen = c - zone->zone_rootpath + 1 - path_offset;
6906         rootpath_start = (zone->zone_rootpath + path_offset);
6907         if (strncmp(path, rootpath_start, pathlen) == 0)
6908             break;
6909     }
6910     if (zone == NULL)
6911         zone = global_zone;
6912     zone_hold(zone);
6913     mutex_exit(&zonehash_lock);
6914     return (zone);
6915 }
6916
6917 /*
6918  * Finds a zone_dl_t with the given linkid in the given zone. Returns the
6919  * zone_dl_t pointer if found, and NULL otherwise.
6920  */
6921 static zone_dl_t *
6922 zone_find_dl(zone_t *zone, datalink_id_t linkid)
6923 {
6924     zone_dl_t *zdl;
6925
6926     ASSERT(mutex_owned(&zone->zone_lock));
6927     for (zdl = list_head(&zone->zone_dl_list); zdl != NULL;
6928          zdl = list_next(&zone->zone_dl_list, zdl)) {
6929         if (zdl->zdl_id == linkid)
6930             break;
6931     }
6932     return (zdl);
6933 }
6934
6935 static boolean_t
6936 zone_dl_exists(zone_t *zone, datalink_id_t linkid)
6937 {
6938     boolean_t exists;
6939
6940     mutex_enter(&zone->zone_lock);
6941     exists = (zone_find_dl(zone, linkid) != NULL);
6942     mutex_exit(&zone->zone_lock);
6943     return (exists);
6944 }
6945
6946 /*

```



```

6947 * Add an data link name for the zone.
6948 */
6949 static int
6950 zone_add_datalink(zoneid_t zoneid, datalink_id_t linkid)
6951 {
6952     zone_dl_t *zdl;
6953     zone_t *zone;
6954     zone_t *thiszone;
6955
6956     if ((thiszone = zone_find_by_id(zoneid)) == NULL)
6957         return (set_errno(ENXIO));
6958
6959     /* Verify that the datalink ID doesn't already belong to a zone. */
6960     mutex_enter(&zonehash_lock);
6961     for (zone = list_head(&zone_active); zone != NULL;
6962          zone = list_next(&zone_active, zone)) {
6963         if (zone_dl_exists(zone, linkid)) {
6964             mutex_exit(&zonehash_lock);
6965             zone_rele(thiszone);
6966             return (set_errno((zone == thiszone) ? EEXIST : EPERM));
6967         }
6968     }
6969
6970     zdl = kmem_zalloc(sizeof (*zdl), KM_SLEEP);
6971     zdl->zdl_id = linkid;
6972     zdl->zdl_net = NULL;
6973     mutex_enter(&thiszone->zone_lock);
6974     list_insert_head(&thiszone->zone_dl_list, zdl);
6975     mutex_exit(&thiszone->zone_lock);
6976     mutex_exit(&zonehash_lock);
6977     zone_rele(thiszone);
6978     return (0);
6979 }
6980
6981 static int
6982 zone_remove_datalink(zoneid_t zoneid, datalink_id_t linkid)
6983 {
6984     zone_dl_t *zdl;
6985     zone_t *zone;
6986     int err = 0;
6987
6988     if ((zone = zone_find_by_id(zoneid)) == NULL)
6989         return (set_errno(EINVAL));
6990
6991     mutex_enter(&zone->zone_lock);
6992     if ((zdl = zone_find_dl(zone, linkid)) == NULL) {
6993         err = ENXIO;
6994     } else {
6995         list_remove(&zone->zone_dl_list, zdl);
6996         nvlist_free(zdl->zdl_net);
6997         kmem_free(zdl, sizeof (zone_dl_t));
6998     }
6999     mutex_exit(&zone->zone_lock);
7000     zone_rele(zone);
7001     return (err == 0 ? 0 : set_errno(err));
7002 }
7003
7004 /*
7005  * Using the zoneidp as ALL_ZONES, we can lookup which zone has been assigned
7006  * the linkid. Otherwise we just check if the specified zoneidp has been
7007  * assigned the supplied linkid.
7008  */
7009 int
7010 zone_check_datalink(zoneid_t *zoneidp, datalink_id_t linkid)
7011 {
7012     zone_t *zone;

```

```

7013     int err = ENXIO;
7014
7015     if (*zoneidp != ALL_ZONES) {
7016         if ((zone = zone_find_by_id(*zoneidp)) != NULL) {
7017             if (zone_dl_exists(zone, linkid))
7018                 err = 0;
7019             zone_rele(zone);
7020         }
7021         return (err);
7022     }
7023
7024     mutex_enter(&zonehash_lock);
7025     for (zone = list_head(&zone_active); zone != NULL;
7026          zone = list_next(&zone_active, zone)) {
7027         if (zone_dl_exists(zone, linkid)) {
7028             *zoneidp = zone->zone_id;
7029             err = 0;
7030             break;
7031         }
7032     }
7033     mutex_exit(&zonehash_lock);
7034     return (err);
7035 }
7036
7037 /*
7038  * Get the list of datalink IDs assigned to a zone.
7039  *
7040  * On input, *nump is the number of datalink IDs that can fit in the supplied
7041  * idarray. Upon return, *nump is either set to the number of datalink IDs
7042  * that were placed in the array if the array was large enough, or to the
7043  * number of datalink IDs that the function needs to place in the array if the
7044  * array is too small.
7045  */
7046 static int
7047 zone_list_datalink(zoneid_t zoneid, int *nump, datalink_id_t *idarray)
7048 {
7049     uint_t num, dlcount;
7050     zone_t *zone;
7051     zone_dl_t *zdl;
7052     datalink_id_t *idptr = idarray;
7053
7054     if (copyin(nump, &dlcount, sizeof (dlcount)) != 0)
7055         return (set_errno(EFAULT));
7056     if ((zone = zone_find_by_id(zoneid)) == NULL)
7057         return (set_errno(ENXIO));
7058
7059     num = 0;
7060     mutex_enter(&zone->zone_lock);
7061     for (zdl = list_head(&zone->zone_dl_list); zdl != NULL;
7062          zdl = list_next(&zone->zone_dl_list, zdl)) {
7063         /*
7064          * If the list is bigger than what the caller supplied, just
7065          * count, don't do copyout.
7066          */
7067         if (++num > dlcount)
7068             continue;
7069         if (copyout(&zdl->zdl_id, idptr, sizeof (*idptr)) != 0) {
7070             mutex_exit(&zone->zone_lock);
7071             zone_rele(zone);
7072             return (set_errno(EFAULT));
7073         }
7074         idptr++;
7075     }
7076     mutex_exit(&zone->zone_lock);
7077     zone_rele(zone);

```

```

7079      /* Increased or decreased, caller should be notified. */
7080      if (num != dlcount) {
7081          if (copyout(&num, nump, sizeof(num)) != 0)
7082              return (set_errno(EFAULT));
7083      }
7084      return (0);
7085 }

7087 /*
7088  * Public interface for looking up a zone by zoneid. It's a customized version
7089  * for netstack_zone_create(). It can only be called from the zsd create
7090  * callbacks, since it doesn't have reference on the zone structure hence if
7091  * it is called elsewhere the zone could disappear after the zonehash_lock
7092  * is dropped.
7093  *
7094  * Furthermore it
7095  * 1. Doesn't check the status of the zone.
7096  * 2. It will be called even before zone_init is called, in that case the
7097  *    address of zone0 is returned directly, and netstack_zone_create()
7098  *    will only assign a value to zone0.zone_netstack, won't break anything.
7099  * 3. Returns without the zone being held.
7100  */
7101 zone_t *
7102 zone_find_by_id_nolock(zoneid_t zoneid)
7103 {
7104     zone_t *zone;

7106     mutex_enter(&zonehash_lock);
7107     if (zonehashbyid == NULL)
7108         zone = &zone0;
7109     else
7110         zone = zone_find_all_by_id(zoneid);
7111     mutex_exit(&zonehash_lock);
7112     return (zone);
7113 }

7115 /*
7116  * Walk the datalinks for a given zone
7117  */
7118 int
7119 zone_datalink_walk(zoneid_t zoneid, int (*cb)(datalink_id_t, void *),
7120 void *data)
7121 {
7122     zone_t *zone;
7123     zone_dl_t *zdl;
7124     datalink_id_t *idarray;
7125     uint_t idcount = 0;
7126     int i, ret = 0;

7128     if ((zone = zone_find_by_id(zoneid)) == NULL)
7129         return (ENOENT);

7131     /*
7132      * We first build an array of linkid's so that we can walk these and
7133      * execute the callback with the zone_lock dropped.
7134      */
7135     mutex_enter(&zone->zone_lock);
7136     for (zdl = list_head(&zone->zone_dl_list); zdl != NULL;
7137          zdl = list_next(&zone->zone_dl_list, zdl)) {
7138         idcount++;
7139     }

7141     if (idcount == 0) {
7142         mutex_exit(&zone->zone_lock);
7143         zone_rele(zone);
7144         return (0);

```

```

7145     }

7147     idarray = kmem_alloc(sizeof(datalink_id_t) * idcount, KM_NOSLEEP);
7148     if (idarray == NULL) {
7149         mutex_exit(&zone->zone_lock);
7150         zone_rele(zone);
7151         return (ENOMEM);
7152     }

7154     for (i = 0, zdl = list_head(&zone->zone_dl_list); zdl != NULL;
7155          i++, zdl = list_next(&zone->zone_dl_list, zdl)) {
7156         idarray[i] = zdl->zdl_id;
7157     }

7159     mutex_exit(&zone->zone_lock);

7161     for (i = 0; i < idcount && ret == 0; i++) {
7162         if ((ret = (*cb)(idarray[i], data)) != 0)
7163             break;
7164     }

7166     zone_rele(zone);
7167     kmem_free(idarray, sizeof(datalink_id_t) * idcount);
7168     return (ret);
7169 }

7171 static char *
7172 zone_net_type2name(int type)
7173 {
7174     switch (type) {
7175     case ZONE_NETWORK_ADDRESS:
7176         return (ZONE_NET_ADDRNAME);
7177     case ZONE_NETWORK_DEFROUTER:
7178         return (ZONE_NET_RTRNAME);
7179     default:
7180         return (NULL);
7181     }
7182 }

7184 static int
7185 zone_set_network(zoneid_t zoneid, zone_net_data_t *znbuf)
7186 {
7187     zone_t *zone;
7188     zone_dl_t *zdl;
7189     nvlist_t *nvl;
7190     int err = 0;
7191     uint8_t *new = NULL;
7192     char *nvname;
7193     int bufsize;
7194     datalink_id_t linkid = znbuf->zn_linkid;

7196     if (secpolicy_zone_config(CRED()) != 0)
7197         return (set_errno(EPERM));

7199     if (zoneid == GLOBAL_ZONEID)
7200         return (set_errno(EINVAL));

7202     nvname = zone_net_type2name(znbuf->zn_type);
7203     bufsize = znbuf->zn_len;
7204     new = znbuf->zn_val;
7205     if (nvname == NULL)
7206         return (set_errno(EINVAL));

7208     if ((zone = zone_find_by_id(zoneid)) == NULL) {
7209         return (set_errno(EINVAL));
7210     }

```

```

7212     mutex_enter(&zone->zone_lock);
7213     if ((zdl = zone_find_dl(zone, linkid)) == NULL) {
7214         err = ENXIO;
7215         goto done;
7216     }
7217     if ((nvl = zdl->zdl_net) == NULL) {
7218         if (nvlst_alloc(&nvl, NV_UNIQUE_NAME, KM_SLEEP)) {
7219             err = ENOMEM;
7220             goto done;
7221         } else {
7222             zdl->zdl_net = nvl;
7223         }
7224     }
7225     if (nvlst_exists(nvl, nvname)) {
7226         err = EINVAL;
7227         goto done;
7228     }
7229     err = nvlst_add_uint8_array(nvl, nvname, new, bufsize);
7230     ASSERT(err == 0);
7231 done:
7232     mutex_exit(&zone->zone_lock);
7233     zone_rele(zone);
7234     if (err != 0)
7235         return (set_errno(err));
7236     else
7237         return (0);
7238 }

7240 static int
7241 zone_get_network(zoneid_t zoneid, zone_net_data_t *znbuf)
7242 {
7243     zone_t *zone;
7244     zone_dl_t *zdl;
7245     nvlst_t *nvl;
7246     uint8_t *ptr;
7247     uint_t psize;
7248     int err = 0;
7249     char *nvname;
7250     int bufsize;
7251     void *buf;
7252     datalink_id_t linkid = znbuf->zn_linkid;

7254     if (zoneid == GLOBAL_ZONEID)
7255         return (set_errno(EINVAL));

7257     nvname = zone_net_type2name(znbuf->zn_type);
7258     bufsize = znbuf->zn_len;
7259     buf = znbuf->zn_val;

7261     if (nvname == NULL)
7262         return (set_errno(EINVAL));
7263     if ((zone = zone_find_by_id(zoneid)) == NULL)
7264         return (set_errno(EINVAL));

7266     mutex_enter(&zone->zone_lock);
7267     if ((zdl = zone_find_dl(zone, linkid)) == NULL) {
7268         err = ENXIO;
7269         goto done;
7270     }
7271     if ((nvl = zdl->zdl_net) == NULL || !nvlst_exists(nvl, nvname)) {
7272         err = ENOENT;
7273         goto done;
7274     }
7275     err = nvlst_lookup_uint8_array(nvl, nvname, &ptr, &psize);
7276     ASSERT(err == 0);

```

```

7278     if (psize > bufsize) {
7279         err = ENOBUFS;
7280         goto done;
7281     }
7282     znbuf->zn_len = psize;
7283     bcopy(ptr, buf, psize);
7284 done:
7285     mutex_exit(&zone->zone_lock);
7286     zone_rele(zone);
7287     if (err != 0)
7288         return (set_errno(err));
7289     else
7290         return (0);
7291 }

```

```

*****
22223 Thu Jun 30 22:03:37 2016
new/usr/src/uts/common/sys/Makefile
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
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 #
23 # Copyright (c) 1989, 2010, Oracle and/or its affiliates. All rights reserved.
24 # Copyright 2014, Joyent, Inc. All rights reserved.
25 # Copyright 2013 Garrett D'Amore <garrett@damore.org>
26 # Copyright 2013 Saso Kiselkov. All rights reserved.
27 # Copyright 2015 Igor Kozhukhov <ikozhukhov@gmail.com>
28 # Copyright 2016 Nexenta Systems, Inc.
29 #
30 #
31 include $(SRC)/uts/Makefile.uts
32 #
33 FILEMODE=644
34 #
35 #
36 # Note that the following headers are present in the kernel but
37 # neither installed or shipped as part of the product:
38 # cpuid_drv.h: Private interface for cpuid consumers
39 # unix_bb_info.h: Private interface to kcov
40 #
41 #
42 i386_HDRS= \
43     agp/agpamd64gart_io.h \
44     agp/agpdefs.h \
45     agp/agpgart_impl.h \
46     agp/agpmaster_io.h \
47     agp/agptarget_io.h \
48     agpgart.h \
49     asy.h \
50     fd_debug.h \
51     fdc.h \
52     fdmedia.h \
53     mouse.h \
54     ucode.h \
55 #
56 sparc_HDRS= \
57     mouse.h \
58     scsi/targets/ssddef.h \

```

```

59     $(MDESCHDRS)
60 #
61 # Generated headers
62 GENHDRS= \
63     priv_const.h \
64     priv_names.h \
65     usb/usbdevs.h \
66 #
67 CHKHDRS= \
68     acpi_drv.h \
69     acct.h \
70     acctctl.h \
71     acl.h \
72     acl_impl.h \
73     aggr.h \
74     aggr_impl.h \
75     aio.h \
76     aio_impl.h \
77     aio_req.h \
78     aiocb.h \
79     ascii.h \
80     asynch.h \
81     atomic.h \
82     attr.h \
83     audio.h \
84     audioio.h \
85     autoconf.h \
86     auxv.h \
87     auxv_386.h \
88     auxv_SPARC.h \
89     avl.h \
90     avl_impl.h \
91     bitmap.h \
92     bitset.h \
93     bl.h \
94     blkdev.h \
95     bofi.h \
96     bofi_impl.h \
97     bpp_io.h \
98     bootstat.h \
99     brand.h \
100    buf.h \
101    bufmod.h \
102    bustypes.h \
103    byteorder.h \
104    callb.h \
105    callo.h \
106    cap_util.h \
107    cpucaps.h \
108    cpucaps_impl.h \
109    ccompile.h \
110    cdio.h \
111    cladm.h \
112    class.h \
113    clconf.h \
114    clock_impl.h \
115    cmlb.h \
116    cmn_err.h \
117    compress.h \
118    condvar.h \
119    condvar_impl.h \
120    conf.h \
121    consdev.h \
122    console.h \
123    consplat.h \
124    vt.h \

```

```

125     vtdaemon.h      \|
126     kd.h            \|
127     contract.h     \|
128     contract_impl.h \|
129     copyops.h      \|
130     core.h         \|
131     corectl.h      \|
132     cpc_impl.h     \|
133     cpc_pcbe.h     \|
134     cpr.h          \|
135     cpupart.h      \|
136     cpuvar.h       \|
137     crc32.h        \|
138     cred.h         \|
139     cred_impl.h   \|
140     crtctl.h       \|
141     cryptmod.h     \|
142     csioctl.h     \|
143     ctf.h          \|
144     ctfs.h         \|
145     ctfs_impl.h   \|
146     ctf_api.h     \|
147     ctype.h        \|
148     cyclic.h       \|
149     cyclic_impl.h \|
150     dacf.h         \|
151     dacf_impl.h   \|
152     damap.h        \|
153     damap_impl.h  \|
154     dc_ki.h        \|
155     ddi.h          \|
156     ddifm.h        \|
157     ddifm_impl.h  \|
158     ddi_hp.h       \|
159     ddi_hp_impl.h \|
160     ddi_intr.h     \|
161     ddi_intr_impl.h \|
162     ddi_impldefs.h \|
163     ddi_implfuncs.h \|
164     ddi_obsolete.h \|
165     ddi_periodic.h \|
166     ddidevmap.h   \|
167     ddidmareq.h   \|
168     ddimapreq.h   \|
169     ddipropdefs.h \|
170     ddiypes.h     \|
171     debug.h        \|
172     des.h          \|
173     devctl.h      \|
174     devcache.h    \|
175     devcache_impl.h \|
176     devfm.h       \|
177     devid_cache.h \|
178     devinfo_impl.h \|
179     devops.h       \|
180     devpolicy.h   \|
181     devpoll.h     \|
182     dirent.h      \|
183     disp.h        \|
184     dkbad.h       \|
185     dkio.h        \|
186     dklablel.h   \|
187     dl.h          \|
188     dlpi.h        \|
189     dld.h         \|
190     dld_impl.h    \|

```

```

191     dld_ioc.h      \|
192     dls.h          \|
193     dls_mgmt.h    \|
194     dls_impl.h    \|
195     dma_i8237A.h  \|
196     dnlc.h        \|
197     door.h        \|
198     door_data.h   \|
199     door_impl.h   \|
200     dtrace.h      \|
201     dtrace_impl.h \|
202     dumpadm.h     \|
203     dumphdr.h     \|
204     ecppsys.h     \|
205     ecppio.h      \|
206     ecppreg.h     \|
207     ecppvar.h     \|
208     edonr.h       \|
209     efi_partition.h \|
210     elf.h         \|
211     elf_386.h     \|
212     elf_SPARC.h   \|
213     elf_notes.h   \|
214     elf_amd64.h   \|
215     elftypes.h    \|
216     emul64.h      \|
217     emul64cmd.h   \|
218     emul64var.h   \|
219     epm.h         \|
220     epoll.h       \|
221     errno.h       \|
222     errorq.h      \|
223     errorq_impl.h \|
224     esunddi.h     \|
225     ethernet.h    \|
226     euc.h         \|
227     eucioctl.h    \|
228     eventfd.h     \|
229     exacct.h      \|
230     exacct_catalog.h \|
231     exacct_impl.h \|
232     exec.h        \|
233     exechnr.h     \|
234     extdirent.h   \|
235     fault.h       \|
236     fasttrap.h    \|
237     fasttrap_impl.h \|
238     fbio.h        \|
239     fbuf.h        \|
240     fcntl.h       \|
241     fct.h         \|
242     fct_defines.h \|
243     fctio.h       \|
244     fdbuffer.h    \|
245     fdio.h        \|
246     feature_tests.h \|
247     fem.h         \|
248     file.h        \|
249     filio.h       \|
250     flock.h       \|
251     flock_impl.h  \|
252     fork.h        \|
253     fss.h         \|
254     fssprioctl.h  \|
255     fsid.h        \|
256     fssnap.h      \|

```

```

257     fssnap_if.h      \
258     fstyp.h         \
259     ftrace.h        \
260     fx.h            \
261     fxpriocntl.h   \
262     gfs.h           \
263     gld.h           \
264     gldpriv.h      \
265     group.h         \
266     hdio.h          \
267     hook.h          \
268     hook_event.h   \
269     hook_impl.h    \
270     hwconf.h       \
271     ia.h            \
272     iapriocntl.h   \
273     ibpart.h       \
274     id32.h          \
275     idmap.h         \
276     ieeeefp.h      \
277     id_space.h     \
278     instance.h     \
279     int_const.h    \
280     int_fmtio.h    \
281     int_limits.h   \
282     int_types.h    \
283     inttypes.h     \
284     ioccom.h       \
285     ioctl.h        \
286     ipc.h           \
287     ipc_impl.h     \
288     ipc_rctl.h     \
289     ipd.h           \
290     ipmi.h          \
291     isa_defs.h     \
292     iscsi_authclient.h \
293     iscsi_authclientglue.h \
294     iscsi_protocol.h \
295     jioctl.h       \
296     kbd.h           \
297     kbdreg.h       \
298     kbio.h         \
299     kcpic.h        \
300     kdi.h          \
301     kdi_impl.h    \
302     kiconv.h       \
303     kiconv_big5_utf8.h \
304     kiconv_cck_common.h \
305     kiconv_cp950hkscs_utf8.h \
306     kiconv_emeal.h \
307     kiconv_emea2.h \
308     kiconv_euckr_utf8.h \
309     kiconv_euctw_utf8.h \
310     kiconv_gb18030_utf8.h \
311     kiconv_gb2312_utf8.h \
312     kiconv_hkscs_utf8.h \
313     kiconv_ja.h   \
314     kiconv_ja_jis_to_unicode.h \
315     kiconv_ja_unicode_to_jis.h \
316     kiconv_ko.h   \
317     kiconv_latin1.h \
318     kiconv_sc.h   \
319     kiconv_tc.h   \
320     kiconv_uhc_utf8.h \
321     kiconv_utf8_big5.h \
322     kiconv_utf8_cp950hkscs.h \

```

```

323     kiconv_utf8_euckr.h \
324     kiconv_utf8_euctw.h \
325     kiconv_utf8_gb18030.h \
326     kiconv_utf8_gb2312.h \
327     kiconv_utf8_hkscs.h \
328     kiconv_utf8_uhc.h \
329     kidmap.h          \
330     klpd.h           \
331     klwp.h           \
332     kmdb.h           \
333     kmem.h           \
334     kmem_impl.h     \
335     kobj.h           \
336     kobj_impl.h     \
337     ksocket.h       \
338     kstat.h         \
339     kstr.h           \
340     ksyms.h         \
341     ksynch.h        \
342     ldterm.h        \
343     lgrp.h           \
344     lgrp_user.h     \
345     libc_kernel.h   \
346     link.h           \
347     list.h           \
348     list_impl.h     \
349     llc1.h           \
350     loadavg.h       \
351     lock.h           \
352     lockfs.h        \
353     lockstat.h     \
354     lofi.h           \
355     log.h            \
356     loginmux.h      \
357     loginmux_impl.h \
358     lwp.h            \
359     lwp_timer_impl.h \
360     lwp_upimutex_impl.h \
361     lpif.h           \
362     mac.h            \
363     mac_client.h    \
364     mac_client_impl.h \
365     mac_ether.h     \
366     mac_flow.h      \
367     mac_flow_impl.h \
368     mac_impl.h      \
369     mac_provider.h  \
370     mac_soft_ring.h \
371     mac_stat.h      \
372     machelf.h       \
373     map.h            \
374     md4.h            \
375     md5.h            \
376     md5_consts.h    \
377     mdi_impldefs.h  \
378     mem.h            \
379     mem_config.h    \
380     memlist.h       \
381     mkdev.h         \
382     mhd.h           \
383     mi.h             \
384     miiregs.h       \
385     mixer.h         \
386     mman.h          \
387     mmapobj.h       \
388     mntent.h        \

```

```

389     mntio.h          \|
390     mnttab.h         \|
391     modctl.h         \|
392     mode.h           \|
393     model.h          \|
394     modhash.h        \|
395     modhash_impl.h  \|
396     mount.h          \|
397     mouse.h          \|
398     msacct.h         \|
399     msg.h            \|
400     msg_impl.h       \|
401     msio.h           \|
402     msreg.h          \|
403     mtio.h           \|
404     multidata.h      \|
405     multidata_impl.h \|
406     mutex.h          \|
407     nbmlock.h        \|
408     ndifm.h          \|
409     ndi_impldefs.h  \|
410     net80211.h       \|
411     net80211_crypto.h \|
412     net80211_ht.h   \|
413     net80211_proto.h \|
414     netconfig.h      \|
415     neti.h           \|
416     netstack.h       \|
417     nexusdefs.h     \|
418     note.h           \|
419     null.h           \|
420     nvpair.h         \|
421     nvpair_impl.h   \|
422     objfs.h          \|
423     objfs_impl.h     \|
424     ontrap.h         \|
425     open.h           \|
426     openpromio.h    \|
427     panic.h          \|
428     param.h          \|
429     pathconf.h       \|
430     pathname.h       \|
431     pattn.h          \|
432     queue.h          \|
433     serializer.h     \|
434     pbio.h           \|
435     pccard.h         \|
436     pci.h            \|
437     pcie.h           \|
438     pci_impl.h       \|
439     pci_tools.h      \|
440     pcmcia.h         \|
441     ptypes.h         \|
442     pfmod.h          \|
443     pg.h             \|
444     pghw.h           \|
445     physmem.h        \|
446     pkp_hash.h       \|
447     pm.h             \|
448     policy.h         \|
449     poll.h           \|
450     poll_impl.h      \|
451     pool.h           \|
452     pool_impl.h      \|
453     pool_pset.h      \|
454     port.h           \|

```

```

455     port_impl.h      \|
456     port_kernel.h    \|
457     portif.h         \|
458     ppmio.h          \|
459     pppt_ic_if.h     \|
460     pppt_ioctl.h     \|
461     priocntl.h       \|
462     priv.h           \|
463     priv_impl.h      \|
464     prnio.h          \|
465     proc.h           \|
466     processor.h      \|
467     procfs.h         \|
468     procset.h        \|
469     project.h        \|
470     protosw.h        \|
471     prsystem.h       \|
472     pset.h           \|
473     pshot.h          \|
474     ptem.h           \|
475     ptms.h           \|
476     ptyvar.h         \|
477     raidioctl.h      \|
478     ramdisk.h        \|
479     random.h         \|
480     rctl.h           \|
481     rctl_impl.h      \|
482     rds.h            \|
483     reboot.h         \|
484     refstr.h         \|
485     refstr_impl.h    \|
486     resource.h       \|
487     rliocntl.h       \|
488     rt.h             \|
489     rtpriocntl.h     \|
490     rwlock.h         \|
491     rwlock_impl.h    \|
492     rwstlock.h       \|
493     sad.h            \|
494     schedctl.h       \|
495     sdt.h            \|
496     secflags.h       \|
497     #endif /* ! codereview */
498     select.h         \|
499     sem.h            \|
500     sem_impl.h       \|
501     sema_impl.h      \|
502     semaphore.h      \|
503     sendfile.h       \|
504     ser_sync.h        \|
505     session.h        \|
506     sha1.h           \|
507     sha1_consts.h    \|
508     sha2.h           \|
509     sha2_consts.h    \|
510     share.h          \|
511     shm.h            \|
512     shm_impl.h       \|
513     sid.h            \|
514     signinfo.h       \|
515     signal.h         \|
516     signalfd.h       \|
517     skein.h          \|
518     sleepq.h         \|
519     sbios.h          \|
520     sbios_impl.h     \|

```

```

521     subject.h      \
522     socket.h       \
523     socket_impl.h  \
524     socket_proto.h \
525     socketvar.h    \
526     sockfilter.h   \
527     sockio.h       \
528     soundcard.h    \
529     squeue.h       \
530     squeue_impl.h  \
531     srn.h          \
532     sservice.h     \
533     stat.h         \
534     statfs.h       \
535     statvfs.h      \
536     stdbool.h      \
537     stdint.h       \
538     stermio.h      \
539     stmf.h         \
540     stmf_defines.h \
541     stmf_ioctl.h   \
542     stmf_sbd_ioctl.h \
543     stream.h       \
544     strft.h        \
545     strlog.h       \
546     strmddep.h     \
547     stropts.h      \
548     strredir.h     \
549     strstat.h      \
550     strsubr.h      \
551     strsun.h       \
552     strtty.h       \
553     sunddi.h       \
554     sunldi.h       \
555     sunldi_impl.h  \
556     sunmdi.h       \
557     sunndi.h       \
558     sunos_dhcp_class.h \
559     sunpm.h        \
560     suntpl.h       \
561     suntty.h       \
562     swap.h         \
563     synch.h        \
564     sysdc.h        \
565     sysdc_impl.h   \
566     syscall.h      \
567     sysconf.h      \
568     sysconfig.h    \
569     sysevent.h     \
570     sysevent_impl.h \
571     sysinfo.h      \
572     syslog.h       \
573     sysmacros.h    \
574     sysmsg_impl.h  \
575     systeminfo.h   \
576     system.h       \
577     task.h         \
578     taskq.h        \
579     taskq_impl.h   \
580     t_kuser.h      \
581     t_lock.h       \
582     telioctl.h     \
583     termio.h       \
584     termios.h      \
585     termiox.h      \
586     thread.h       \

```

```

587     ticlts.h       \
588     ticots.h       \
589     ticotsord.h    \
590     tihdr.h        \
591     time.h         \
592     time_impl.h    \
593     time_std_impl.h \
594     timeb.h        \
595     timer.h        \
596     timerfd.h      \
597     times.h        \
598     timex.h        \
599     timod.h        \
600     tirdwr.h       \
601     tiuser.h       \
602     tl.h           \
603     tnf.h          \
604     tnf_com.h      \
605     tnf_probe.h    \
606     tnf_writer.h   \
607     todio.h        \
608     tpicommon.h    \
609     ts.h           \
610     tspriocntl.h   \
611     ttcompat.h     \
612     ttold.h        \
613     tty.h          \
614     ttychars.h     \
615     ttydev.h       \
616     tuneable.h     \
617     turnstile.h    \
618     types.h        \
619     types32.h      \
620     tzfile.h       \
621     u8_textprep.h  \
622     u8_textprep_data.h \
623     uadmin.h       \
624     ucred.h        \
625     uio.h          \
626     ulimit.h       \
627     un.h           \
628     unistd.h       \
629     user.h         \
630     ustat.h        \
631     utime.h        \
632     utsname.h      \
633     utssys.h       \
634     uuid.h         \
635     va_impl.h      \
636     va_list.h      \
637     var.h          \
638     varargs.h      \
639     vfs.h          \
640     vfs_opreg.h    \
641     vfstab.h       \
642     vgareg.h       \
643     videodev2.h    \
644     visual_io.h    \
645     vlan.h         \
646     vm.h           \
647     vm_usage.h     \
648     vmem.h         \
649     vmem_impl.h    \
650     vmsystem.h     \
651     vnic.h         \
652     vnic_impl.h    \

```



```

653     vnode.h          \
654     vscan.h          \
655     vtoc.h           \
656     vtrace.h         \
657     vuid_event.h     \
658     vuid_wheel.h     \
659     vuid_queue.h     \
660     vuid_state.h     \
661     vuid_store.h     \
662     wait.h           \
663     waitq.h          \
664     wanboot_impl.h   \
665     watchpoint.h     \
666     winlockio.h     \
667     zcons.h          \
668     zone.h           \
669     xti_inet.h       \
670     xti_osi.h        \
671     xti_xtiopt.h    \
672     zmod.h           \

674 HDRS=                \
675     $(GENHDRS)        \
676     $(CHKHDRS)       \

678 AUDIOHDRS=          \
679     ac97.h            \
680     audio_common.h   \
681     audio_driver.h   \
682     audio_oss.h      \
683     g711.h           \

685 AVHDRS=             \
686     iec61883.h      \

688 BSCHDRS=            \
689     bscbus.h         \
690     bscv_impl.h     \
691     lom_ebuscodes.h \
692     lom_io.h         \
693     lom_priv.h      \
694     lombus.h        \

696 MDESCHDRS=          \
697     mdesc.h          \
698     mdesc_impl.h    \

700 CPUDRVHDRS=         \
701     cpudrv.h        \

703 CRYPTOHDRS=         \
704     elfsign.h       \
705     ioctl.h         \
706     ioctladmin.h   \
707     common.h        \
708     impl.h          \
709     spi.h           \
710     api.h           \
711     ops_impl.h     \
712     sched_impl.h   \

714 DCAMHDRS=           \
715     dcaml394_io.h  \

717 IBHDRS=              \
718     ib_types.h     \

```

```

719     ib_pkt_hdrs.h   \

721 IBTLHDRS=            \
722     ibtl_types.h    \
723     ibtl_status.h   \
724     ibti.h          \
725     ibti_cm.h       \
726     ibci.h          \
727     ibti_common.h   \
728     ibvti.h         \
729     ibtl_ci_types.h \

731 IBTLIMPLHDRS=        \
732     ibtl_util.h     \

734 IBNEXHDRS=          \
735     ibnex_devctl.h  \

737 IBMFHDRS=            \
738     ibmf.h          \
739     ibmf_msg.h      \
740     ibmf_saa.h      \
741     ibmf_utils.h    \

743 IBMGTHDRS=          \
744     ib_dm_attr.h   \
745     ib_mad.h        \
746     sm_attr.h       \
747     sa_recs.h       \

749 IBDHDRS=            \
750     ibd.h           \

752 OFHDRS=              \
753     ofa_solaris.h  \
754     ofed_kernel.h  \

756 RDMAHDRS=           \
757     ib_addr.h      \
758     ib_user_mad.h  \
759     ib_user_sa.h   \
760     ib_user_verbs.h \
761     ib_verbs.h     \
762     rdma_cm.h      \
763     rdma_user_cm.h \

765 SOL_UVERBSHDRS=     \
766     sol_uverbs.h   \
767     sol_uverbs2ucma.h \
768     sol_uverbs_comp.h \
769     sol_uverbs_hca.h \
770     sol_uverbs_qp.h \
771     sol_uverbs_event.h \

773 SOL_UMADHDRS=       \
774     sol_umad.h     \

776 SOL_UCMAHDRS=       \
777     sol_ucma.h     \
778     sol_rdma_user_cm.h \

780 SOL_OFSHDRS=        \
781     sol_cma.h      \
782     sol_ib_cma.h   \
783     sol_ofs_common.h \
784     sol_kverb_impl.h \

```

```

786 TAVORHDRS= \
787     tavor_ioctl.h \

789 HERMONHDRS= \
790     hermon_ioctl.h \

792 MLNXHDRS= \
793     mlnx_umap.h \

795 IDMHDRS= \
796     idm.h \
797     idm_impl.h \
798     idm_so.h \
799     idm_text.h \
800     idm_transport.h \
801     idm_conn_sm.h \

803 ISCSITHDRS= \
804     radius_packet.h \
805     radius_protocol.h \
806     chap.h \
807     isns_protocol.h \
808     iscsi_if.h \
809     iscsit_common.h \

811 ISOHDRS= \
812     signal_iso.h \

814 DERIVED_LVMHDRS= \
815     md_mdiox.h \
816     md_basic.h \
817     mdmed.h \
818     md_mhdx.h \
819     mdmn_commd.h \

821 LVMHDRS= \
822     md_convert.h \
823     md_crc.h \
824     md_hotspares.h \
825     md_mddb.h \
826     md_mirror.h \
827     md_mirror_shared.h \
828     md_names.h \
829     md_notify.h \
830     md_raid.h \
831     md_rename.h \
832     md_sp.h \
833     md_stripe.h \
834     md_trans.h \
835     mdio.h \
836     mdvar.h \

838 ALL_LVMHDRS= \
839     $(LVMHDRS) \
840     $(DERIVED_LVMHDRS) \

842 FMHDRS= \
843     protocol.h \
844     util.h \

846 FMFSHDRS= \
847     zfs.h \

849 FMIOHDRS= \
850     ddi.h \

```

```

851     disk.h \
852     pci.h \
853     scsi.h \
854     sun4upci.h \
855     opl_mc_fm.h \

857 FSHDRS= \
858     autofs.h \
859     decomp.h \
860     dv_node.h \
861     sdev_impl.h \
862     fifonode.h \
863     hsfs_isospec.h \
864     hsfs_node.h \
865     hsfs_rrip.h \
866     hsfs_spec.h \
867     hsfs_susp.h \
868     lofs_info.h \
869     lofs_node.h \
870     mntdata.h \
871     namenode.h \
872     pc_dir.h \
873     pc_fs.h \
874     pc_label.h \
875     pc_node.h \
876     pxfs_ki.h \
877     snode.h \
878     swapnode.h \
879     tmp.h \
880     tmpnode.h \
881     udf_inode.h \
882     udf_volume.h \
883     ufs_acl.h \
884     ufs_bio.h \
885     ufs_filio.h \
886     ufs_fs.h \
887     ufs_fsdir.h \
888     ufs_inode.h \
889     ufs_lockfs.h \
890     ufs_log.h \
891     ufs_mount.h \
892     ufs_panic.h \
893     ufs_prot.h \
894     ufs_quota.h \
895     ufs_snap.h \
896     ufs_trans.h \
897     zfs.h \
898     zut.h \

900 SCSEHDRS= \
901     scsi.h \
902     scsi_address.h \
903     scsi_ctl.h \
904     scsi_fm.h \
905     scsi_names.h \
906     scsi_params.h \
907     scsi_pkt.h \
908     scsi_resource.h \
909     scsi_types.h \
910     scsi_watch.h \

912 SCSECONFHDRS= \
913     autoconf.h \
914     device.h \

916 SCSEGENHDRS= \

```

```

917     commands.h      \
918     dad_mode.h      \
919     inquiry.h       \
920     message.h       \
921     mode.h          \
922     persist.h       \
923     sense.h         \
924     sff_frames.h    \
925     smp_frames.h    \
926     status.h        \

928 SCIIIMPLHDRS=      \
929     commands.h      \
930     inquiry.h       \
931     mode.h          \
932     scsi_reset_notify.h \
933     scsi_sas.h      \
934     sense.h         \
935     services.h      \
936     smp_transport.h \
937     spc3_types.h    \
938     status.h        \
939     transport.h     \
940     types.h         \
941     uscsm.h         \
942     usmp.h          \

944 SCSTITARGETSHDRS= \
945     ses.h           \
946     sesio.h         \
947     sgendef.h       \
948     stdef.h         \
949     sddef.h         \
950     smp.h           \

952 SCSIADHDRS=

954 SCASICADHDRS=

956 SCIIISCSIHDRS=    \
957     iscsi_door.h   \
958     iscsi_if.h     \

960 SCIVHCIHDRS=      \
961     scsi_vhci.h    \
962     mpapi_impl.h   \
963     mpapi_scsi_vhci.h \

965 SDCARDHDRS=       \
966     sda.h          \
967     sda_impl.h     \
968     sda_ioctl.h    \

970 FC4HDRS=          \
971     fc_transport.h \
972     linkapp.h      \
973     fc.h           \
974     fcp.h          \
975     fcal_transport.h \
976     fcal.h         \
977     fcal_linkapp.h \
978     fcio.h         \

980 FCHDRS=           \
981     fc.h           \
982     fcio.h         \

```

```

983     fc_types.h     \
984     fc_appif.h     \

986 FCIMPLHDRS=       \
987     fc_error.h     \
988     fcph.h         \

990 FCULPHDRS=        \
991     fcp_util.h     \
992     fcsm.h         \

994 SATAGENHDRS=     \
995     sata_hba.h     \
996     sata_defs.h    \
997     sata_cfgadm.h  \

999 SYSEVENTHDRS=    \
1000     ap_driver.h    \
1001     dev.h           \
1002     domain.h       \
1003     dr.h            \
1004     env.h           \
1005     eventdefs.h    \
1006     ipmp.h         \
1007     pwrctl.h       \
1008     svm.h           \
1009     vrrp.h         \

1011 CONTRACTHDRS=    \
1012     process.h       \
1013     process_impl.h \
1014     device.h        \
1015     device_impl.h  \

1017 USBHDRS=          \
1018     usba.h          \
1019     usbai.h         \

1021 USBAUDHDRS=       \
1022     usb_audio.h     \

1024 USBHUBDHDRS=     \
1025     hub.h           \
1026     hubd_impl.h    \

1028 USBHIDHDRS=      \
1029     hid.h           \

1031 USBMSHDRS=       \
1032     usb_bulkonly.h \
1033     usb_cbi.h       \

1035 USBPRNHDRS=      \
1036     usb_printer.h  \

1038 USBDCDCHDRS=     \
1039     usb_cdc.h       \

1041 USBVIDHDRS=      \
1042     usbvc.h         \

1044 USBWCMHDRS=      \
1045     usbwcm.h        \

1047 UGENHDRS=        \
1048     usb_ugen.h      \

```

```

1050 HOTPLUGHDRS= \
1051     hpcsvc.h \
1052     hpctrl.h

1054 HOTPLUGPCIHDRS= \
1055     pcicfg.h \
1056     pcihp.h

1058 RSMHDRS= \
1059     rsm.h \
1060     rsm_common.h \
1061     rsmapi_common.h \
1062     rsmapi.h \
1063     rsmapi_driver.h \
1064     rsmka_path_int.h

1066 TSOLHDRS= \
1067     label.h \
1068     label_macro.h \
1069     priv.h \
1070     tndb.h \
1071     tsyscall.h

1073 I1394HDRS= \
1074     cmd1394.h \
1075     id1394.h \
1076     ieee1212.h \
1077     ieee1394.h \
1078     ix1394.h \
1079     s1394_impl.h \
1080     t1394.h

1082 # "cmdk" headers used on sparc
1083 SDKTPHDRS= \
1084     dadkio.h \
1085     fdisk.h

1087 # "cmdk" headers used on i386
1088 DKTPHDRS= \
1089     altsctr.h \
1090     bbh.h \
1091     cm.h \
1092     cmddev.h \
1093     cmdk.h \
1094     cmpkt.h \
1095     controller.h \
1096     dadev.h \
1097     dadk.h \
1098     dadkio.h \
1099     fctypes.h \
1100     fdisk.h \
1101     flowctrl.h \
1102     gda.h \
1103     quetypes.h \
1104     queue.h \
1105     tgcom.h \
1106     tgdk.h

1108 # "pc" header files used on i386
1109 PCHDRS= \
1110     avintr.h \
1111     dma_engine.h \
1112     i8272A.h \
1113     pcic_reg.h \
1114     pcic_var.h

```

```

1115     pic.h \
1116     pit.h \
1117     rtc.h

1119 NXGEHDRS= \
1120     nxge.h \
1121     nxge_common.h \
1122     nxge_common_impl.h \
1123     nxge_defs.h \
1124     nxge_hw.h \
1125     nxge_impl.h \
1126     nxge_ipp.h \
1127     nxge_ipp_hw.h \
1128     nxge_mac.h \
1129     nxge_mac_hw.h \
1130     nxge_fflp.h \
1131     nxge_fflp_hw.h \
1132     nxge_mii.h \
1133     nxge_rxdma.h \
1134     nxge_rxdma_hw.h \
1135     nxge_txc.h \
1136     nxge_txc_hw.h \
1137     nxge_txdma.h \
1138     nxge_txdma_hw.h \
1139     nxge_virtual.h \
1140     nxge_espc.h

1142 include Makefile.syshdrs

1144 dcam/%.check: dcam/%.h
1145     $(DOT_H_CHECK)

1147 CHECKHDRS= \
1148     $( $(MACH)_HDRS:%.h=%_check) \
1149     $(AUDIOHDRS:%.h=audio/%_check) \
1150     $(AVHDRS:%.h=av/%_check) \
1151     $(BSCHDRS:%.h=%_check) \
1152     $(CHKHDRS:%.h=%_check) \
1153     $(CPUDRVHDRS:%.h=%_check) \
1154     $(CRYPTOHDRS:%.h=crypto/%_check) \
1155     $(DCAMHDRS:%.h=dcam/%_check) \
1156     $(FC4HDRS:%.h=fc4/%_check) \
1157     $(FCHDRS:%.h=fibre-channel/%_check) \
1158     $(FCIMPLHDRS:%.h=fibre-channel/impl/%_check) \
1159     $(FCULPHDRS:%.h=fibre-channel/ulp/%_check) \
1160     $(IBHDRS:%.h=ib/%_check) \
1161     $(IBDHDRS:%.h=ib/clients/ibd/%_check) \
1162     $(IBTLHDRS:%.h=ib/ibt1/%_check) \
1163     $(IBTLIMPLHDRS:%.h=ib/ibt1/impl/%_check) \
1164     $(IBNEXHDRS:%.h=ib/ibnex/%_check) \
1165     $(IBMGTHDRS:%.h=ib/mgt/%_check) \
1166     $(IBMFHDRS:%.h=ib/mgt/ibmf/%_check) \
1167     $(OFHDRS:%.h=ib/clients/of/%_check) \
1168     $(RDMAHDRS:%.h=ib/clients/of/rdma/%_check) \
1169     $(SOL_UVERBSHDRS:%.h=ib/clients/of/sol_uverbs/%_check) \
1170     $(SOL_UCMAHDRS:%.h=ib/clients/of/sol_ucma/%_check) \
1171     $(SOL_OFSHDRS:%.h=ib/clients/of/sol_ofs/%_check) \
1172     $(TAVORHDRS:%.h=ib/adapters/tavor/%_check) \
1173     $(HERMONHDRS:%.h=ib/adapters/hermon/%_check) \
1174     $(MLNXHDRS:%.h=ib/adapters/%_check) \
1175     $(IDMHDRS:%.h=idm/%_check) \
1176     $(ISCSIHDRS:%.h=iscsi/%_check) \
1177     $(ISCSITHDRS:%.h=iscsit/%_check) \
1178     $(ISOHDRS:%.h=iso/%_check) \
1179     $(FMHDRS:%.h=fm/%_check) \
1180     $(FMFHDRS:%.h=fm/fs/%_check)

```

```

1181 $(FMIOHDRS:%.h=fm/io/.check) \
1182 $(FSDHDRS:%.h=fs/.check) \
1183 $(LVMHDRS:%.h=lvm/.check) \
1184 $(SCSIHDRS:%.h=scsi/.check) \
1185 $(SCSIADHDRS:%.h=scsi/adapters/.check) \
1186 $(SCSICONFHDRS:%.h=scsi/conf/.check) \
1187 $(SCSIIMPLHDRS:%.h=scsi/impl/.check) \
1188 $(SCSIISCSIHDRS:%.h=scsi/adapters/.check) \
1189 $(SCSISGENHDRS:%.h=scsi/generic/.check) \
1190 $(SCSITARGETSHDRS:%.h=scsi/targets/.check) \
1191 $(SCSIVHCIHDRS:%.h=scsi/adapters/.check) \
1192 $(SATAGENHDRS:%.h=sata/.check) \
1193 $(SDCARDHDRS:%.h=sdcard/.check) \
1194 $(SYSEVENTHDRS:%.h=sysevent/.check) \
1195 $(CONTRACTHDRS:%.h=contract/.check) \
1196 $(USBAUDHDRS:%.h=usb/clients/audio/.check) \
1197 $(USBHUBDHDRS:%.h=usb/hubd/.check) \
1198 $(USBHIDHDRS:%.h=usb/clients/hid/.check) \
1199 $(USBMSHDRS:%.h=usb/clients/mass_storage/.check) \
1200 $(USBPRNHDRS:%.h=usb/clients/printer/.check) \
1201 $(USBDCDCHDRS:%.h=usb/clients/usbcdc/.check) \
1202 $(USBVIDHDRS:%.h=usb/clients/video/usbvc/.check) \
1203 $(USBWCMHDRS:%.h=usb/clients/usbinput/usbwcm/.check) \
1204 $(UGENHDRS:%.h=usb/clients/ugen/.check) \
1205 $(USBHDRS:%.h=usb/.check) \
1206 $(I1394HDRS:%.h=1394/.check) \
1207 $(RSMHDRS:%.h=rsm/.check) \
1208 $(TSOLHDRS:%.h=tso1/.check) \
1209 $(NXGEHDRS:%.h=nxge/.check)

```

```
1212 .KEEP_STATE:
```

```

1214 .PARALLEL: \
1215 $(CHECKHDRS) \
1216 $(ROOTHDRS) \
1217 $(ROOTAUDHDRS) \
1218 $(ROOTAVHDRS) \
1219 $(ROOTCRYPTOHDRS) \
1220 $(ROOTDCAMHDRS) \
1221 $(ROOTISOHDRS) \
1222 $(ROOTIDMHDRS) \
1223 $(ROOTISCSIHDRS) \
1224 $(ROOTISCSITHDRS) \
1225 $(ROOTFC4HDRS) \
1226 $(ROOTFCHDRS) \
1227 $(ROOTFCIMPLHDRS) \
1228 $(ROOTFCULPHDRS) \
1229 $(ROOTFMHDRS) \
1230 $(ROOTFMIOHDRS) \
1231 $(ROOTFMFSDHDRS) \
1232 $(ROOTFSDHDRS) \
1233 $(ROOTIBDHDRS) \
1234 $(ROOTIBHDRS) \
1235 $(ROOTIBTLHDRS) \
1236 $(ROOTIBTLIMPLHDRS) \
1237 $(ROOTIBNEXHDRS) \
1238 $(ROOTIBMGTHDRS) \
1239 $(ROOTIBMFDHDRS) \
1240 $(ROOTOFHDRS) \
1241 $(ROOTRDMAHDRS) \
1242 $(ROOTSOL_OFSDHDRS) \
1243 $(ROOTSOL_UMADHDRS) \
1244 $(ROOTSOL_UVERBSHDRS) \
1245 $(ROOTSOL_UCMAHDRS) \
1246 $(ROOTAVORHDRS) \

```

```

1247 $(ROOTHERMONHDRS) \
1248 $(ROOTMLNXHDRS) \
1249 $(ROOTLVMHDRS) \
1250 $(ROOTSCSIHDRS) \
1251 $(ROOTSCSIADHDRS) \
1252 $(ROOTSCSICONFHDRS) \
1253 $(ROOTSCSIISCSIHDRS) \
1254 $(ROOTSCSISGENHDRS) \
1255 $(ROOTSCSIIMPLHDRS) \
1256 $(ROOTSCSIVHCIHDRS) \
1257 $(ROOTSDCARDHDRS) \
1258 $(ROOTSYSEVENTHDRS) \
1259 $(ROOTCONTRACTHDRS) \
1260 $(ROOTUSBHDRS) \
1261 $(ROOTUWBHDRS) \
1262 $(ROOTUWBAHDRS) \
1263 $(ROOTUSBAUDHDRS) \
1264 $(ROOTUSBHUBDHDRS) \
1265 $(ROOTUSBHIDHDRS) \
1266 $(ROOTUSBHRCCHDRS) \
1267 $(ROOTUSBMSHDRS) \
1268 $(ROOTUSBPRNHDRS) \
1269 $(ROOTUSBDCDCHDRS) \
1270 $(ROOTUSBVIDHDRS) \
1271 $(ROOTUSBWCMHDRS) \
1272 $(ROOTUGENHDRS) \
1273 $(ROOTI1394HDRS) \
1274 $(ROOTHOTPLUGHDRS) \
1275 $(ROOTHOTPLUGPCIHDRS) \
1276 $(ROOTRSMHDRS) \
1277 $(ROOTTSOLHDRS) \
1278 $( $(MACH)_ROOTHDRS)

```

```

1281 install_h: \
1282 $(ROOTDIRS) \
1283 LVMDERIVED_H \
1284 .WAIT \
1285 $(ROOTHDRS) \
1286 $(ROOTAUDHDRS) \
1287 $(ROOTAVHDRS) \
1288 $(ROOTCRYPTOHDRS) \
1289 $(ROOTDCAMHDRS) \
1290 $(ROOTISOHDRS) \
1291 $(ROOTIDMHDRS) \
1292 $(ROOTISCSIHDRS) \
1293 $(ROOTISCSITHDRS) \
1294 $(ROOTFC4HDRS) \
1295 $(ROOTFCHDRS) \
1296 $(ROOTFCIMPLHDRS) \
1297 $(ROOTFCULPHDRS) \
1298 $(ROOTFMHDRS) \
1299 $(ROOTFMFSDHDRS) \
1300 $(ROOTFMIOHDRS) \
1301 $(ROOTFSDHDRS) \
1302 $(ROOTIBDHDRS) \
1303 $(ROOTIBHDRS) \
1304 $(ROOTIBTLHDRS) \
1305 $(ROOTIBTLIMPLHDRS) \
1306 $(ROOTIBNEXHDRS) \
1307 $(ROOTIBMGTHDRS) \
1308 $(ROOTIBMFDHDRS) \
1309 $(ROOTOFHDRS) \
1310 $(ROOTRDMAHDRS) \
1311 $(ROOTSOL_OFSDHDRS) \
1312 $(ROOTSOL_UMADHDRS) \

```

```
1313 $(ROOTSOL_UVERBSHDRS) \
1314 $(ROOTSOL_UCMAHDRS) \
1315 $(ROOTTAVORHDRS) \
1316 $(ROOTTHERMONHDRS) \
1317 $(ROOTMLNXHDRS) \
1318 $(ROOTLVMHDRS) \
1319 $(ROOTSCSIHDRS) \
1320 $(ROOTSCSIADHDRS) \
1321 $(ROOTSCSIIISCSIIHDRS) \
1322 $(ROOTSCSICONFHDRS) \
1323 $(ROOTSCSIGENHDRS) \
1324 $(ROOTSCSIIMPLHDRS) \
1325 $(ROOTSCSIVHCIHDRS) \
1326 $(ROOTSDCARDHDRS) \
1327 $(ROOTSYSEVENTHDRS) \
1328 $(ROOTCONTRACTHDRS) \
1329 $(ROOTUWBHDRS) \
1330 $(ROOTUWBAHDRS) \
1331 $(ROOTUSBHDRS) \
1332 $(ROOTUSBAUDHDRS) \
1333 $(ROOTUSBHUBDHDRS) \
1334 $(ROOTUSBHIDHDRS) \
1335 $(ROOTUSBHRCHDRS) \
1336 $(ROOTUSBMSHDRS) \
1337 $(ROOTUSBPRNHDRS) \
1338 $(ROOTUSBCDCHDRS) \
1339 $(ROOTUSBVIDHDRS) \
1340 $(ROOTUSBWCMHDRS) \
1341 $(ROOTUGENHDRS) \
1342 $(ROOT1394HDRS) \
1343 $(ROOTHOTPLUGHDRS) \
1344 $(ROOTHOTPLUGPCIHDRS) \
1345 $(ROOTRSMHDRS) \
1346 $(ROOTTSOLHDRS) \
1347 $(MACH)_ROOTHDRS)

1349 all_h: $(GENHDRS)

1351 priv_const.h: $(PRIVS_AWK) $(PRIVS_DEF)
1352 $(AWK) -f $(PRIVS_AWK) < $(PRIVS_DEF) -v privhfile=$@

1354 priv_names.h: $(PRIVS_AWK) $(PRIVS_DEF)
1355 $(AWK) -f $(PRIVS_AWK) < $(PRIVS_DEF) -v pubhfile=$@

1357 usb/usbdevs.h: $(USBDEVS_AWK) $(USBDEVS_DATA)
1358 $(AWK) -f $(USBDEVS_AWK) $(USBDEVS_DATA) -H > $@

1360 LVMDERIVED_H:
1361 cd $(SRC)/uts/common/sys/lvm; pwd; $(MAKE) all_h

1363 clean:
1364 $(RM) $(GENHDRS)

1366 clobber: clean
1367 cd $(SRC)/uts/common/sys/lvm; pwd; $(MAKE) clobber

1369 check: $(CHECKHDRS)

1371 FRC:
```

```

*****
26448 Thu Jun 30 22:03:38 2016
new/usr/src/uts/common/sys/elf.h
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
_____unchanged_portion_omitted_____

741 typedef Elf64_Xword   Elf64_Capinfo;
742 typedef Elf64_Word    Elf64_Capchain;

744 /*
745 *   Macros to compose and decompose values for capabilities info.
746 *
747 *   sym = ELF64_C_SYM(info)
748 *   grp = ELF64_C_GROUP(info)
749 *   info = ELF64_C_INFO(sym, grp)
750 */
751 #define ELF64_C_SYM(info)      ((info)>>32)
752 #define ELF64_C_GROUP(info)   ((Elf64_Word)(info))
753 #define ELF64_C_INFO(sym, grp) (((Elf64_Xword)(sym)<<32)+(Elf64_Xword)(grp))

755 #endif /* defined(_LP64) || defined(_LONGLONG_TYPE) */
756 #endif

758 /*
759 * Version numbers for SHT_SUNW_capinfo and SHT_SUNW_capchain.
760 */
761 #define CAPINFO_NONE      0
762 #define CAPINFO_CURRENT  1
763 #define CAPINFO_NUM      2

765 #define CAPCHAIN_NONE     0
766 #define CAPCHAIN_CURRENT 1
767 #define CAPCHAIN_NUM     2

769 /*
770 * A SHT_SUNW_capinfo table mirrors a symbol table. A capabilities symbol has
771 * a SHT_SUNW_capinfo table entry that provides an index into the associated
772 * SHT_SUNW_cap capabilities group, and the symbol index of the associated lead
773 * symbol. A capabilities symbol is a local symbol. A global lead capabilities
774 * symbol is tagged with a group CAPINFO_SUNW_GLOB.
775 */
776 #define CAPINFO_SUNW_GLOB    0xff

778 /*
779 * Capabilities values.
780 */
781 #define CA_SUNW_NULL      0
782 #define CA_SUNW_HW_1     1           /* first hardware capabilities entry */
783 #define CA_SUNW_SF_1     2           /* first software capabilities entry */
784 #define CA_SUNW_HW_2     3           /* second hardware capabilities entry */
785 #define CA_SUNW_PLAT     4           /* platform capability entry */
786 #define CA_SUNW_MACH     5           /* machine capability entry */
787 #define CA_SUNW_ID       6           /* capability identifier */
788 #define CA_SUNW_NUM      7

790 /*
791 * Define software capabilities (CA_SUNW_SF_1 values). Note, hardware
792 * capabilities (CA_SUNW_HW_1 values) are taken directly from sys/auxv_$MACH.h.
793 */
794 #define SF1_SUNW_FPKNWN 0x001        /* frame pointer usage is known */
795 #define SF1_SUNW_FPUSED 0x002        /* frame pointer is in use */
796 #define SF1_SUNW_ADDR32 0x004        /* 32-bit address space requirement */

```

```

797 #define SF1_SUNW_MASK    0x007      /* known software capabilities mask */

799 /*
800 *   Known values for note entry types (e_type == ET_CORE)
801 */
802 #define NT_PRSTATUS      1           /* prstatus_t <sys/old_procfs.h> */
803 #define NT_PRFPREG      2           /* prfpregset_t <sys/old_procfs.h> */
804 #define NT_PRPSINFO     3           /* prpsinfo_t <sys/old_procfs.h> */
805 #define NT_PRXREG       4           /* prxregset_t <sys/procfs.h> */
806 #define NT_PLATFORM     5           /* string from sysinfo(SI_PLATFORM) */
807 #define NT_AUXV         6           /* auxv_t array <sys/auxv.h> */
808 #define NT_GWINDOWS     7           /* gwindows_t SPARC only */
809 #define NT_ASRS         8           /* asrset_t SPARC V9 only */
810 #define NT_LDT          9           /* ssd array <sys/sysi86.h> IA32 only */
811 #define NT_PSTATUS     10          /* pstatus_t <sys/procfs.h> */
812 #define NT_PSINFO      13          /* psinfo_t <sys/procfs.h> */
813 #define NT_PRCRED      14          /* prcred_t <sys/procfs.h> */
814 #define NT_UTSNAME     15          /* struct utsname <sys/utsname.h> */
815 #define NT_LWPSTATUS   16          /* lwpstatus_t <sys/procfs.h> */
816 #define NT_LWPINFO     17          /* lwpinfo_t <sys/procfs.h> */
817 #define NT_PRPRIV      18          /* prpriv_t <sys/procfs.h> */
818 #define NT_PRPRIVINFO  19          /* priv_impl_info_t <sys/priv.h> */
819 #define NT_CONTENT     20          /* core_content_t <sys/corectl.h> */
820 #define NT_ZONENAME    21          /* string from getzonenamebyid(3C) */
821 #define NT_FDINFO      22          /* open fd info */
822 #define NT_SPYMASTER   23          /* psinfo_t for agent LWP spymaster */
823 #define NT_SECFLAGS    24          /* process security-flags */
824 #define NT_NUM         24
823 #define NT_NUM         23

827 #ifndef _KERNEL
828 /*
829 * The following routine checks the processor-specific
830 * fields of an ELF header.
831 */
832 int elfheadcheck(unsigned char, Elf32_Half, Elf32_Word);
833 #endif

835 #ifdef __cplusplus
836 }
_____unchanged_portion_omitted_____

```

```

*****
23385 Thu Jun 30 22:03:39 2016
new/usr/src/uts/common/sys/link.h
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
unchanged portion omitted
69 #endif /* defined(_LP64) || defined(_LONGLONG_TYPE) */
70 #endif /* _ASM */

72 /*
73 * Tag values
74 */
75 #define DT_NULL 0 /* last entry in list */
76 #define DT_NEEDED 1 /* a needed object */
77 #define DT_PLTRELSZ 2 /* size of relocations for the PLT */
78 #define DT_PLTGOT 3 /* addresses used by procedure linkage table */
79 #define DT_HASH 4 /* hash table */
80 #define DT_STRTAB 5 /* string table */
81 #define DT_SYMTAB 6 /* symbol table */
82 #define DT_RELA 7 /* addr of relocation entries */
83 #define DT_RELASZ 8 /* size of relocation table */
84 #define DT_RELAENT 9 /* base size of relocation entry */
85 #define DT_STRSZ 10 /* size of string table */
86 #define DT_SYMENT 11 /* size of symbol table entry */
87 #define DT_INIT 12 /* _init addr */
88 #define DT_FINI 13 /* _fini addr */
89 #define DT_SONAME 14 /* name of this shared object */
90 #define DT_RPATH 15 /* run-time search path */
91 #define DT_SYMBOLIC 16 /* shared object linked -Bsymbolic */
92 #define DT_REL 17 /* addr of relocation entries */
93 #define DT_RELSZ 18 /* size of relocation table */
94 #define DT_RELENT 19 /* base size of relocation entry */
95 #define DT_PLTREL 20 /* relocation type for PLT entry */
96 #define DT_DEBUG 21 /* pointer to r_debug structure */
97 #define DT_TEXTREL 22 /* text relocations remain for this object */
98 #define DT_JMPREL 23 /* pointer to the PLT relocation entries */
99 #define DT_BIND_NOW 24 /* perform all relocations at load of object */
100 #define DT_INIT_ARRAY 25 /* pointer to .init_array */
101 #define DT_FINI_ARRAY 26 /* pointer to .fini_array */
102 #define DT_INIT_ARRAYSZ 27 /* size of .init_array */
103 #define DT_FINI_ARRAYSZ 28 /* size of .fini_array */
104 #define DT_RUNPATH 29 /* run-time search path */
105 #define DT_FLAGS 30 /* state flags - see DF_*/

107 /*
108 * DT_* encoding rules: The value of each dynamic tag determines the
109 * interpretation of the d_un union. This convention provides for simpler
110 * interpretation of dynamic tags by external tools. A tag whose value
111 * is an even number indicates a dynamic section entry that uses d_ptr.
112 * A tag whose value is an odd number indicates a dynamic section entry
113 * that uses d_val, or that uses neither d_ptr nor d_val.
114 *
115 * There are exceptions to the above rule:
116 * - Tags with values that are less than DT_ENCODING.
117 * - Tags with values that fall between DT_LOOS and DT_SUNW_ENCODING
118 * - Tags with values that fall between DT_HIOS and DT_LOPROC
119 *
120 * Third party tools must handle these exception ranges explicitly
121 * on an item by item basis.
122 */
123 #define DT_ENCODING 32 /* positive tag DT_* encoding rules */
124 /* start after this */
125 #define DT_PREINIT_ARRAY 32 /* pointer to .preinit_array */

```

```

126 #define DT_PREINIT_ARRAYSZ 33 /* size of .preinit_array */
128 #define DT_MAXPOSTAGS 34 /* number of positive tags */

130 /*
131 * DT_* encoding rules do not apply between DT_LOOS and DT_SUNW_ENCODING
132 */
133 #define DT_LOOS 0x6000000d /* OS specific range */
134 #define DT_SUNW_AUXILIARY 0x6000000d /* symbol auxiliary name */
135 #define DT_SUNW_RTLDINF 0x6000000e /* ld.so.1 info (private) */
136 #define DT_SUNW_FILTER 0x6000000f /* symbol filter name */
137 #define DT_SUNW_CAP 0x60000010 /* hardware/software */
138 /* capabilities */
139 #define DT_SUNW_SYMTAB 0x60000011 /* symtab with local fcn */
140 /* symbols immediately */
141 /* preceding DT_SYMTAB */
142 #define DT_SUNW_SYMSZ 0x60000012 /* Size of SUNW_SYMTAB table */

144 /*
145 * DT_* encoding rules apply between DT_SUNW_ENCODING and DT_HIOS
146 */
147 #define DT_SUNW_ENCODING 0x60000013 /* DT_* encoding rules resume */
148 /* after this */
149 #define DT_SUNW_SORTENT 0x60000013 /* sizeof [SYM|TLS]SORT entry */
150 #define DT_SUNW_SYMSORT 0x60000014 /* sym indices sorted by addr */
151 #define DT_SUNW_SYMSORTSZ 0x60000015 /* size of SUNW_SYMSORT */
152 #define DT_SUNW_TLSSORT 0x60000016 /* tls sym ndx sort by offset */
153 #define DT_SUNW_TLSSORTSZ 0x60000017 /* size of SUNW_TLSSORT */
154 #define DT_SUNW_CAPINFO 0x60000018 /* capabilities symbols */
155 #define DT_SUNW_STRPAD 0x60000019 /* # of unused bytes at the */
156 /* end of dynstr */
157 #define DT_SUNW_CAPCHAIN 0x6000001a /* capabilities chain info */
158 #define DT_SUNW_LDMACH 0x6000001b /* EM machine code of linker */
159 /* that produced object */
160 #define DT_SUNW_CAPCHAINENT 0x6000001d /* capabilities chain entry */
161 #define DT_SUNW_CAPCHAINSZ 0x6000001f /* capabilities chain size */
162 /* 0x60000021 would be DT_SUNW_PARENT */
163 #define DT_SUNW_ASLR 0x60000023 /* executable ASLR desire */
164 #endif /* ! codereview */

166 /*
167 * DT_* encoding rules do not apply between DT_HIOS and DT_LOPROC
168 */
169 #define DT_HIOS 0x6ffff000

171 /*
172 * The following values have been deprecated and remain here to allow
173 * compatibility with older binaries.
174 */
175 #define DT_DEPRECATED_SPARC_REGISTER 0x7000001

177 /*
178 * DT_* entries which fall between DT_VALRNGHI & DT_VALRNGLO use the
179 * Dyn.d_un.d_val field of the Elf*_Dyn structure.
180 */
181 #define DT_VALRNGLO 0x6ffffd00

183 #define DT_GNU_PRELINKED 0x6ffffdf5 /* prelinking timestamp (unused) */
184 #define DT_GNU_CONFLICTSZ 0x6ffffdf6 /* size of conflict section (unused) */
185 #define DT_GNU_LIBLISTSZ 0x6ffffdf7 /* size of library list (unused) */
186 #define DT_CHECKSUM 0x6ffffdf8 /* elf checksum */
187 #define DT_PLTPADSZ 0x6ffffdf9 /* pltpadding size */
188 #define DT_MOVEENT 0x6ffffdfa /* move table entry size */
189 #define DT_MOVESZ 0x6ffffdfb /* move table size */
190 #define DT_FEATURE_1 0x6ffffdfc /* feature holder (unused) */
191 #define DT_POSFLAG_1 0x6ffffdfd /* flags for DT_* entries, effecting */

```



```

192 /* the following DT_* entry. */
193 /* See DF_PL_* definitions */
194 #define DT_SYMINSZ 0x6ffffdfe /* syminfo table size (in bytes) */
195 #define DT_SYMINENT 0x6ffffdff /* syminfo entry size (in bytes) */
196 #define DT_VALRNGHI 0x6ffffdff

198 /*
199 * DT_* entries which fall between DT_ADDRRNGHI & DT_ADDRRNGLO use the
200 * Dyn.d_un.d_ptr field of the Elf*_Dyn structure.
201 *
202 * If any adjustment is made to the ELF object after it has been
203 * built, these entries will need to be adjusted.
204 */
205 #define DT_ADDRRNGLO 0x6ffffe00

207 #define DT_GNU_HASH 0x6ffffef5 /* GNU-style hash table (unused) */
208 #define DT_TLSDESC_PLT 0x6ffffef6 /* GNU (unused) */
209 #define DT_TLSDESC_GOT 0x6ffffef7 /* GNU (unused) */
210 #define DT_GNU_CONFLICT 0x6ffffef8 /* start of conflict section (unused) */
211 #define DT_GNU_LIBLIST 0x6ffffef9 /* Library list (unused) */

213 #define DT_CONFIG 0x6ffffefa /* configuration information */
214 #define DT_DEPAUDIT 0x6ffffefb /* dependency auditing */
215 #define DT_AUDIT 0x6ffffefc /* object auditing */
216 #define DT_PLTPAD 0x6ffffefd /* pltpadding (sparcv9) */
217 #define DT_MOVETAB 0x6ffffefe /* move table */
218 #define DT_SYMINFO 0x6ffffeff /* syminfo table */
219 #define DT_ADDRRNGHI 0x6ffffeff

221 /*
222 * The following DT_* entries should have been assigned within one of the
223 * DT_* ranges, but existed before such ranges had been established.
224 */
225 #define DT_VERSYM 0x6ffffff0 /* version symbol table - unused by */
226 /* Solaris (see libld/update.c) */

228 #define DT_RELACOUNT 0x6ffffff9 /* number of RELATIVE relocations */
229 #define DT_RELCOUNT 0x6ffffffa /* number of RELATIVE relocations */
230 #define DT_FLAGS_1 0x6ffffffb /* state flags - see DF_1_* defs */
231 #define DT_VERDEF 0x6ffffffc /* version definition table and */
232 #define DT_VERDEFNUM 0x6ffffffd /* associated no. of entries */
233 #define DT_VERNEED 0x6ffffffe /* version needed table and */
234 #define DT_VERNEEDNUM 0x6fffffff /* associated no. of entries */

236 /*
237 * DT_* entries between DT_HIPROC and DT_LOPROC are reserved for processor
238 * specific semantics.
239 *
240 * DT_* encoding rules apply to all tag values larger than DT_LOPROC.
241 */
242 #define DT_LOPROC 0x70000000 /* processor specific range */
243 #define DT_AUXILIARY 0x7fffffff /* shared library auxiliary name */
244 #define DT_USED 0x7fffffff /* ignored - same as needed */
245 #define DT_FILTER 0x7fffffff /* shared library filter name */
246 #define DT_HIPROC 0x7fffffff

249 /*
250 * Values for DT_FLAGS
251 */
252 #define DF_ORIGIN 0x00000001 /* ORIGIN processing required */
253 #define DF_SYMBOLIC 0x00000002 /* symbolic bindings in effect */
254 #define DF_TEXTREL 0x00000004 /* text relocations remain */
255 #define DF_BIND_NOW 0x00000008 /* process all relocations */
256 #define DF_STATIC_TLS 0x00000010 /* obj. contains static TLS refs */

```

```

258 /*
259 * Values for the DT_POSFLAG_1 .dynamic entry.
260 * These values only affect the following DT_* entry.
261 */
262 #define DF_P1_LAZYLOAD 0x00000001 /* following object is to be */
263 /* lazy loaded */
264 #define DF_P1_GROUPELPERM 0x00000002 /* following object's symbols are */
265 /* not available for general */
266 /* symbol bindings. */
267 #define DF_P1_DEFERRED 0x00000004 /* following object is deferred */

269 /*
270 * Values for the DT_FLAGS_1 .dynamic entry.
271 */
272 #define DF_1_NOW 0x00000001 /* set RTLD_NOW for this object */
273 #define DF_1_GLOBAL 0x00000002 /* set RTLD_GLOBAL for this object */
274 #define DF_1_GROUP 0x00000004 /* set RTLD_GROUP for this object */
275 #define DF_1_NODELETE 0x00000008 /* set RTLD_NODELETE for this object */
276 #define DF_1_LOADFLTR 0x00000010 /* trigger filtee loading at runtime */
277 #define DF_1_INITFIRST 0x00000020 /* set RTLD_INITFIRST for this object */
278 #define DF_1_NOOPEN 0x00000040 /* set RTLD_NOOPEN for this object */
279 #define DF_1_ORIGIN 0x00000080 /* ORIGIN processing required */
280 #define DF_1_DIRECT 0x00000100 /* direct binding enabled */
281 #define DF_1_TRANS 0x00000200 /* unused obsolete name */
282 #define DF_1_INTERPOSE 0x00000400 /* object is an interposer */
283 #define DF_1_NODEFLIB 0x00000800 /* ignore default library search path */
284 #define DF_1_NODUMP 0x00001000 /* object can't be dldump(3x)'ed */
285 #define DF_1_CONFALT 0x00002000 /* configuration alternative created */
286 #define DF_1_ENDFILTEE 0x00004000 /* filtee terminates filters search */
287 #define DF_1_DISPRELDNE 0x00008000 /* disp reloc applied at build time */
288 #define DF_1_DISPRLPND 0x00010000 /* disp reloc applied at run-time */
289 #define DF_1_NODIRECT 0x00020000 /* object contains symbols that */
290 /* cannot be directly bound to */
291 #define DF_1_IGNMULDEF 0x00040000 /* internal: krtld ignore muldefs */
292 #define DF_1_NOKSYMS 0x00080000 /* internal: don't export object's */
293 /* symbols via /dev/ksyms */
294 #define DF_1_NOHDR 0x00100000 /* mapfile: 1st segment mapping */
295 #define DF_1_INTERPOSED 0x00200000 /* omits ELF & program headers */
296 #define DF_1_EDITED 0x00200000 /* object has been modified since */
297 /* being built by 'ld' */
298 #define DF_1_NORELOC 0x00400000 /* internal: unrelocated object */
299 #define DF_1_SYMINTPOSE 0x00800000 /* individual symbol interposers */
300 /* exist */
301 #define DF_1_GLOBAUDIT 0x01000000 /* establish global auditing */
302 #define DF_1_SINGLETON 0x02000000 /* singleton symbols exist */

304 /*
305 * Values set to DT_FEATURE_1 tag's d_val (unused obsolete tag)
306 */
307 #define DTF_1_PARINIT 0x00000001 /* partially initialization feature */
308 #define DTF_1_CONFEXP 0x00000002 /* configuration file expected */

311 /*
312 * Version structures. There are three types of version structure:
313 *
314 * o A definition of the versions within the image itself.
315 * Each version definition is assigned a unique index (starting from
316 * VER_NDX_BGNDEF) which is used to cross-reference symbols associated to
317 * the version. Each version can have one or more dependencies on other
318 * version definitions within the image. The version name, and any
319 * dependency names, are specified in the version definition auxiliary
320 * array. Version definition entries require a version symbol index table.
321 *
322 * o A version requirement on a needed dependency. Each needed entry
323 * specifies the shared object dependency (as specified in DT_NEEDED).

```

```

324 *      One or more versions required from this dependency are specified in the
325 *      version needed auxiliary array.
326 *
327 * o A version symbol index table. Each symbol indexes into this array
328 * to determine its version index. Index values of VER_NDX_BGNDEF or
329 * greater indicate the version definition to which a symbol is associated.
330 * (the size of a symbol index entry is recorded in the sh_info field).
331 */
332 #ifndef _ASM
333
334 typedef struct {
335     Elf32_Half    vd_version; /* Version Definition Structure. */
336     Elf32_Half    vd_flags;   /* this structures version revision */
337     Elf32_Half    vd_ndx;     /* version information */
338     Elf32_Half    vd_cnt;     /* version index */
339     Elf32_Word    vd_hash;    /* no. of associated aux entries */
340     Elf32_Word    vd_aux;     /* version name hash value */
341     Elf32_Word    vd_next;    /* no. of bytes from start of this */
342     Elf32_Word    vd_next;    /* verdef to verdaux array */
343 } Elf32_Verdef; /* no. of bytes from start of this */
/* verdef to next verdef entry */
344
345 typedef struct {
346     Elf32_Word    vda_name;   /* Verdef Auxiliary Structure. */
347     Elf32_Word    vda_name;   /* first element defines the version */
348     Elf32_Word    vda_name;   /* name. Additional entries */
349     Elf32_Word    vda_name;   /* define dependency names. */
350     Elf32_Word    vda_name;   /* no. of bytes from start of this */
351     Elf32_Word    vda_name;   /* verdaux to next verdaux entry */
352 } Elf32_Verdaux;
353
354 typedef struct {
355     Elf32_Half    vn_version; /* Version Requirement Structure. */
356     Elf32_Half    vn_cnt;     /* this structures version revision */
357     Elf32_Word    vn_file;    /* no. of associated aux entries */
358     Elf32_Word    vn_aux;     /* name of needed dependency (file) */
359     Elf32_Word    vn_aux;     /* no. of bytes from start of this */
360     Elf32_Word    vn_next;    /* verneed to vernaux array */
361     Elf32_Word    vn_next;    /* no. of bytes from start of this */
362     Elf32_Word    vn_next;    /* verneed to next verneed entry */
363 } Elf32_Verneed;
364
365 typedef struct {
366     Elf32_Word    vna_hash;   /* Verneed Auxiliary Structure. */
367     Elf32_Half    vna_flags;  /* version name hash value */
368     Elf32_Half    vna_flags;  /* version information */
369     Elf32_Half    vna_flags;  /* version name */
370     Elf32_Word    vna_name;   /* version name */
371     Elf32_Word    vna_name;   /* no. of bytes from start of this */
372     Elf32_Word    vna_name;   /* vernaux to next vernaux entry */
373 } Elf32_Vernaux;
374
375 typedef Elf32_Half Elf32_Versym; /* Version symbol index array */
376
377 typedef struct {
378     Elf32_Half    si_boundto; /* Verneed Auxiliary Structure. */
379     Elf32_Half    si_boundto; /* version name hash value */
380     Elf32_Half    si_boundto; /* version information */
381     Elf32_Half    si_boundto; /* version name */
382     Elf32_Half    si_boundto; /* version name */
383     Elf32_Half    si_boundto; /* version name */
384     Elf32_Half    si_boundto; /* version name */
385     Elf32_Half    si_boundto; /* version name */
386     Elf32_Half    si_boundto; /* version name */
387     Elf32_Half    si_boundto; /* version name */
388     Elf32_Half    si_boundto; /* version name */
389 } Elf32_Syminfo;
390
391 #if defined(_LP64) || defined(_LONGLONG_TYPE)
392 typedef struct {
393     Elf64_Half    vd_version; /* this structures version revision */
394     Elf64_Half    vd_flags;   /* version information */
395     Elf64_Half    vd_ndx;     /* version index */
396     Elf64_Half    vd_cnt;     /* no. of associated aux entries */
397     Elf64_Word    vd_hash;    /* version name hash value */
398     Elf64_Word    vd_aux;     /* no. of bytes from start of this */
399     Elf64_Word    vd_next;    /* verdef to verdaux array */
400     Elf64_Word    vd_next;    /* no. of bytes from start of this */
401     Elf64_Word    vd_next;    /* verdef to next verdef entry */
402 } Elf64_Verdef;

```

```

390 typedef struct {
391     Elf64_Word    vda_name;   /* first element defines the version */
392     Elf64_Word    vda_name;   /* name. Additional entries */
393     Elf64_Word    vda_name;   /* define dependency names. */
394     Elf64_Word    vda_name;   /* no. of bytes from start of this */
395     Elf64_Word    vda_name;   /* verdaux to next verdaux entry */
396 } Elf64_Verdaux;
397
398 typedef struct {
399     Elf64_Half    vn_version; /* this structures version revision */
400     Elf64_Half    vn_cnt;     /* no. of associated aux entries */
401     Elf64_Word    vn_file;    /* name of needed dependency (file) */
402     Elf64_Word    vn_aux;     /* no. of bytes from start of this */
403     Elf64_Word    vn_next;    /* verneed to vernaux array */
404     Elf64_Word    vn_next;    /* no. of bytes from start of this */
405     Elf64_Word    vn_next;    /* verneed to next verneed entry */
406 } Elf64_Verneed;
407
408 typedef struct {
409     Elf64_Word    vna_hash;   /* Verneed Auxiliary Structure. */
410     Elf64_Half    vna_flags;  /* version name hash value */
411     Elf64_Half    vna_flags;  /* version information */
412     Elf64_Half    vna_flags;  /* version name */
413     Elf64_Word    vna_name;   /* version name */
414     Elf64_Word    vna_name;   /* no. of bytes from start of this */
415     Elf64_Word    vna_name;   /* vernaux to next vernaux entry */
416 } Elf64_Vernaux;
417
418 typedef Elf64_Half Elf64_Versym;
419
420 typedef struct {
421     Elf64_Half    si_boundto; /* direct bindings - symbol bound to */
422     Elf64_Half    si_boundto; /* per symbol flags */
423 } Elf64_Syminfo;
424 #endif /* defined(_LP64) || defined(_LONGLONG_TYPE) */
425
426 #endif /* _ASM */
427
428 /*
429 * Versym symbol index values. Values greater than VER_NDX_GLOBAL
430 * and less than VER_NDX_LORESERVE associate symbols with user
431 * specified version descriptors.
432 */
433 #define VER_NDX_LOCAL 0 /* symbol is local */
434 #define VER_NDX_GLOBAL 1 /* symbol is global and assigned to */
435 /* the base version */
436 #define VER_NDX_LORESERVE 0xff00 /* beginning of RESERVED entries */
437 #define VER_NDX_ELIMINATE 0xff01 /* symbol is to be eliminated */
438
439 /*
440 * Verdef (vd_flags) and Vernaux (vna_flags) flags values.
441 */
442 #define VER_FLG_BASE 0x1 /* version definition of file itself */
443 /* (Verdef only) */
444 #define VER_FLG_WEAK 0x2 /* weak version identifier */
445 #define VER_FLG_INFO 0x4 /* version is recorded in object for */
446 /* informational purposes */
447 /* (Versym reference) only. No */
448 /* runtime verification is */
449 /* required. (Vernaux only) */
450
451 /*
452 * Verdef version values.
453 */
454 #define VER_DEF_NONE 0 /* Ver_def version */
455 #define VER_DEF_CURRENT 1
456 #define VER_DEF_NUM 2
457
458 /*
459 * Verneed version values.

```

```

456 */
457 #define VER_NEED_NONE      0      /* Ver_need version */
458 #define VER_NEED_CURRENT  1
459 #define VER_NEED_NUM      2

462 /*
463 * Syminfo flag values
464 */
465 #define SYMINFO_FLG_DIRECT  0x0001 /* symbol ref has direct association */
466                               /* to object containing defn. */
467 #define SYMINFO_FLG_FILTER  0x0002 /* symbol ref is associated to a */
468                               /* standard filter */
469 #define SYMINFO_FLG_PASSTHRU SYMINFO_FLG_FILTER /* unused obsolete name */
470 #define SYMINFO_FLG_COPY    0x0004 /* symbol is a copy-reloc */
471 #define SYMINFO_FLG_LAZYLOAD 0x0008 /* object containing defn. should be */
472                               /* lazily-loaded */
473 #define SYMINFO_FLG_DIRECTBIND 0x0010 /* ref should be bound directly to */
474                               /* object containing defn. */
475 #define SYMINFO_FLG_NOEXTDIRECT 0x0020 /* don't let an external reference */
476                               /* directly bind to this symbol */
477 #define SYMINFO_FLG_AUXILIARY 0x0040 /* symbol ref is associated to a */
478                               /* auxiliary filter */
479 #define SYMINFO_FLG_INTERPOSE 0x0080 /* symbol defines an interposer */
480 #define SYMINFO_FLG_CAP      0x0100 /* symbol is capabilities specific */
481 #define SYMINFO_FLG_DEFERRED 0x0200 /* symbol should not be included in */
482                               /* BIND_NOW relocations */

484 /*
485 * Syminfo.si_boundto values.
486 */
487 #define SYMINFO_BT_SELF     0xffff /* symbol bound to self */
488 #define SYMINFO_BT_PARENT  0xfffe /* symbol bound to parent */
489 #define SYMINFO_BT_NONE    0xffffd /* no special symbol binding */
490 #define SYMINFO_BT_EXTERN  0xfffc /* symbol defined as external */
491 #define SYMINFO_BT_LOWRESERVE 0xff00 /* beginning of reserved entries */

493 /*
494 * Syminfo version values.
495 */
496 #define SYMINFO_NONE      0      /* Syminfo version */
497 #define SYMINFO_CURRENT  1
498 #define SYMINFO_NUM      2

501 /*
502 * Public structure defined and maintained within the runtime linker
503 */
504 #ifndef _ASM

506 typedef struct link_map Link_map;

508 struct link_map {
509     unsigned long l_addr; /* address at which object is mapped */
510     char *l_name; /* full name of loaded object */
511     #ifdef LP64
512     Elf64_Dyn *l_ld; /* dynamic structure of object */
513     #else
514     Elf32_Dyn *l_ld; /* dynamic structure of object */
515     #endif
516     Link_map *l_next; /* next link object */
517     Link_map *l_prev; /* previous link object */
518     char *l_refname; /* filters reference name */
519 };

521 #ifdef _SYSCALL32

```

```

522 typedef struct link_map32 Link_map32;

524 struct link_map32 {
525     Elf32_Word l_addr;
526     Elf32_Addr l_name;
527     Elf32_Addr l_ld;
528     Elf32_Addr l_next;
529     Elf32_Addr l_prev;
530     Elf32_Addr l_refname;
531 };
532 #endif

534 typedef enum {
535     RT_CONSISTENT,
536     RT_ADD,
537     RT_DELETE
538 } r_state_e;

540 typedef enum {
541     RD_FL_NONE = 0, /* no flags */
542     RD_FL_ODBG = (1<<0), /* old style debugger present */
543     RD_FL_DBG = (1<<1) /* debugging enabled */
544 } rd_flags_e;

548 /*
549 * Debugging events enabled inside of the runtime linker. To
550 * access these events see the librtld_db interface.
551 */
552 typedef enum {
553     RD_NONE = 0, /* no event */
554     RD_PREINIT, /* the Initial rendezvous before .init */
555     RD_POSTINIT, /* the Second rendezvous after .init */
556     RD_DLACTIVITY /* a dlopen or dlclose has happened */
557 } rd_event_e;

559 struct r_debug {
560     int r_version; /* debugging info version no. */
561     Link_map *r_map; /* address of link_map */
562     unsigned long r_brk; /* address of update routine */
563     r_state_e r_state;
564     unsigned long r_ldbase; /* base addr of ld.so */
565     Link_map *r_ldsomap; /* address of ld.so.1's link map */
566     rd_event_e r_rdevent; /* debug event */
567     rd_flags_e r_flags; /* misc flags. */
568 };

570 #ifdef _SYSCALL32
571 struct r_debug32 {
572     Elf32_Word r_version; /* debugging info version no. */
573     Elf32_Addr r_map; /* address of link_map */
574     Elf32_Word r_brk; /* address of update routine */
575     r_state_e r_state;
576     Elf32_Word r_ldbase; /* base addr of ld.so */
577     Elf32_Addr r_ldsomap; /* address of ld.so.1's link map */
578     rd_event_e r_rdevent; /* debug event */
579     rd_flags_e r_flags; /* misc flags. */
580 };
581 #endif

584 #define R_DEBUG_VERSION 2 /* current r_debug version */
585 #endif /* _ASM */

587 /*

```

```
588 * Attribute/value structures used to bootstrap ELF-based dynamic linker.
589 */
590 #ifndef _ASM
591 typedef struct {
592     Elf32_Sword eb_tag;           /* what this one is */
593     union {                       /* possible values */
594         Elf32_Word eb_val;
595         Elf32_Addr eb_ptr;
596         Elf32_Off eb_off;
597     } eb_un;
598 } Elf32_Boot;

600 #if defined(_LP64) || defined(_LONGLONG_TYPE)
601 typedef struct {
602     Elf64_Xword eb_tag;           /* what this one is */
603     union {                       /* possible values */
604         Elf64_Xword eb_val;
605         Elf64_Addr eb_ptr;
606         Elf64_Off eb_off;
607     } eb_un;
608 } Elf64_Boot;
609 #endif /* defined(_LP64) || defined(_LONGLONG_TYPE) */
610 #endif /* _ASM */

612 /*
613  * Attributes
614  */
615 #define EB_NULL 0 /* (void) last entry */
616 #define EB_DYNAMIC 1 /* (*) dynamic structure of subject */
617 #define EB_LDSO_BASE 2 /* (caddr_t) base address of ld.so */
618 #define EB_ARGV 3 /* (caddr_t) argument vector */
619 #define EB_ENVV 4 /* (char **) environment strings */
620 #define EB_AUXV 5 /* (auxv_t *) auxiliary vector */
621 #define EB_DEVZERO 6 /* (int) fd for /dev/zero */
622 #define EB_PAGESIZE 7 /* (int) page size */
623 #define EB_MAX 8 /* number of "EBs" */
624 #define EB_MAX_SIZE32 64 /* size in bytes, _ILP32 */
625 #define EB_MAX_SIZE64 128 /* size in bytes, _LP64 */

628 #ifndef _ASM
630 /*
631  * Concurrency communication structure for libc callbacks.
632  */
633 extern void _ld_libc(void *);

635 #pragma unknown_control_flow(_ld_libc)
636 #endif /* _ASM */

638 #ifdef __cplusplus
639 }
640 #endif

642 #endif /* _SYS_LINK_H */
```

new/usr/src/uts/common/sys/mman.h

1

```
*****
15218 Thu Jun 30 22:03:40 2016
new/usr/src/uts/common/sys/mman.h
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */

22 /* Copyright 2013 OmniTI Computer Consulting, Inc. All rights reserved. */
23 /*
24  * Copyright 2014 Garrett D'Amore <garrett@damore.org>
25  *
26  * Copyright 2008 Sun Microsystems, Inc. All rights reserved.
27  * Use is subject to license terms.
28  * Copyright 2015 Joyent, Inc. All rights reserved.
29  */

31 /*      Copyright (c) 1983, 1984, 1985, 1986, 1987, 1988, 1989 AT&T      */
32 /*      All Rights Reserved      */

34 /*
35  * University Copyright- Copyright (c) 1982, 1986, 1988
36  * The Regents of the University of California
37  * All Rights Reserved
38  *
39  * University Acknowledgment- Portions of this document are derived from
40  * software developed by the University of California, Berkeley, and its
41  * contributors.
42  */

44 #ifndef _SYS_MMAN_H
45 #define _SYS_MMAN_H

47 #include <sys/feature_tests.h>

49 #ifdef __cplusplus
50 extern "C" {
51 #endif

53 #if !defined(_ASM) && !defined(_KERNEL)
54 #include <sys/types.h>
55 #endif /* !_ASM && !_KERNEL */

57 /*
58  * Protections are chosen from these bits, or-ed together.
```

new/usr/src/uts/common/sys/mman.h

2

```
59 * Note - not all implementations literally provide all possible
60 * combinations. PROT_WRITE is often implemented as (PROT_READ |
61 * PROT_WRITE) and (PROT_EXECUTE as PROT_READ | PROT_EXECUTE).
62 * However, no implementation will permit a write to succeed
63 * where PROT_WRITE has not been set. Also, no implementation will
64 * allow any access to succeed where prot is specified as PROT_NONE.
65 */
66 #define PROT_READ      0x1          /* pages can be read */
67 #define PROT_WRITE     0x2          /* pages can be written */
68 #define PROT_EXEC      0x4          /* pages can be executed */

70 #ifdef _KERNEL
71 #define PROT_USER      0x8          /* pages are user accessible */
72 #define PROT_ZFOD      (PROT_READ | PROT_WRITE | PROT_EXEC | PROT_USER)
73 #define PROT_ALL       (PROT_READ | PROT_WRITE | PROT_EXEC | PROT_USER)
74 #endif /* _KERNEL */

76 #define PROT_NONE      0x0          /* pages cannot be accessed */

78 /* sharing types: must choose either SHARED or PRIVATE */
79 #define MAP_SHARED     1          /* share changes */
80 #define MAP_PRIVATE    2          /* changes are private */
81 #define MAP_TYPE       0xf        /* mask for share type */

83 /* other flags to mmap (or-ed in to MAP_SHARED or MAP_PRIVATE) */
84 #define MAP_FIXED      0x10       /* user assigns address */
85 /* Not implemented */
86 #define MAP_RENAME     0x20       /* rename private pages to file */
87 #endif /* !codereview */
88 #define MAP_NORESERVE  0x40       /* don't reserve needed swap area */
89 /* Note that 0x80 is _MAP_LOW32, defined below */
90 #endif /* !codereview */
91 #define MAP_ANON       0x100      /* map anonymous pages directly */
92 #define MAP_ANONYMOUS  MAP_ANON   /* (source compatibility) */
93 #define MAP_ALIGN      0x200      /* addr specifies alignment */
94 #define MAP_TEXT       0x400      /* map code segment */
95 #define MAP_INITDATA   0x800      /* map data segment */

97 #ifdef _KERNEL
98 #define _MAP_TEXTREPL  0x1000
99 #define _MAP_RANDOMIZE 0x2000
100 #endif /* !codereview */
101 #endif /* _KERNEL */

85 /* these flags not yet implemented */
86 #define MAP_RENAME     0x20       /* rename private pages to file */

103 #if (_POSIX_C_SOURCE <= 2) && !defined(_XPG4_2)
104 /* these flags are used by memcntl */
105 #define PROC_TEXT      (PROT_EXEC | PROT_READ)
106 #define PROC_DATA      (PROT_READ | PROT_WRITE | PROT_EXEC)
107 #define SHARED         0x10
108 #define PRIVATE        0x20
109 #define VALID_ATTR     (PROT_READ|PROT_WRITE|PROT_EXEC|SHARED|PRIVATE)
110 #endif /* (_POSIX_C_SOURCE <= 2) && !defined(_XPG4_2) */

112 #if (_POSIX_C_SOURCE <= 2) || defined(_XPG4_2)
113 #ifdef _KERNEL
114 #define PROT_EXCL      0x20
115 #endif /* _KERNEL */

117 #define _MAP_LOW32     0x80       /* force mapping in lower 4G of address space */
118 #define MAP_32BIT      _MAP_LOW32

120 /*
121  * For the sake of backward object compatibility, we use the _MAP_NEW flag.
```

```

122 * This flag will be automatically or'ed in by the C library for all
123 * new mmap calls. Previous binaries with old mmap calls will continue
124 * to get 0 or -1 for return values. New mmap calls will get the mapped
125 * address as the return value if successful and -1 on errors. By default,
126 * new mmap calls automatically have the kernel assign the map address
127 * unless the MAP_FIXED flag is given.
128 */
129 #define _MAP_NEW      0x80000000    /* users should not need to use this */
130 #endif /* (_POSIX_C_SOURCE <= 2) */

133 #if !defined(__XOPEN_OR_POSIX) || defined(__EXTENSIONS__)
134 /* External flags for mmapobj syscall (Exclusive of MAP_* flags above) */
135 #define MMOBJ_PADDING      0x10000
136 #define MMOBJ_INTERPRET    0x20000

138 #define MMOBJ_ALL_FLAGS    (MMOBJ_PADDING | MMOBJ_INTERPRET)

140 /*
141 * Values for mr_flags field of mmapobj_result_t below.
142 * The bottom 16 bits are mutually exclusive and thus only one
143 * of them can be set at a time. Use MR_GET_TYPE below to check this value.
144 * The top 16 bits are used for flags which are not mutually exclusive and
145 * thus more than one of these flags can be set for a given mmapobj_result_t.
146 *
147 * MR_PADDING being set indicates that this memory range represents the user
148 * requested padding.
149 *
150 * MR_HDR_ELF being set indicates that the ELF header of the mapped object
151 * is mapped at mr_addr + mr_offset.
152 *
153 * MR_HDR_AOUT being set indicates that the AOUT (4.x) header of the mapped
154 * object is mapped at mr_addr + mr_offset.
155 */

157 /*
158 * External flags for mr_flags field below.
159 */
160 #define MR_PADDING      0x1
161 #define MR_HDR_ELF      0x2
162 #define MR_HDR_AOUT     0x3

164 /*
165 * Internal flags for mr_flags field below.
166 */
167 #ifdef _KERNEL
168 #define MR_RESV 0x80000000    /* overmapped /dev/null */
169 #endif /* _KERNEL */

171 #define MR_TYPE_MASK 0x0000ffff
172 #define MR_GET_TYPE(val)    ((val) & MR_TYPE_MASK)

174 #if !defined(_ASM)
175 typedef struct mmapobj_result {
176     caddr_t      mr_addr;        /* mapping address */
177     size_t       mr_msize;      /* mapping size */
178     size_t       mr_fsize;      /* file size */
179     size_t       mr_offset;     /* offset into file */
180     uint_t       mr_prot;       /* the protections provided */
181     uint_t       mr_flags;      /* info on the mapping */
182 } mmapobj_result_t;
_____unchanged_portion_omitted_____

```

```

*****
9133 Thu Jun 30 22:03:41 2016
new/usr/src/uts/common/sys/policy.h
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
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 (c) 2003, 2010, Oracle and/or its affiliates. All rights reserved.
23 * Copyright 2015, Joyent, Inc. All rights reserved.
24 */

26 #ifndef _SYS_POLICY_H
27 #define _SYS_POLICY_H

29 #include <sys/types.h>
30 #include <sys/cred.h>
31 #include <sys/vnode.h>
32 #include <sys/fs/snode.h>

34 #ifdef __cplusplus
35 extern "C" {
36 #endif

38 #ifdef _KERNEL

40 #ifndef _IN_PORT_T
41 #define _IN_PORT_T
42 typedef uint16_t in_port_t;
43 #endif

45 /*
46  * Policy routines; in case we check privileges in-line.
47  *
48  * priv_policy
49  *     privilege debugging
50  *     audits success & failure
51  *     returns 0 on success, error on failure
52  *
53  * priv_policy_choice
54  *     determines extend of operation
55  *     audit on success
56  *     returns a boolean_t indicating success (B_TRUE) or failure.
57  *
58  * priv_policy_only

```

```

59  *     when auditing is in appropriate (interrupt context)
60  *     to determine context of operation
61  *     returns a boolean_t indicating success (B_TRUE) or failure.
62  *
63  */
64 int priv_policy(const cred_t *, int, boolean_t, int, const char *);
65 boolean_t priv_policy_only(const cred_t *, int, boolean_t);
66 boolean_t priv_policy_choice(const cred_t *, int, boolean_t);

68 struct kipc_perm;
69 struct vfs;
70 struct proc;
71 struct priv_set;

73 int secpolicy_acct(const cred_t *);
74 int secpolicy_require_privs(const cred_t *, const struct priv_set *);
75 int secpolicy_allow_setid(const cred_t *, uid_t, boolean_t);
76 int secpolicy_audit_config(const cred_t *);
77 int secpolicy_audit_getattr(const cred_t *, boolean_t);
78 int secpolicy_audit_modify(const cred_t *);
79 int secpolicy_blacklist(const cred_t *);
80 int secpolicy_chroot(const cred_t *);
81 int secpolicy_clock_highres(const cred_t *);
82 int secpolicy_console(const cred_t *);
83 int secpolicy_contract_identity(const cred_t *);
84 int secpolicy_contract_observer(const cred_t *, struct contract *);
85 boolean_t secpolicy_contract_observer_choice(const cred_t *);
86 int secpolicy_contract_event(const cred_t *);
87 boolean_t secpolicy_contract_event_choice(const cred_t *);
88 int secpolicy_coreadm(const cred_t *);
89 int secpolicy_cpc_cpu(const cred_t *);
90 int secpolicy_dispadm(const cred_t *);
91 int secpolicy_error_inject(const cred_t *);
92 int secpolicy_excl_open(const cred_t *);
93 int secpolicy_fs_allowed_mount(const cred_t *);
94 int secpolicy_fs_config(const cred_t *, const struct vfs *);
95 int secpolicy_fs_linkdir(const cred_t *, const struct vfs *);
96 int secpolicy_fs_minfree(const cred_t *, const struct vfs *);
97 int secpolicy_fs_mount(const cred_t *, vnode_t *, struct vfs *);
98 int secpolicy_fs_quota(const cred_t *, const struct vfs *);
99 int secpolicy_fs_unmount(const cred_t *, struct vfs *);
100 int secpolicy_idmap(const cred_t *);
101 int secpolicy_ip(const cred_t *, int, boolean_t);
102 int secpolicy_ip_config(const cred_t *, boolean_t);
103 int secpolicy_dl_config(const cred_t *);
104 int secpolicy iptun_config(const cred_t *);
105 int secpolicy_ipc_access(const cred_t *, const struct kipc_perm *, mode_t);
106 int secpolicy_ipc_config(const cred_t *);
107 int secpolicy_ipc_owner(const cred_t *, const struct kipc_perm *);
108 int secpolicy_kmdb(const cred_t *);
109 int secpolicy_lock_memory(const cred_t *);
110 int secpolicy_meminfo(const cred_t *);
111 int secpolicy_modctl(const cred_t *, int);
112 int secpolicy_net(const cred_t *, int, boolean_t);
113 int secpolicy_net_bindmlp(const cred_t *);
114 int secpolicy_net_config(const cred_t *, boolean_t);
115 int secpolicy_net_icmpaccess(const cred_t *);
116 int secpolicy_net_mac_aware(const cred_t *);
117 int secpolicy_net_mac_implicit(const cred_t *);
118 int secpolicy_net_observability(const cred_t *);
119 int secpolicy_net_privaddr(const cred_t *, in_port_t, int proto);
120 int secpolicy_net_rawaccess(const cred_t *);
121 boolean_t secpolicy_net_reply_equal(const cred_t *);
122 int secpolicy_newproc(const cred_t *);
123 int secpolicy_nfs(const cred_t *);
124 int secpolicy_pbind(const cred_t *);

```

```

125 int secpolicy_pcfs_modify_bootpartition(const cred_t *);
126 int secpolicy_pfexec_register(const cred_t *);
127 int secpolicy_ponline(const cred_t *);
128 int secpolicy_pool(const cred_t *);
129 int secpolicy_power_mgmt(const cred_t *);
130 int secpolicy_ppp_config(const cred_t *);
131 int secpolicy_proc_access(const cred_t *);
132 int secpolicy_proc_excl_open(const cred_t *);
133 int secpolicy_proc_owner(const cred_t *, const cred_t *, int);
134 int secpolicy_proc_zone(const cred_t *);
135 int secpolicy_psecflags(const cred_t *, struct proc *, struct proc *);
136 #endif /* ! codereview */
137 int secpolicy_pset(const cred_t *);
138 int secpolicy_rctlsys(const cred_t *, boolean_t);
139 int secpolicy_resource(const cred_t *);
140 int secpolicy_resource_anon_mem(const cred_t *);
141 int secpolicy_rpcmod_open(const cred_t *);
142 int secpolicy_rsm_access(const cred_t *, uid_t, mode_t);
143 int secpolicy_raisepriority(const cred_t *);
144 int secpolicy_setpriority(const cred_t *);
145 int secpolicy_settime(const cred_t *);
146 int secpolicy_smb(const cred_t *);
147 int secpolicy_smbfs_login(const cred_t *, uid_t);
148 int secpolicy_spec_open(const cred_t *, struct vnode *, int);
149 int secpolicy_sti(const cred_t *);
150 int secpolicy_swapctl(const cred_t *);
151 int secpolicy_sys_config(const cred_t *, boolean_t);
152 int secpolicy_zone_admin(const cred_t *, boolean_t);
153 int secpolicy_zone_config(const cred_t *);
154 int secpolicy_sys_devices(const cred_t *);
155 int secpolicy_systeminfo(const cred_t *);
156 int secpolicy_tasksys(const cred_t *);
157 int secpolicy_vnode_access(const cred_t *, vnode_t *, uid_t, mode_t);
158 int secpolicy_vnode_access2(const cred_t *, vnode_t *, uid_t, mode_t, mode_t);
159 int secpolicy_vnode_any_access(const cred_t *, vnode_t *, uid_t);
160 int secpolicy_vnode_chown(const cred_t *, uid_t);
161 int secpolicy_vnode_create_gid(const cred_t *);
162 int secpolicy_vnode_owner(const cred_t *, uid_t);
163 int secpolicy_vnode_remove(const cred_t *);
164 int secpolicy_vnode_setdac(const cred_t *, uid_t);
165 int secpolicy_vnode_setid_retain(const cred_t *, boolean_t);
166 int secpolicy_vnode_setids_setgids(const cred_t *, gid_t);
167 int secpolicy_vnode_stky_modify(const cred_t *);
168 int secpolicy_vscan(const cred_t *);
169 int secpolicy_zinject(const cred_t *);
170 int secpolicy_zfs(const cred_t *);
171 int secpolicy_ucose_update(const cred_t *);
172 int secpolicy_sadopen(const cred_t *);
173 void secpolicy_setid_clear(vattr_t *, cred_t *);
174 void secpolicy_fs_mount_clearopts(cred_t *, struct vfs *);
175 int secpolicy_setid_setsticky_clear(vnode_t *, vattr_t *,
176     const vattr_t *, cred_t *);
177 int secpolicy_xvattr(xvattr_t *, uid_t, cred_t *, vtype_t);
178 int secpolicy_xvm_control(const cred_t *);

180 int secpolicy_basic_exec(const cred_t *, vnode_t *);
181 int secpolicy_basic_fork(const cred_t *);
182 int secpolicy_basic_link(const cred_t *);
183 int secpolicy_basic_file_read(const cred_t *, vnode_t *, const char *);
184 int secpolicy_basic_file_write(const cred_t *, vnode_t *, const char *);
185 int secpolicy_basic_net_access(const cred_t *);
186 int secpolicy_basic_proc(const cred_t *);
187 int secpolicy_basic_procinfo(const cred_t *, struct proc *, struct proc *);

189 int secpolicy_gart_access(const cred_t *);
190 int secpolicy_gart_map(const cred_t *);

```

```

191 /*
192  * This function to be called from xxfs_setattr().
193  * Must be called with the node's attributes read-write locked.
194  */
195 *         cred_t *         - acting credentials
196 *         struct vnode *   - vnode we're operating on
197 *         struct vattr *va  - new attributes, va_mask may be
198 *                             changed on return from a call
199 *         struct vattr *oldva - old attributes, need include owner
200 *                             and mode only
201 *         int flags        - setattr flags
202 *         int iaccess(void *node, int mode, cred_t *cr)
203 *                             - non-locking internal access function
204 *                             mode be checked
205 *                             w/ VREAD|VWRITE|VEXEC, not fs
206 *                             internal mode encoding.
207 *
208 *         void *node       - internal node (inode, tmpnode) to
209 *                             pass as arg to iaccess
210 */
211 int secpolicy_vnode_setattr(cred_t *, struct vnode *, struct vattr *,
212     const struct vattr *, int, int (void *, int, cred_t *), void *);

214 /*
215  * Test privilege. Audit success or failure, allow privilege debugging.
216  * Returns 0 for success, err for failure.
217 */
218 #define PRIV_POLICY(cred, priv, all, err, reason) \
219     priv_policy((cred), (priv), (all), (err), (reason))

221 /*
222  * Test privilege. Audit success only, no privilege debugging.
223  * Returns 1 for success, and 0 for failure.
224 */
225 #define PRIV_POLICY_CHOICE(cred, priv, all) \
226     priv_policy_choice((cred), (priv), (all))

228 /*
229  * Test privilege. No priv_debugging, no auditing.
230  * Returns 1 for success, and 0 for failure.
231 */

233 #define PRIV_POLICY_ONLY(cred, priv, all) \
234     priv_policy_only((cred), (priv), (all))

237 #endif

239 #ifdef __cplusplus
240 }
241 #endif

243 #endif /* _SYS_POLICY_H */

```


new/usr/src/uts/common/sys/priv_impl.h

1

```
*****
3628 Thu Jun 30 22:03:42 2016
new/usr/src/uts/common/sys/priv_impl.h
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
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, Version 1.0 only
6  * (the "License"). You may not use this file except in compliance
7  * with the License.
8  *
9  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
10 * or http://www.opensolaris.org/os/licensing.
11 * See the License for the specific language governing permissions
12 * and limitations under the License.
13 *
14 * When distributing Covered Code, include this CDDL HEADER in each
15 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
16 * If applicable, add the following below this CDDL HEADER, with the
17 * fields enclosed by brackets "[]" replaced with your own identifying
18 * information: Portions Copyright [yyyy] [name of copyright owner]
19 *
20 * CDDL HEADER END
21 */
22 /*
23  * Copyright 2003 Sun Microsystems, Inc. All rights reserved.
24  * Use is subject to license terms.
25 */

27 #ifndef _SYS_PRIV_IMPL_H
28 #define _SYS_PRIV_IMPL_H

30 #pragma ident      "%Z%M% %I%      %E% SMI"

30 #include <sys/priv_const.h>
31 #include <sys/priv.h>

33 #ifdef __cplusplus
34 extern "C" {
35 #endif

37 #if defined(_KERNEL) || defined(_KMEMUSER)
38 /*
39  * priv_set_t is a structure holding a set of privileges
40  */

42 struct priv_set {
43     priv_chunk_t pbits[PRIV_SETSIZE];
44 };
45     unchanged_portion_omitted

51 #endif

53 #ifdef _KERNEL

55 extern priv_set_t *priv_basic;
56 extern priv_set_t priv_unsafe;
57 extern priv_set_t priv_fullset;
58 extern void priv_init(void);

60 /* The CR_PRIVS macro is defined in <sys/cred_impl.h> */
```

new/usr/src/uts/common/sys/priv_impl.h

2

```
61 #define CR_EPRIV(c)      (CR_PRIVS(c)->crprivs[PRIV_EFFECTIVE])
62 #define CR_IPRIV(c)      (CR_PRIVS(c)->crprivs[PRIV_INHERITABLE])
63 #define CR_PPRIV(c)      (CR_PRIVS(c)->crprivs[PRIV_PERMITTED])
64 #define CR_LPRIV(c)      (CR_PRIVS(c)->crprivs[PRIV_LIMIT])

66 #define CR_FLAGS(c)      (CR_PRIVS(c)->crpriv_flags)

68 #define PRIV_SETBYTES    (PRIV_NSET * PRIV_SETSIZE * sizeof (priv_chunk_t))

70 #define PRIV_EISAWARE(c) ((CR_FLAGS(c) & PRIV_AWARE) || (c)->cr_uid != 0)
71 #define PRIV_PISAWARE(c) ((CR_FLAGS(c) & PRIV_AWARE) || \
72     ((c)->cr_uid != 0 && (c)->cr_suid != 0 && \
73     (c)->cr_ruid != 0))

75 #define CR_OEPRIV(c)     (*(PRIV_EISAWARE(c) ? &CR_EPRIV(c) : &CR_LPRIV(c)))
76 #define CR_OPPIV(c)     (*(PRIV_PISAWARE(c) ? &CR_PPRIV(c) : &CR_LPRIV(c)))

78 #define PRIV_VALIDSET(s) ((s) >= 0 && (s) < PRIV_NSET)
79 #define PRIV_VALIDOP(op) ((op) >= PRIV_ON && (op) <= PRIV_SET)

81 #define PRIV_FULLSET    &priv_fullset /* Require full set */

83 /*
84  * Privilege macros bits manipulation macros; DEBUG kernels will
85  * ASSERT() that privileges are not out of range.
86  */
87 #ifndef NBBY
88 #define NBBY            8
89 #endif

91 #define __NBWRD        (NBBY * sizeof (priv_chunk_t))

93 #define privmask(n)    (1U << ((__NBWRD - 1) - ((n) % __NBWRD)))
94 #define privword(n)    ((n)/__NBWRD)

96 /*
97  * PRIV_ADDSET(a, b) sets privilege "b" in privilege set "a".
98  * PRIV_DELSET(a, b) clears privilege "b" in privilege set "a".
99  * PRIV_ISMEMBER(a, b) tests if privilege 'b' is asserted in privilege set 'a'.
100 * PRIV_ASSERT(a, b) sets privilege "b" in privilege set "a".
101 * PRIV_CLEAR(a,b) clears privilege "b" in privilege set "a".
102 * PRIV_ISASSERT tests if privilege 'b' is asserted in privilege set 'a'.
103 */

102 #define __PRIV_ADDSET(a, b) ((a)->pbits[privword(b)] |= privmask(b))
103 #define __PRIV_DELSET(a, b) ((a)->pbits[privword(b)] &= ~privmask(b))
104 #define __PRIV_ISMEMBER(a, b) ((a)->pbits[privword(b)] & privmask(b))
104 #define __PRIV_ASSERT(a, b) ((a)->pbits[privword(b)] |= privmask(b))
105 #define __PRIV_CLEAR(a, b) ((a)->pbits[privword(b)] &= ~privmask(b))
106 #define __PRIV_ISASSERT(a, b) ((a)->pbits[privword(b)] & privmask(b))

106 #ifdef DEBUG
107 #define PRIV_DELSET(a, b)    priv_delset((a), (b))
108 #define PRIV_ADDSET(a, b)    priv_addset((a), (b))
109 #define PRIV_ISMEMBER(a, b)  priv_ismember((a), (b))
109 #define PRIV_CLEAR(a, b)    priv_delset((a), (b))
110 #define PRIV_ASSERT(a, b)    priv_addset((a), (b))
111 #define PRIV_ISASSERT(a, b)  priv_ismember((a), (b))
112 #else
111 #define PRIV_DELSET(a, b)    __PRIV_DELSET((a), (b))
112 #define PRIV_ADDSET(a, b)    __PRIV_ADDSET((a), (b))
113 #define PRIV_ISMEMBER(a, b)  __PRIV_ISMEMBER((a), (b))
113 #define PRIV_CLEAR(a, b)    __PRIV_CLEAR((a), (b))
114 #define PRIV_ASSERT(a, b)    __PRIV_ASSERT((a), (b))
115 #define PRIV_ISASSERT(a, b)  __PRIV_ISASSERT((a), (b))
114 #endif
```

new/usr/src/uts/common/sys/priv_impl.h

3

```
116 #endif /* _KERNEL */
```

```
118 #ifdef __cplusplus  
119 }
```

```
_____unchanged_portion_omitted_
```

```

*****
29673 Thu Jun 30 22:03:43 2016
new/usr/src/uts/common/sys/proc.h
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */

22 /*
23  * Copyright (c) 1988, 2010, Oracle and/or its affiliates. All rights reserved.
24 */

26 /*      Copyright (c) 1984, 1986, 1987, 1988, 1989 AT&T */
27 /*      All Rights Reserved */

29 #ifndef _SYS_PROC_H
30 #define _SYS_PROC_H

32 #include <sys/time.h>
33 #include <sys/thread.h>
34 #include <sys/cred.h>
35 #include <sys/user.h>
36 #include <sys/watchpoint.h>
37 #include <sys/timer.h>
38 #if defined(__x86)
39 #include <sys/tss.h>
40 #include <sys/segments.h>
41 #endif
42 #include <sys/utrap.h>
43 #include <sys/model.h>
44 #include <sys/refstr.h>
45 #include <sys/avl.h>
46 #include <sys/rctl.h>
47 #include <sys/list.h>
48 #include <sys/avl.h>
49 #include <sys/door_impl.h>
50 #include <sys/signalfd.h>
51 #include <sys/secflags.h>
52 #endif /* !codereview */

54 #ifdef __cplusplus
55 extern "C" {
56 #endif

58 /*

```

```

59  * Profile arguments.
60  */
61 struct prof {
62     void                *pr_base;    /* buffer base */
63     uintptr_t          pr_off;      /* pc offset */
64     size_t             pr_size;     /* buffer size */
65     uint32_t           pr_scale;    /* pc scaling */
66     long               pr_samples;  /* sample count */
67 };

69 /*
70  * An lwp directory entry.
71  * If le_thread != NULL, this is an active lwp.
72  * If le_thread == NULL, this is an unrealed zombie lwp.
73  */
74 typedef struct lwpent {
75     kthread_t          *le_thread;   /* the active lwp, NULL if zombie */
76     id_t               le_lwpid;     /* its lwpid (t->t_tid) */
77     uint16_t           le_waiters;   /* total number of lwp_wait()ers */
78     uint16_t           le_dwaiters; /* number that are daemons */
79     clock_t            le_start;     /* start time of this lwp */
80     struct vnode       *le_trace;    /* pointer to /proc lwp vnode */
81 } lwpent_t;

83 typedef struct pctxop {
84     void (*save_op)(void *);        /* function to invoke to save ctx */
85     void (*restore_op)(void *);     /* function to invoke to restore ctx */
86     void (*fork_op)(void *, void *); /* invoke to fork context */
87     void (*exit_op)(void *);        /* invoked during process exit */
88     void (*free_op)(void *, int);   /* function which frees the context */
89     void *arg;                      /* argument to above functions */
90     struct pctxop *next;            /* next pcontext ops */
91 } pctxop_t;

93 /*
94  * Elements of the lwp directory, p->p_lwpdir[].
95  *
96  * We allocate lwp directory entries separately from lwp directory
97  * elements because the lwp directory must be allocated as an array.
98  * The number of lwps can grow quite large and we want to keep the
99  * size of the kmem_alloc()d directory as small as possible.
100 *
101 * If ld_entry == NULL, the entry is free and is on the free list,
102 * p->p_lwpfree, linked through ld_next. If ld_entry != NULL, the
103 * entry is used and ld_next is the thread-id hash link pointer.
104 */
105 typedef struct lwpdir {
106     struct lwpdir *ld_next;         /* hash chain or free list */
107     struct lwpent *ld_entry;        /* lwp directory entry */
108 } lwpdir_t;

110 /*
111  * Element of the p_tidhash thread-id (lwpid) hash table.
112  */
113 typedef struct tidhash {
114     kmutex_t          th_lock;
115     lwpdir_t          *th_list;
116 } tidhash_t;

118 /*
119  * Retired tidhash hash tables.
120  */
121 typedef struct ret_tidhash {
122     struct ret_tidhash *rth_next;
123     tidhash_t          *rth_tidhash;
124     uint_t             rth_tidhash_sz;

```

```

125 } ret_tidhash_t;

127 struct pool;
128 struct task;
129 struct zone;
130 struct brand;
131 struct corectl_path;
132 struct corectl_content;

134 /*
135  * One structure allocated per active process. It contains all
136  * data needed about the process while the process may be swapped
137  * out. Other per-process data (user.h) is also inside the proc structure.
138  * Lightweight-process data (lwp.h) and the kernel stack may be swapped out.
139  */
140 typedef struct proc {
141     /*
142      * Fields requiring no explicit locking
143      */
144     struct vnode *p_exec;          /* pointer to a.out vnode */
145     struct as *p_as;              /* process address space pointer */
146     struct plock *p_lockp;        /* ptr to proc struct's mutex lock */
147     kmutex_t p_crlock;           /* lock for p_cred */
148     struct cred *p_cred;         /* process credentials */
149     /*
150      * Fields protected by pidlock
151      */
152     int p_swapcnt;               /* number of swapped out lwps */
153     char p_stat;                  /* status of process */
154     char p_wcode;                 /* current wait code */
155     ushort_t p_pidflag;          /* flags protected only by pidlock */
156     int p_wdata;                  /* current wait return value */
157     pid_t p_ppid;                 /* process id of parent */
158     struct proc *p_link;          /* forward link */
159     struct proc *p_parent;        /* ptr to parent process */
160     struct proc *p_child;         /* ptr to first child process */
161     struct proc *p_sibling;       /* ptr to next sibling proc on chain */
162     struct proc *p_psibling;      /* ptr to prev sibling proc on chain */
163     struct proc *p_sibling_ns;    /* prt to siblings with new state */
164     struct proc *p_child_ns;     /* prt to children with new state */
165     struct proc *p_next;          /* active chain link next */
166     struct proc *p_prev;          /* active chain link prev */
167     struct proc *p_nextofkin;     /* gets accounting info at exit */
168     struct proc *p_orphan;
169     struct proc *p_nextorph;
170     struct proc *p_pglink;        /* process group hash chain link next */
171     struct proc *p_ppglink;       /* process group hash chain link prev */
172     struct sess *p_sessp;         /* session information */
173     struct pid *p_pidp;          /* process ID info */
174     struct pid *p_pgidp;         /* process group ID info */
175     /*
176      * Fields protected by p_lock
177      */
178     kcondvar_t p_cv;              /* proc struct's condition variable */
179     kcondvar_t p_flag_cv;
180     kcondvar_t p_lwpexit;         /* waiting for some lwp to exit */
181     kcondvar_t p_holdlwps;       /* process is waiting for its lwps */
182     /* to be held. */
183     uint_t p_proc_flag;           /* /proc-related flags */
184     uint_t p_flag;                /* protected while set. */
185     /* flags defined below */
186     clock_t p_utime;              /* user time, this process */
187     clock_t p_stime;              /* system time, this process */
188     clock_t p_cstime;             /* sum of children's user time */
189     clock_t p_cstime;             /* sum of children's system time */
190     avl_tree_t *p_segacct;        /* System V shared segment list */

```

```

191     avl_tree_t *p_semacct;        /* System V semaphore undo list */
192     caddr_t p_bssbase;           /* base addr of last bss below heap */
193     caddr_t p_brkbase;          /* base addr of heap */
194     size_t p_brksize;           /* heap size in bytes */
195     uint_t p_brkpagesz;         /* preferred heap max page size code */
196     /*
197      * Per process signal stuff.
198      */
199     k_sigset_t p_sig;            /* signals pending to this process */
200     k_sigset_t p_extsig;         /* signals sent from another contract */
201     k_sigset_t p_ignore;        /* ignore when generated */
202     k_sigset_t p_siginfo;       /* gets signal info with signal */
203     void *p_sigfd;              /* signalfd support state */
204     struct sigqueue *p_sigqueue; /* queued siginfo structures */
205     struct sigqhdr *p_sigqhdr;  /* hdr to sigqueue structure pool */
206     struct sigqhdr *p_sighdr;   /* hdr to signotify structure pool */
207     uchar_t p_stopsig;          /* jobcontrol stop signal */

209     /*
210      * Special per-process flag when set will fix misaligned memory
211      * references.
212      */
213     char p_fixalignment;

215     /*
216      * Per process lwp and kernel thread stuff
217      */
218     id_t p_lwpid;                /* most recently allocated lwpid */
219     int p_lwpcnt;                /* number of lwps in this process */
220     int p_lwprcnt;              /* number of not stopped lwps */
221     int p_lwpdaemon;            /* number of TP_DAEMON lwps */
222     int p_lwpwait;              /* number of lwps in lwp_wait() */
223     int p_lwpdwait;             /* number of daemons in lwp_wait() */
224     int p_zombcnt;              /* number of zombie lwps */
225     kthread_t *p_tlist;         /* circular list of threads */
226     lwpdir_t *p_lwpdir;         /* thread (lwp) directory */
227     lwpdir_t *p_lwpfree;        /* p_lwpdir free list */
228     tidhash_t *p_tidhash;       /* tid (lwpid) lookup hash table */
229     uint_t p_lwpdir_sz;         /* number of p_lwpdir[] entries */
230     uint_t p_tidhash_sz;        /* number of p_tidhash[] entries */
231     ret_tidhash_t *p_ret_tidhash; /* retired tidhash hash tables */
232     uint64_t p_lgrpset;         /* unprotected hint of set of lgrps */
233     /* on which process has threads */
234     volatile lgrp_id_t p_tl_lgrp; /* main's thread lgroup id */
235     volatile lgrp_id_t p_tr_lgrp; /* text replica's lgroup id */
236 #if defined(LP64)
237     uintptr_t p_lgrpres2;       /* reserved for lgrp migration */
238 #endif

239     /*
240      * /proc (process filesystem) debugger interface stuff.
241      */
242     k_sigset_t p_sigmask;        /* mask of traced signals (/proc) */
243     k_filtset_t p_filtmask;     /* mask of traced faults (/proc) */
244     struct vnode *p_trace;       /* pointer to primary /proc vnode */
245     struct vnode *p_plist;       /* list of /proc vnodes for process */
246     kthread_t *p_agenttp;       /* thread ptr for /proc agent lwp */
247     avl_tree_t p_warea;         /* list of watched areas */
248     avl_tree_t p_wpage;         /* remembered watched pages (vfork) */
249     watched_page_t *p_wprot;    /* pages that need to have prot set */
250     int p_mapcnt;               /* number of active pr_mappage()s */
251     kmutex_t p_maplock;         /* lock for pr_mappage() */
252     struct proc *p_rlink;       /* linked list for server */
253     kcondvar_t p_srchan_cv;
254     size_t p_stksize;           /* process stack size in bytes */
255     uint_t p_stkpagesz;         /* preferred stack max page size code */

```

```

257 /*
258  * Microstate accounting, resource usage, and real-time profiling
259  */
260 hrtime_t p_mstart; /* hi-res process start time */
261 hrtime_t p_mterm; /* hi-res process termination time */
262 hrtime_t p_mlreal; /* elapsed time sum over defunct lwps */
263 hrtime_t p_acct[NMSTATES]; /* microstate sum over defunct lwps */
264 hrtime_t p_cacct[NMSTATES]; /* microstate sum over child procs */
265 struct lrusage p_ru; /* lrusage sum over defunct lwps */
266 struct lrusage p_cru; /* lrusage sum over child procs */
267 struct itimerval p_rprof_timer; /* ITIMER_REALPROF interval timer */
268 uintptr_t p_rprof_cyclic; /* ITIMER_REALPROF cyclic */
269 uint_t p_defunct; /* number of defunct lwps */
270 /*
271  * profiling. A lock is used in the event of multiple lwp's
272  * using the same profiling base/size.
273  */
274 kmutex_t p_plock; /* protects user profile arguments */
275 struct prof p_prof; /* profile arguments */
276
277 /*
278  * Doors.
279  */
280 door_pool_t p_server_threads; /* common thread pool */
281 struct door_node *p_door_list; /* active doors */
282 struct door_node *p_unref_list;
283 kcondvar_t p_unref_cv;
284 char p_unref_thread; /* unref thread created */
285
286 /*
287  * Kernel probes
288  */
289 uchar_t p_tnf_flags;
290
291 /*
292  * Solaris Audit
293  */
294 struct p_audit_data *p_audit_data; /* per process audit structure */
295
296 pctxop_t *p_pctx;
297
298 #if defined(__x86)
299 /*
300  * LDT support.
301  */
302 kmutex_t p_ldtlock; /* protects the following fields */
303 user_desc_t *p_ldt; /* Pointer to private LDT */
304 system_desc_t p_ldt_desc; /* segment descriptor for private LDT */
305 ushort_t p_ldtlimit; /* highest selector used */
306 #endif
307 size_t p_swrss; /* resident set size before last swap */
308 struct aio *p_aio; /* pointer to async I/O struct */
309 struct itimer **p_itimer; /* interval timers */
310 timeout_id_t p_alarmid; /* alarm's timeout id */
311 caddr_t p_usrstack; /* top of the process stack */
312 uint_t p_stkprot; /* stack memory protection */
313 uint_t p_datprot; /* data memory protection */
314 model_t p_model; /* data model determined at exec time */
315 struct lwpchan_data *p_lcp; /* lwpchan cache */
316 kmutex_t p_lcp_lock; /* protects assignments to p_lcp */
317 utrap_handler_t *p_utrap; /* pointer to user trap handlers */
318 struct corectl_path *p_corefile; /* pattern for core file */
319 struct task *p_task; /* our containing task */
320 struct proc *p_taskprev; /* ptr to previous process in task */
321 struct proc *p_tasknext; /* ptr to next process in task */
322 kmutex_t p_sc_lock; /* protects p_pagep */

```

```

323 struct sc_page_ctl *p_pagep; /* list of process's shared pages */
324 struct rctl_set *p_rctls; /* resource controls for this process */
325 rlim64_t p_stk_ctl; /* currently enforced stack size */
326 rlim64_t p_fsz_ctl; /* currently enforced file size */
327 rlim64_t p_vmem_ctl; /* currently enforced addr-space size */
328 rlim64_t p_fno_ctl; /* currently enforced file-desc limit */
329 pid_t p_ancpid; /* ancestor pid, used by exact */
330 struct itimerval p_realitimer; /* real interval timer */
331 timeout_id_t p_itimerid; /* real interval timer's timeout id */
332 struct corectl_content *p_content; /* content of core file */
333
334 avl_tree_t p_ct_held; /* held contracts */
335 struct ct_queue **p_ct_queue; /* process-type event queues */
336
337 struct cont_process *p_ct_process; /* process contract */
338 list_node_t p_ct_member; /* process contract membership */
339 sigqueue_t *p_killsp; /* sigqueue pointer for SIGKILL */
340
341 int p_dtrace_probes; /* are there probes for this proc? */
342 uint64_t p_dtrace_count; /* number of DTrace tracepoints */
343 /* (protected by P_PR_LOCK) */
344 void *p_dtrace_helpers; /* DTrace helpers, if any */
345 struct pool *p_pool; /* pointer to containing pool */
346 kcondvar_t p_poolcv; /* synchronization with pools */
347 uint_t p_poolcnt; /* # threads inside pool barrier */
348 uint_t p_poolflag; /* pool-related flags (see below) */
349 uintptr_t p_portcnt; /* event ports counter */
350 struct zone *p_zone; /* zone in which process lives */
351 struct vnode *p_execdir; /* directory that p_exec came from */
352 struct brand *p_brand; /* process's brand */
353 void *p_brand_data; /* per-process brand state */
354 psecflags_t p_secflags; /* per-process security flags */
355 #endif /* ! codereview */
356
357 /* additional lock to protect p_sessp (but not its contents) */
358 kmutex_t p_splck;
359 rctl_qty_t p_locked_mem; /* locked memory charged to proc */
360 /* protected by p_lock */
361 rctl_qty_t p_crypto_mem; /* /dev/crypto memory charged to proc */
362 /* protected by p_lock */
363 clock_t p_pttime; /* buffered task time */
364
365 /*
366  * The user structure
367  */
368 struct user p_user; /* (see sys/user.h) */
369 } proc_t;
370
371 #define PROC_T /* headers relying on proc_t are OK */
372
373 #ifdef _KERNEL
374
375 /* active process chain */
376
377 extern proc_t *practive;
378
379 /* Well known processes */
380
381 extern proc_t *proc_sched; /* memory scheduler */
382 extern proc_t *proc_init; /* init */
383 extern proc_t *proc_pageout; /* pageout daemon */
384 extern proc_t *proc_fsflush; /* filesystem sync-er */
385
386 #endif /* _KERNEL */
387
388 /*

```

```

389 * Stuff to keep track of the number of processes each uid has.
390 * It is tracked on a per-zone basis; that is, if users in different
391 * zones have the same uid, they are tracked separately.
392 *
393 * A structure is allocated when a new <uid,zoneid> pair shows up
394 * There is a hash to find each structure.
395 */
396 struct upcount {
397     struct upcount *up_next;
398     uid_t          up_uid;
399     zoneid_t       up_zoneid;
400     uint_t         up_count;
401 };

403 /* process ID info */

405 struct pid {
406     unsigned int pid_prinactive :1;
407     unsigned int pid_pgorphaned :1;
408     unsigned int pid_padding :6; /* used to be pid_ref, now an int */
409     unsigned int pid_prslot :24;
410     pid_t pid_id;
411     struct proc *pid_pglink;
412     struct proc *pid_pgtail;
413     struct pid *pid_link;
414     uint_t pid_ref;
415 };

417 #define p_pgrp p_pgidp->pid_id
418 #define p_pid p_pidp->pid_id
419 #define p_slot p_pidp->pid_prslot
420 #define p_detached p_pgidp->pid_pgorphaned

422 #define PID_HOLD(pidp)  ASSERT(MUTEX_HELD(&pidlock)); \
423     ++(pidp)->pid_ref;
424 #define PID_RELE(pidp)  ASSERT(MUTEX_HELD(&pidlock)); \
425     (pidp)->pid_ref > 1 ? \
426     --(pidp)->pid_ref : pid_rele(pidp);

428 /*
429 * Structure containing persistent process lock. The structure and
430 * macro allow "mutex_enter(&p->p_lock)" to continue working.
431 */
432 struct plock {
433     kmutex_t pl_lock;
434 };
435 #define p_lock p_lockp->pl_lock

437 #ifndef _KERNEL
438 extern proc_t p0; /* process 0 */
439 extern struct plock p0lock; /* p0's plock */
440 extern struct pid pid0; /* p0's pid */

442 /* pid_allocate() flags */
443 #define PID_ALLOC_PROC 0x0001 /* assign a /proc slot as well */

445 #endif /* _KERNEL */

447 /* stat codes */

449 #define SSLEEP 1 /* awaiting an event */
450 #define SRUN 2 /* runnable */
451 #define SZOMB 3 /* process terminated but not waited for */
452 #define SSTOP 4 /* process stopped by debugger */
453 #define SIDL 5 /* intermediate state in process creation */
454 #define SONPROC 6 /* process is being run on a processor */

```

```

455 #define SWAIT 7 /* process is waiting to become runnable */

457 /* p_pidflag codes */
458 #define CLDPEND 0x0001 /* have yet to post a SIGCHLD to the parent */
459 #define CLDCONT 0x0002 /* child has notified parent of CLD_CONTINUE */
460 #define CLDNOSIGCHLD 0x0004 /* do not post SIGCHLD when child terminates */
461 #define CLDWAITPID 0x0008 /* only waitid(P_PID, pid) can reap the child */

463 /* p_proc_flag codes -- these flags are mostly private to /proc */
464 #define P_PR_TRACE 0x0001 /* signal, fault or syscall tracing via /proc */
465 #define P_PR_PTRACE 0x0002 /* ptrace() compatibility mode */
466 #define P_PR_FORK 0x0004 /* child inherits tracing flags */
467 #define P_PR_LOCK 0x0008 /* process locked by /proc */
468 #define P_PR_ASYNC 0x0010 /* asynchronous stopping via /proc */
469 #define P_PR_EXEC 0x0020 /* process is in exec() */
470 #define P_PR_BPTADJ 0x0040 /* adjust pc on breakpoint trap */
471 #define P_PR_RUNLCL 0x0080 /* set process running on last /proc close */
472 #define P_PR_KILLLCL 0x0100 /* kill process on last /proc close */

474 /*
475 * p_flag codes
476 *
477 * note that two of these flags, SMSACCT and SSYS, are exported to /proc's
478 * psinfo_t.p_flag field. Historically, all were, but since they are
479 * implementation dependant, we only export the ones people have come to
480 * rely upon. Hence, the bit positions of SSYS and SMSACCT should not be
481 * altered.
482 */
483 #define SSYS 0x00000001 /* system (resident) process */
484 #define SEXITING 0x00000002 /* process is exiting */
485 #define SITBUSY 0x00000004 /* setitimer(ITIMER_REAL) in progress */
486 #define SFORKING 0x00000008 /* tells called functions that we're forking */
487 #define SWATCHOK 0x00000010 /* proc in acceptable state for watchpoints */
488 #define SKILLED 0x000000100 /* SIGKILL has been posted to the process */
489 #define SSCONT 0x00000200 /* SIGCONT has been posted to the process */
490 #define SZONETOP 0x00000400 /* process has no valid PPID in its zone */
491 #define SEXTKILLED 0x00000800 /* SKILLED is from another contract */
492 #define SUGID 0x00002000 /* process was result of set[ug]id exec */
493 #define SEXECED 0x00004000 /* this process has execed */
494 #define SJCTL 0x00010000 /* SIGCHLD sent when children stop/continue */
495 #define SNOWAIT 0x00020000 /* children never become zombies */
496 #define SVFORK 0x00040000 /* child of vfork that has not yet exec'd */
497 #define SVFWAIT 0x00080000 /* parent of vfork waiting for child to exec */
498 #define SEXITLWPS 0x00100000 /* have lwps exit within the process */
499 #define SHOLDFORK 0x00200000 /* hold lwps where they're cloneable */
500 #define SHOLDFORK1 0x00800000 /* hold lwps in place (not cloning) */
501 #define SCOREDUMP 0x01000000 /* process is dumping core */
502 #define SMSACCT 0x02000000 /* process is keeping micro-state accounting */
503 #define SLPWRAP 0x04000000 /* process has wrapped its lwp ids */
504 #define SAUTOLPG 0x08000000 /* kernel controls page sizes */
505 #define SNOCD 0x10000000 /* new creds from VSxID, do not coredump */
506 #define SHOLDWATCH 0x20000000 /* hold lwps for watchpoint operation */
507 #define SMSFORK 0x40000000 /* child inherits micro-state accounting */
508 #define SDCORE 0x80000000 /* process will attempt to dump core */

510 /*
511 * p_poolflag codes
512 *
513 * These flags are used to synchronize with the pool subsystem to allow
514 * re-binding of processes to new pools.
515 */
516 #define PEWAIT 0x0001 /* process should wait outside fork/exec/exit */
517 #define PEXITED 0x0002 /* process exited and about to become zombie */

519 /* Macro to convert proc pointer to a user block pointer */
520 #define PTOU(p) (&p)->p_user

```

```

522 #define tracing(p, sig) (sigismember(&(p)->p_sigmask, sig))
524 /* Macro to reduce unnecessary calls to issig() */
526 #define ISSIG(t, why)    ISSIG_FAST(t, ttolwp(t), ttoproc(t), why)
528 /*
529  * Fast version of ISSIG.
530  * 1. uses register pointers to lwp and proc instead of reloading them.
531  * 2. uses bit-wise OR of tests, since the usual case is that none of them
532  *   are true, this saves orcc's and branches.
533  * 3. load the signal flags instead of using sigisemtpy() macro which does
534  *   a branch to convert to boolean.
535  */
536 #define ISSIG_FAST(t, lwp, p, why)          \
537     (ISSIG_PENDING(t, lwp, p) && issig(why))
539 #define ISSIG_PENDING(t, lwp, p)          \
540     ((lwp->lwp_cursig |                     \
541      sigcheck((p), (t)) |                 \
542      (p->p_stopsig |                       \
543      (t->t_dtrace_stop |                  \
544      (t->t_dtrace_sig |                   \
545      ((t->t_proc_flag & (TP_PRSTOP|TP_HOLDLWP|TP_CHKPT|TP_PAUSE)) | \
546      ((p->p_flag & (SEXITLWPS|SKILLED|SHOLDFORK1|SHOLDWATCH))))
548 #define ISSTOP(sig)      (u.u_signal[sig-1] == SIG_DFL && \
549                          sigismember(&stopdefault, sig))
551 #define ISHOLD(p)        ((p)->p_flag & SHOLDFORK)
553 #define MUSTRETURN(p, t)    (ISHOLD(p) | (t)->t_activefd.a_stale)
555 /*
556  * Determine if there are any watchpoints active in the process.
557  */
558 #define pr_watch_active(p)    (avl_numnodes(&(p)->p_warea) != 0)
560 /* Reasons for calling issig() */
562 #define FORREAL          0    /* Usual side-effects */
563 #define JUSTLOOKING     1    /* Don't stop the process */
565 /* 'what' values for stop(PR_SUSPENDED, what) */
566 #define SUSPEND_NORMAL  0
567 #define SUSPEND_PAUSE   1
569 /* pseudo-flag to lwp_create() */
570 #define NOCLASS (-1)
572 /* unused scheduling class ID */
573 #define CLASS_UNUSED    (-2)
575 /* LWP stats updated via lwp_stats_update() */
576 typedef enum {
577     LWP_STAT_INBLK,
578     LWP_STAT_OUBLK,
579     LWP_STAT_MSGRCV,
580     LWP_STAT_MSGSND
581 } lwp_stat_id_t;
583 typedef struct prkillinfo {
584     int32_t prk_error;    /* errno */
585     int32_t prk_pad;     /* pad */
586     siginfo_t prk_info;  /* siginfo of killing signal */

```

```

587 } prkillinfo_t;
589 #ifdef _KERNEL
591 /* user profiling functions */
593 extern void profil_tick(uintptr_t);
595 /* process management functions */
597 extern int newproc(void (*)(), caddr_t, id_t, int, struct contract **, pid_t);
598 extern void vfwait(pid_t);
599 extern void proc_detach(proc_t *);
600 extern void freeproc(proc_t *);
601 extern void setrun(kthread_t *);
602 extern void setrun_locked(kthread_t *);
603 extern void exit(int, int);
604 extern int proc_exit(int, int);
605 extern void proc_is_exiting(proc_t *);
606 extern void relvm(void);
607 extern void add_ns(proc_t *, proc_t *);
608 extern void delete_ns(proc_t *, proc_t *);
609 extern void upcount_inc(uid_t, zoneid_t);
610 extern void upcount_dec(uid_t, zoneid_t);
611 extern int upcount_get(uid_t, zoneid_t);
612 #if defined(__x86)
613 extern selector_t setup_thrptr(proc_t *, uintptr_t);
614 extern void deferred_singlestep_trap(caddr_t);
615 #endif
617 extern void sigcld(proc_t *, sigqueue_t *);
618 extern void sigcld_delete(k_siginfo_t *);
619 extern void sigcld_repost(void);
620 extern int fsig(k_sigset_t *, kthread_t *);
621 extern void psig(void);
622 extern void stop(int, int);
623 extern int stop_on_fault(uint_t, k_siginfo_t *);
624 extern int issig(int);
625 extern int jobstopped(proc_t *);
626 extern void psignal(proc_t *, int);
627 extern void tsignal(kthread_t *, int);
628 extern void sigtoproc(proc_t *, kthread_t *, int);
629 extern void trapsig(k_siginfo_t *, int);
630 extern void realsigprof(int, int, int);
631 extern int eat_signal(kthread_t *, int);
632 extern int signal_is_blocked(kthread_t *, int);
633 extern int sigcheck(proc_t *, kthread_t *);
634 extern void sigdefault(proc_t *);
636 extern void pid_setmin(void);
637 extern pid_t pid_allocate(proc_t *, pid_t, int);
638 extern int pid_rele(struct pid *);
639 extern void pid_exit(proc_t *, struct task *);
640 extern void proc_entry_free(struct pid *);
641 extern proc_t *prfind(pid_t);
642 extern proc_t *prfind_zone(pid_t, zoneid_t);
643 extern proc_t *pgfind(pid_t);
644 extern proc_t *pgfind_zone(pid_t, zoneid_t);
645 extern proc_t *sprlock(pid_t);
646 extern proc_t *sprlock_zone(pid_t, zoneid_t);
647 extern int sprtrylock_proc(proc_t *);
648 extern void sprwaitlock_proc(proc_t *);
649 extern void sprlock_proc(proc_t *);
650 extern void sprunlock(proc_t *);
651 extern void pid_init(void);
652 extern proc_t *pid_entry(int);

```

```

653 extern int pid_slot(proc_t *);
654 extern void signal(pid_t, int);
655 extern void prsignal(struct pid *, int);
656 extern int uread(proc_t *, void *, size_t, uintptr_t);
657 extern int uwrite(proc_t *, void *, size_t, uintptr_t);

659 extern void pgsignal(struct pid *, int);
660 extern void pgjoin(proc_t *, struct pid *);
661 extern void pgcreate(proc_t *);
662 extern void pgexit(proc_t *);
663 extern void pgdetach(proc_t *);
664 extern int pgmembers(pid_t);

666 extern void init_mstate(kthread_t *, int);
667 extern int new_mstate(kthread_t *, int);
668 extern void restore_mstate(kthread_t *);
669 extern void term_mstate(kthread_t *);
670 extern void estimate_msacct(kthread_t *, hrtime_t);
671 extern void disable_msacct(proc_t *);
672 extern hrtime_t mstate_aggr_state(proc_t *, int);
673 extern hrtime_t mstate_thread_onproc_time(kthread_t *);
674 extern void mstate_systhread_times(kthread_t *, hrtime_t *, hrtime_t *);
675 extern void syscall_mstate(int, int);

677 extern uint_t cpu_update_pct(kthread_t *, hrtime_t);

679 extern void set_proc_pre_sys(proc_t *p);
680 extern void set_proc_post_sys(proc_t *p);
681 extern void set_proc_sys(proc_t *p);
682 extern void set_proc_ast(proc_t *p);
683 extern void set_all_proc_sys(void);
684 extern void set_all_zone_usr_proc_sys(zoneid_t);

686 /* thread function prototypes */

688 extern kthread_t *thread_create(
689     caddr_t stk,
690     size_t stksize,
691     void (*proc)(),
692     void *arg,
693     size_t len,
694     proc_t *pp,
695     int state,
696     pri_t pri);
697 extern void thread_exit(void) __NORETURN;
698 extern void thread_free(kthread_t *);
699 extern void thread_rele(kthread_t *);
700 extern void thread_join(kt_did_t);
701 extern int reaper(void);
702 extern void installctx(kthread_t *, void *, void (*)(), void (*)(),
703     void (*)(), void (*)(), void (*)(), void (*)());
704 extern int removectx(kthread_t *, void *, void (*)(), void (*)(),
705     void (*)(), void (*)(), void (*)(), void (*)());
706 extern void savectx(kthread_t *);
707 extern void restorectx(kthread_t *);
708 extern void forkctx(kthread_t *, kthread_t *);
709 extern int lwprocctx(kthread_t *, kthread_t *);
710 extern void exitctx(kthread_t *);
711 extern void freectx(kthread_t *, int);
712 extern void installpctx(proc_t *, void *, void (*)(), void (*)(),
713     void (*)(), void (*)(), void (*)());
714 extern int removepctx(proc_t *, void *, void (*)(), void (*)(),
715     void (*)(), void (*)(), void (*)());
716 extern void savepctx(proc_t *);
717 extern void restorepctx(proc_t *);
718 extern void forkpctx(proc_t *, proc_t *);

```

```

719 extern void exitpctx(proc_t *);
720 extern void freepctx(proc_t *, int);
721 extern kthread_t *thread_unpin(void);
722 extern void thread_init(void);
723 extern void thread_load(kthread_t *, void (*)(), caddr_t, size_t);

725 extern void tsd_create(uint_t *, void (*) (void *));
726 extern void tsd_destroy(uint_t *);
727 extern void *tsd_getcreate(uint_t *, void (*) (void *), void (*) (void));
728 extern void *tsd_get(uint_t *);
729 extern int tsd_set(uint_t, void *);
730 extern void tsd_exit(void);
731 extern void *tsd_agent_get(kthread_t *, uint_t);
732 extern int tsd_agent_set(kthread_t *, uint_t, void *);

734 /* lwp function prototypes */

736 extern kthread_t *lwp_kernel_create(proc_t *, void (*)(), void *, int, pri_t);
737 extern klwp_t *lwp_create(
738     (*proc)(),
739     caddr_t arg,
740     size_t len,
741     proc_t *p,
742     int state,
743     int pri,
744     const k_sigset_t *smask,
745     int cid,
746     id_t lwpid);
747 extern kthread_t *idtot(proc_t *, id_t);
748 extern void lwp_hash_in(proc_t *, lwpent_t *, tidhash_t *, uint_t, int);
749 extern void lwp_hash_out(proc_t *, id_t);
750 extern lwpdir_t *lwp_hash_lookup(proc_t *, id_t);
751 extern lwpdir_t *lwp_hash_lookup_and_lock(proc_t *, id_t, kmutex_t **);
752 extern void lwp_create_done(kthread_t *);
753 extern void lwp_exit(void);
754 extern void lwp_pcb_exit(void);
755 extern void lwp_cleanup(void);
756 extern int lwp_suspend(kthread_t *);
757 extern void lwp_continue(kthread_t *);
758 extern void holdlwp(void);
759 extern void stoplwp(void);
760 extern int holdlwps(int);
761 extern int holdwatch(void);
762 extern void pokelwps(proc_t *);
763 extern void continuelwps(proc_t *);
764 extern int exitlwps(int);
765 extern void lwp_ctmpl_copy(klwp_t *, klwp_t *);
766 extern void lwp_ctmpl_clear(klwp_t *);
767 extern klwp_t *forklwp(klwp_t *, proc_t *, id_t);
768 extern void lwp_load(klwp_t *, gregset_t, uintptr_t);
769 extern void lwp_setrval(klwp_t *, int, int);
770 extern void lwp_forkregs(klwp_t *, klwp_t *);
771 extern void lwp_freeregs(klwp_t *, int);
772 extern caddr_t lwp_stk_init(klwp_t *, caddr_t);
773 extern void lwp_stk_cache_init(void);
774 extern void lwp_stk_fini(klwp_t *);
775 extern void lwp_installctx(klwp_t *);
776 extern void lwp_rtt(void);
777 extern void lwp_rtt_initial(void);
778 extern int lwp_setprivate(klwp_t *, int, uintptr_t);
779 extern void lwp_stat_update(lwp_stat_id_t, long);
780 extern void lwp_attach_brand_hdlrs(klwp_t *);
781 extern void lwp_detach_brand_hdlrs(klwp_t *);

783 #if defined(__sparcv9)
784 extern void lwp_mmodel_newlwp(void);

```



```
785 extern void lwp_mmodel_shared_as(caddr_t, size_t);
786 #define LWP_MMODEL_NEWLWP() lwp_mmodel_newlwp()
787 #define LWP_MMODEL_SHARED_AS(addr, sz) lwp_mmodel_shared_as((addr), (sz))
788 #else
789 #define LWP_MMODEL_NEWLWP()
790 #define LWP_MMODEL_SHARED_AS(addr, sz)
791 #endif

793 /*
794 * Signal queue function prototypes. Must be here due to header ordering
795 * dependencies.
796 */
797 extern void sigqfree(proc_t *);
798 extern void siginfofree(sigqueue_t *);
799 extern void sigdeq(proc_t *, kthread_t *, int, sigqueue_t **);
800 extern void sigdelq(proc_t *, kthread_t *, int);
801 extern void sigaddq(proc_t *, kthread_t *, k_siginfo_t *, int);
802 extern void sigaddqa(proc_t *, kthread_t *, sigqueue_t *);
803 extern void siggsend(int, proc_t *, kthread_t *, sigqueue_t *);
804 extern void sigdupq(proc_t *, proc_t *);
805 extern int sigwillqueue(int, int);
806 extern sigqhdr_t *sigqhdralloc(size_t, uint_t);
807 extern sigqueue_t *sigqalloc(sigqhdr_t *);
808 extern void sigqhdrfree(sigqhdr_t *);
809 extern sigqueue_t *sigappend(k_sigset_t *, sigqueue_t *,
810     k_sigset_t *, sigqueue_t *);
811 extern sigqueue_t *sigprepend(k_sigset_t *, sigqueue_t *,
812     k_sigset_t *, sigqueue_t *);
813 extern void winfo(proc_t *, k_siginfo_t *, int);
814 extern int wstat(int, int);
815 extern int sendsig(int, k_siginfo_t *, void (*)());
816 #if defined(_SYSCALL32_IMPL)
817 extern int sendsig32(int, k_siginfo_t *, void (*)());
818 #endif

820 #endif /* _KERNEL */

822 #ifdef __cplusplus
823 }
824 #endif

826 #endif /* _SYS_PROC_H */
```

```

*****
35236 Thu Jun 30 22:03:44 2016
new/usr/src/uts/common/sys/procfs.h
Code review comments from jeffpc
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */

22 /*
23  * Copyright 2006 Sun Microsystems, Inc. All rights reserved.
24  * Use is subject to license terms.
25 */
26 /*
27  * Copyright 2012 DEY Storage Systems, Inc. All rights reserved.
28 */

30 #ifndef _SYS_PROCFS_H
31 #define _SYS_PROCFS_H

33 #ifdef __cplusplus
34 extern "C" {
35 #endif

37 /*
38  * This definition is temporary. Structured proc is the preferred API,
39  * and the older ioctl-based interface will be removed in a future version
40  * of Solaris. Until then, by default, including <sys/procfs.h> will
41  * provide the older ioctl-based /proc definitions. To get the structured
42  * /proc definitions, either include <procfs.h> or define _STRUCTURED_PROC
43  * to be 1 before including <sys/procfs.h>.
44 */
45 #ifndef _STRUCTURED_PROC
46 #define _STRUCTURED_PROC 0
47 #endif

49 #if !defined(_KERNEL) && _STRUCTURED_PROC == 0

51 #include <sys/old_procfs.h>

53 #else /* !defined(_KERNEL) && _STRUCTURED_PROC == 0 */

55 #include <sys/feature_tests.h>
56 #include <sys/types.h>
57 #include <sys/time_impl.h>

```

```

58 #include <sys/signal.h>
59 #include <sys/signinfo.h>
60 #include <sys/fault.h>
61 #include <sys/syscall.h>
62 #include <sys/pset.h>
63 #include <sys/procfs_isa.h>
64 #include <sys/priv.h>
65 #include <sys/stat.h>
66 #include <sys/param.h>
67 #include <sys/secflags.h>
68 #endif /* !codereview */

70 /*
71  * System call interfaces for /proc.
72 */

74 /*
75  * Control codes (long values) for messages written to ctl and lwpcctl files.
76 */
77 #define PCNULL 0L /* null request, advance to next message */
78 #define PCSTOP 1L /* direct process or lwp to stop and wait for stop */
79 #define PCDSTOP 2L /* direct process or lwp to stop */
80 #define PCWSTOP 3L /* wait for process or lwp to stop, no timeout */
81 #define PCTWSTOP 4L /* wait for stop, with long millisecond timeout arg */
82 #define PCRUN 5L /* make process/lwp runnable, w/ long flags argument */
83 #define PCCSIG 6L /* clear current signal from lwp */
84 #define PCCFAULT 7L /* clear current fault from lwp */
85 #define PCSSIG 8L /* set current signal from siginfo_t argument */
86 #define PCKILL 9L /* post a signal to process/lwp, long argument */
87 #define PCUNKILL 10L /* delete a pending signal from process/lwp, long arg */
88 #define PCSHOLD 11L /* set lwp signal mask from sigset_t argument */
89 #define PCSTRACE 12L /* set traced signal set from sigset_t argument */
90 #define PCSFAULT 13L /* set traced fault set from fltset_t argument */
91 #define PCSENTRY 14L /* set traced syscall entry set from sysset_t arg */
92 #define PCSEXIT 15L /* set traced syscall exit set from sysset_t arg */
93 #define PCSET 16L /* set modes from long argument */
94 #define PCUNSET 17L /* unset modes from long argument */
95 #define PCSREG 18L /* set lwp general registers from prgregset_t arg */
96 #define PCSFPREG 19L /* set lwp floating-point registers from prfpregset_t */
97 #define PCSXREG 20L /* set lwp extra registers from prxregset_t arg */
98 #define PCNICE 21L /* set nice priority from long argument */
99 #define PCSVADDR 22L /* set %pc virtual address from long argument */
100 #define PCWATCH 23L /* set/unset watched memory area from prwatch_t arg */
101 #define PCAGENT 24L /* create agent lwp with regs from prgregset_t arg */
102 #define PCREAD 25L /* read from the address space via priovec_t arg */
103 #define PCWRITE 26L /* write to the address space via priovec_t arg */
104 #define PCSCRED 27L /* set process credentials from pcred_t argument */
105 #define PCSASRS 28L /* set ancillary state registers from asrset_t arg */
106 #define PCSPRIV 29L /* set process privileges from prpriv_t argument */
107 #define PCSZONE 30L /* set zoneid from zoneid_t argument */
108 #define PCSCREDX 31L /* as PCSCRED but with supplemental groups */
109 /*
110  * PCRUN long operand flags.
111 */
112 #define PRCSIG 0x01 /* clear current signal, if any */
113 #define PRFAULT 0x02 /* clear current fault, if any */
114 #define PRSTEP 0x04 /* direct the lwp to single-step */
115 #define PRSABORT 0x08 /* abort syscall, if in syscall */
116 #define PRSTOP 0x10 /* set directed stop request */

118 /*
119  * lwp status file. /proc/<pid>/lwp/<lwpid>/lwpstatus
120 */
121 #define PRCLSZ 8 /* maximum size of scheduling class name */
122 #define PRSYSARGS 8 /* maximum number of syscall arguments */
123 typedef struct lwpstatus {

```

```

124 int pr_flags; /* flags (see below) */
125 id_t pr_lwpid; /* specific lwp identifier */
126 short pr_why; /* reason for lwp stop, if stopped */
127 short pr_what; /* more detailed reason */
128 short pr_cursig; /* current signal, if any */
129 short pr_padi;
130 siginfo_t pr_info; /* info associated with signal or fault */
131 sigset_t pr_lwppend; /* set of signals pending to the lwp */
132 sigset_t pr_lwphold; /* set of signals blocked by the lwp */
133 struct sigaction pr_action; /* signal action for current signal */
134 stack_t pr_altstack; /* alternate signal stack info */
135 uintptr_t pr_oldcontext; /* address of previous ucontext */
136 short pr_syscall; /* system call number (if in syscall) */
137 short pr_nsysarg; /* number of arguments to this syscall */
138 int pr_errno; /* errno for failed syscall, 0 if successful */
139 long pr_sysarg[PRSYSARGS]; /* arguments to this syscall */
140 long pr_rval1; /* primary syscall return value */
141 long pr_rval2; /* second syscall return value, if any */
142 char pr_clname[PRCLSZ]; /* scheduling class name */
143 timestruc_t pr_tstamp; /* real-time time stamp of stop */
144 timestruc_t pr_utime; /* lwp user cpu time */
145 timestruc_t pr_stime; /* lwp system cpu time */
146 int pr_filler[11 - 2 * sizeof(timestruc_t) / sizeof(int)];
147 int pr_errpriv; /* missing privilege */
148 uintptr_t pr_ustack; /* address of stack boundary data (stack_t) */
149 ulong_t pr_instr; /* current instruction */
150 prgregset_t pr_reg; /* general registers */
151 prfpregset_t pr_fpreg; /* floating-point registers */
152 } lwpstatus_t;

154 /*
155 * process status file. /proc/<pid>/status
156 */
157 typedef struct pstatus {
158 int pr_flags; /* flags (see below) */
159 int pr_nlwp; /* number of active lwps in the process */
160 pid_t pr_pid; /* process id */
161 pid_t pr_ppid; /* parent process id */
162 pid_t pr_pgid; /* process group id */
163 pid_t pr_sid; /* session id */
164 id_t pr_aslwpid; /* historical; now always zero */
165 id_t pr_agentid; /* lwp id of the /proc agent lwp, if any */
166 sigset_t pr_sigpend; /* set of process pending signals */
167 uintptr_t pr_brkbase; /* address of the process heap */
168 size_t pr_brksize; /* size of the process heap, in bytes */
169 uintptr_t pr_stkbase; /* address of the process stack */
170 size_t pr_stksize; /* size of the process stack, in bytes */
171 timestruc_t pr_utime; /* process user cpu time */
172 timestruc_t pr_stime; /* process system cpu time */
173 timestruc_t pr_cutime; /* sum of children's user times */
174 timestruc_t pr_cstime; /* sum of children's system times */
175 sigset_t pr_sigtrace; /* set of traced signals */
176 fltset_t pr_fltrace; /* set of traced faults */
177 sysset_t pr_sysentry; /* set of system calls traced on entry */
178 sysset_t pr_sysexit; /* set of system calls traced on exit */
179 char pr_dmodel; /* data model of the process (see below) */
180 char pr_pad[3];
181 taskid_t pr_taskid; /* task id */
182 projid_t pr_projid; /* project id */
183 int pr_nzomb; /* number of zombie lwps in the process */
184 zoneid_t pr_zoneid; /* zone id */
185 int pr_filler[15]; /* reserved for future use */
186 lwpstatus_t pr_lwp; /* status of the representative lwp */
187 } pstatus_t;

189 /*

```

```

190 * pr_flags (same values appear in both pstatus_t and lwpstatus_t pr_flags).
191 *
192 * These flags do *not* apply to psinfo_t.pr_flag or lwpsinfo_t.pr_flag
193 * (which are both deprecated).
194 */
195 /* The following flags apply to the specific or representative lwp */
196 #define PR_STOPPED 0x00000001 /* lwp is stopped */
197 #define PR_ISTOP 0x00000002 /* lwp is stopped on an event of interest */
198 #define PR_DSTOP 0x00000004 /* lwp has a stop directive in effect */
199 #define PR_STEP 0x00000008 /* lwp has a single-step directive in effect */
200 #define PR_SLEEP 0x00000010 /* lwp is sleeping in a system call */
201 #define PR_PCINVAL 0x00000020 /* contents of pr_instr undefined */
202 #define PR_ASLWP 0x00000040 /* obsolete flag; never set */
203 #define PR_AGENT 0x00000080 /* this lwp is the /proc agent lwp */
204 #define PR_DETACH 0x00000100 /* this is a detached lwp */
205 #define PR_DAEMON 0x00000200 /* this is a daemon lwp */
206 #define PR_IDLE 0x00000400 /* lwp is a cpu's idle thread */
207 /* The following flags apply to the process, not to an individual lwp */
208 #define PR_ISSYS 0x00001000 /* this is a system process */
209 #define PR_VFORKP 0x00002000 /* process is the parent of a vfork(d) child */
210 #define PR_ORPHAN 0x00004000 /* process's process group is orphaned */
211 #define PR_NOSIGCHLD 0x00008000 /* process will not generate SIGCHLD on exit */
212 #define PR_WAITPID 0x00010000 /* only waitid(P_PID, pid) can reap the child */
213 /* The following process flags are modes settable by PCSET/PCUNSET */
214 #define PR_FORK 0x00100000 /* inherit-on-fork is in effect */
215 #define PR_RLC 0x00200000 /* run-on-last-close is in effect */
216 #define PR_KLC 0x00400000 /* kill-on-last-close is in effect */
217 #define PR_ASYNC 0x00800000 /* asynchronous-stop is in effect */
218 #define PR_MSACCT 0x01000000 /* micro-state usage accounting is in effect */
219 #define PR_BPTADJ 0x02000000 /* breakpoint trap pc adjustment is in effect */
220 #define PR_PTRACE 0x04000000 /* ptrace-compatibility mode is in effect */
221 #define PR_MSFFORK 0x08000000 /* micro-state accounting inherited on fork */

222 /*
223 * See <sys/procfs_isa.h> for possible values of pr_dmodel.
224 */

227 /*
228 * Reasons for stopping (pr_why).
229 */
230 #define PR_REQUESTED 1
231 #define PR_SIGNALED 2
232 #define PR_SYSENTRY 3
233 #define PR_SYSEXIT 4
234 #define PR_JOBCONTROL 5
235 #define PR_FAULTED 6
236 #define PR_SUSPENDED 7
237 #define PR_CHECKPOINT 8

239 /*
240 * lwp ps(1) information file. /proc/<pid>/lwp/<lwpid>/lwpsinfo
241 */
242 #define PRFNSZ 16 /* Maximum size of execed filename */
243 typedef struct lwpsinfo {
244 int pr_flag; /* lwp flags (DEPRECATED; do not use) */
245 id_t pr_lwpid; /* lwp id */
246 uintptr_t pr_addr; /* internal address of lwp */
247 uintptr_t pr_wchan; /* wait addr for sleeping lwp */
248 char pr_stype; /* synchronization event type */
249 char pr_state; /* numeric lwp state */
250 char pr_sname; /* printable character for pr_state */
251 char pr_nice; /* nice for cpu usage */
252 short pr_syscall; /* system call number (if in syscall) */
253 char pr_oldpri; /* pre-SVR4, low value is high priority */
254 char pr_cpu; /* pre-SVR4, cpu usage for scheduling */
255 int pr_pri; /* priority, high value is high priority */

```

```

256      /* The following percent number is a 16-bit binary */
257      /* fraction [0 .. 1] with the binary point to the */
258      /* right of the high-order bit (1.0 == 0x8000) */
259      ushort_t pr_pctcpu;      /* % of recent cpu time used by this lwp */
260      ushort_t pr_pad;
261      timestruc_t pr_start;    /* lwp start time, from the epoch */
262      timestruc_t pr_time;     /* usr+sys cpu time for this lwp */
263      char pr_clname[PRCLSZ];  /* scheduling class name */
264      char pr_name[PRFNSZ];    /* name of system lwp */
265      processorid_t pr_onpro;  /* processor which last ran this lwp */
266      processorid_t pr_bindpro; /* processor to which lwp is bound */
267      psetid_t pr_bindpset;   /* processor set to which lwp is bound */
268      int pr_lgrp;            /* lwp home lgroup */
269      int pr_filler[4];       /* reserved for future use */
270 } lwpsinfo_t;

272 /*
273  * process ps(1) information file. /proc/<pid>/psinfo
274  */
275 #define PRARGSZ 80          /* number of chars of arguments */
276 typedef struct psinfo {
277     int pr_flag;            /* process flags (DEPRECATED; do not use) */
278     int pr_nlwp;           /* number of active lwps in the process */
279     pid_t pr_pid;          /* unique process id */
280     pid_t pr_ppid;         /* process id of parent */
281     pid_t pr_pgid;         /* pid of process group leader */
282     pid_t pr_sid;          /* session id */
283     uid_t pr_uid;          /* real user id */
284     uid_t pr_euid;         /* effective user id */
285     gid_t pr_gid;          /* real group id */
286     gid_t pr_egid;         /* effective group id */
287     uintptr_t pr_addr;     /* address of process */
288     size_t pr_size;        /* size of process image in Kbytes */
289     size_t pr_rssize;      /* resident set size in Kbytes */
290     size_t pr_pad1;
291     dev_t pr_ttydev;       /* controlling tty device (or PRNODEV) */
292     /* The following percent numbers are 16-bit binary */
293     /* fractions [0 .. 1] with the binary point to the */
294     /* right of the high-order bit (1.0 == 0x8000) */
295     ushort_t pr_pctcpu;    /* % of recent cpu time used by all lwps */
296     ushort_t pr_pctmem;    /* % of system memory used by process */
297     timestruc_t pr_start;  /* process start time, from the epoch */
298     timestruc_t pr_time;   /* usr+sys cpu time for this process */
299     timestruc_t pr_ctime;  /* usr+sys cpu time for reaped children */
300     char pr_fname[PRFNSZ]; /* name of execed file */
301     char pr_psargs[PRARGSZ]; /* initial characters of arg list */
302     int pr_wstat;          /* if zombie, the wait() status */
303     int pr_argc;           /* initial argument count */
304     uintptr_t pr_argv;     /* address of initial argument vector */
305     uintptr_t pr_envp;     /* address of initial environment vector */
306     char pr_dmodel;        /* data model of the process */
307     char pr_pad2[3];
308     taskid_t pr_taskid;    /* task id */
309     projid_t pr_projid;    /* project id */
310     int pr_nzomb;          /* number of zombie lwps in the process */
311     poolid_t pr_poolid;    /* pool id */
312     zoneid_t pr_zoneid;    /* zone id */
313     id_t pr_contract;      /* process contract */
314     int pr_filler[1];      /* reserved for future use */
315     lwpsinfo_t pr_lwp;     /* information for representative lwp */
316 } psinfo_t;

318 #define PRNODEV (dev_t)(-1) /* non-existent device */

320 /*
321  * Memory-map interface. /proc/<pid>/map /proc/<pid>/rmap

```

```

322  */
323 #define PRMAPSZ 64
324 typedef struct prmap {
325     uintptr_t pr_vaddr;     /* virtual address of mapping */
326     size_t pr_size;         /* size of mapping in bytes */
327     char pr_mapname[PRMAPSZ]; /* name in /proc/<pid>/object */
328     offset_t pr_offset;     /* offset into mapped object, if any */
329     int pr_mflags;         /* protection and attribute flags (see below) */
330     int pr_pagesize;        /* pagesize (bytes) for this mapping */
331     int pr_shmid;          /* SysV shmid, -1 if not SysV shared memory */
332     int pr_filler[1];      /* filler for future expansion */
333 } prmap_t;

335 /*
336  * HAT memory-map interface. /proc/<pid>/xmap
337  */
338 typedef struct prxmap {
339     uintptr_t pr_vaddr;     /* virtual address of mapping */
340     size_t pr_size;         /* size of mapping in bytes */
341     char pr_mapname[PRMAPSZ]; /* name in /proc/<pid>/object */
342     offset_t pr_offset;     /* offset into mapped object, if any */
343     int pr_mflags;         /* protection and attribute flags (see below) */
344     int pr_pagesize;        /* pagesize (bytes) for this mapping */
345     int pr_shmid;          /* SysV shmid, -1 if not SysV shared memory */
346     dev_t pr_dev;          /* st_dev from stat64() of mapped object, or PRNODEV */
347     uint64_t pr_ino;        /* st_ino from stat64() of mapped object, if any */
348     size_t pr_rss;         /* pages of resident memory */
349     size_t pr_anon;         /* pages of resident anonymous memory */
350     size_t pr_locked;      /* pages of locked memory */
351     size_t pr_pad;         /* currently unused */
352     uint64_t pr_hatpagesize; /* pagesize of the hat mapping */
353 #ifdef _ILP32
354     ulong_t pr_filler[6];  /* filler for future expansion */
355 #else
356     ulong_t pr_filler[7];  /* filler for future expansion */
357 #endif
358 } prxmap_t;

361 /* Protection and attribute flags */
362 #define MA_READ 0x04        /* readable by the traced process */
363 #define MA_WRITE 0x02      /* writable by the traced process */
364 #define MA_EXEC 0x01       /* executable by the traced process */
365 #define MA_SHARED 0x08     /* changes are shared by mapped object */
366 #define MA_ANON 0x40       /* anonymous memory (e.g. /dev/zero) */
367 #define MA_ISM 0x80        /* intimate shared mem (shared MMU resources) */
368 #define MA_NORESERVE 0x100 /* mapped with MAP_NORESERVE */
369 #define MA_SHM 0x200       /* System V shared memory */
370 #define MA_RESERVED1 0x400 /* reserved for future use */

372 /*
373  * These are obsolete and unreliable.
374  * They are included here only for historical compatibility.
375  */
376 #define MA_BREAK 0x10      /* grown by brk(2) */
377 #define MA_STACK 0x20     /* grown automatically on stack faults */

379 /*
380  * Process credentials. PCSCRED and /proc/<pid>/cred
381  */
382 typedef struct pcred {
383     uid_t pr_euid;         /* effective user id */
384     uid_t pr_ruid;         /* real user id */
385     uid_t pr_suid;         /* saved user id (from exec) */
386     gid_t pr_egid;         /* effective group id */
387     gid_t pr_rgid;         /* real group id */

```

```

388     gid_t   pr_sgid;      /* saved group id (from exec) */
389     int     pr_ngroups;   /* number of supplementary groups */
390     gid_t   pr_groups[1]; /* array of supplementary groups */
391 } pcred_t;

393 /*
394  * Process privileges.  PCSPRIV and /proc/<pid>/priv
395  */
396 typedef struct prpriv {
397     uint32_t   pr_nsets;      /* number of privilege set */
398     uint32_t   pr_setsize;    /* size of privilege set */
399     uint32_t   pr_infsize;    /* size of supplementary data */
400     priv_chunk_t pr_sets[1];  /* array of sets */
401 } prpriv_t;

403 #define PRSECFLAGS_VERSION_1      1
404 #define PRSECFLAGS_VERSION_CURRENT PRSECFLAGS_VERSION_1
405 typedef struct prsecflags {
406     uint32_t pr_version;
407     char pr_pad[4];
408     secflags_t pr_effective;
409     secflags_t pr_inherit;
410     secflags_t pr_lower;
411     secflags_t pr_upper;
412 } prsecflags_t;

414 #endif /* ! codereview */
415 /*
416  * Watchpoint interface.  PCWATCH and /proc/<pid>/watch
417  */
418 typedef struct prwatch {
419     uintptr_t pr_vaddr;      /* virtual address of watched area */
420     size_t pr_size;          /* size of watched area in bytes */
421     int pr_wflags;          /* watch type flags */
422     int pr_pad;
423 } prwatch_t;

425 /* pr_wflags */
426 #define WA_READ      0x04 /* trap on read access */
427 #define WA_WRITE     0x02 /* trap on write access */
428 #define WA_EXEC      0x01 /* trap on execute access */
429 #define WA_TRAPAFTER 0x08 /* trap after instruction completes */

431 /*
432  * PCREAD/PCWRITE I/O interface.
433  */
434 typedef struct priovec {
435     void *pio_base;          /* buffer in controlling process */
436     size_t pio_len;          /* size of read/write request */
437     off_t pio_offset;        /* virtual address in target process */
438 } priovec_t;

440 /*
441  * Resource usage.  /proc/<pid>/usage /proc/<pid>/lwp/<lwpid>/lwpusage
442  */
443 typedef struct prusage {
444     id_t pr_lwpid;           /* lwp id.  0: process or defunct */
445     int pr_count;            /* number of contributing lwps */
446     timestruc_t pr_tstamp;   /* current time stamp */
447     timestruc_t pr_create;    /* process/lwp creation time stamp */
448     timestruc_t pr_term;      /* process/lwp termination time stamp */
449     timestruc_t pr_rtime;     /* total lwp real (elapsed) time */
450     timestruc_t pr_utime;     /* user level cpu time */
451     timestruc_t pr_stime;     /* system call cpu time */
452     timestruc_t pr_ttime;     /* other system trap cpu time */
453     timestruc_t pr_tftime;    /* text page fault sleep time */

```

```

454     timestruc_t pr_dftime;    /* data page fault sleep time */
455     timestruc_t pr_kftime;    /* kernel page fault sleep time */
456     timestruc_t pr_ltime;     /* user lock wait sleep time */
457     timestruc_t pr_slptime;   /* all other sleep time */
458     timestruc_t pr_wtime;     /* wait-cpu (latency) time */
459     timestruc_t pr_stoptime;  /* stopped time */
460     timestruc_t filltime[6];  /* filler for future expansion */
461     ulong_t pr_minf;          /* minor page faults */
462     ulong_t pr_majf;          /* major page faults */
463     ulong_t pr_nswap;         /* swaps */
464     ulong_t pr_inblk;         /* input blocks */
465     ulong_t pr_oublk;         /* output blocks */
466     ulong_t pr_msnd;          /* messages sent */
467     ulong_t pr_mrcv;          /* messages received */
468     ulong_t pr_sigs;          /* signals received */
469     ulong_t pr_vctx;          /* voluntary context switches */
470     ulong_t pr_ictx;          /* involuntary context switches */
471     ulong_t pr_sysc;          /* system calls */
472     ulong_t pr_ioch;          /* chars read and written */
473     ulong_t pr_filler[10];    /* filler for future expansion */
474 } prusage_t;

476 /*
477  * Page data file.  /proc/<pid>/pagedata
478  */

480 /* page data file header */
481 typedef struct prpageheader {
482     timestruc_t pr_tstamp;    /* real time stamp */
483     long pr_nmap;             /* number of address space mappings */
484     long pr_npage;           /* total number of pages */
485 } prpageheader_t;

487 /* page data mapping header */
488 typedef struct prsmap {
489     uintptr_t pr_vaddr;        /* virtual address of mapping */
490     size_t pr_npage;           /* number of pages in mapping */
491     char pr_mapname[PRMAPSZ]; /* name in /proc/<pid>/object */
492     offset_t pr_offset;        /* offset into mapped object, if any */
493     int pr_mflags;             /* protection and attribute flags */
494     int pr_pagesize;           /* pagesize (bytes) for this mapping */
495     int pr_shmid;              /* SysV shmid, -1 if not SysV shared memory */
496     int pr_filler[1];         /* filler for future expansion */
497 } prsmap_t;

499 /*
500  * pr_npage bytes (plus 0-7 null bytes to round up to an 8-byte boundary)
501  * follow each mapping header, each containing zero or more of these flags.
502  */
503 #define PG_REFERENCED 0x02 /* page referenced since last read */
504 #define PG_MODIFIED   0x01 /* page modified since last read */
505 #define PG_HWMAPPED   0x04 /* page is present and mapped */

507 /*
508  * Open files.  Only in core files (for now).  Note that we'd like to use
509  * the stat or stat64 structure, but both of these structures are unfortunately
510  * not consistent between 32 and 64 bit modes.  To keep our lives simpler, we
511  * just define our own structure with types that are not sensitive to this
512  * difference.  Also, it turns out that pfiles omits a lot of info from the
513  * struct stat (e.g. times, device sizes, etc.) so we don't bother adding those
514  * here.
515  */
516 typedef struct prfdinfo {
517     int pr_fd;
518     mode_t pr_mode;

```

```

520     uid_t      pr_uid;
521     gid_t      pr_gid;

523     major_t    pr_major;    /* think stat.st_dev */
524     minor_t    pr_minor;

526     major_t    pr_rmajor;   /* think stat.st_rdev */
527     minor_t    pr_rminor;

529     ino64_t    pr_ino;
530     off64_t    pr_offset;
531     off64_t    pr_size;

533     int        pr_fileflags; /* fcntl(F_GETXFL), etc */
534     int        pr_fdflags;   /* fcntl(F_GETFD), etc. */

536     char       pr_path[MAXPATHLEN];
537 } prfdinfo_t;

539 /*
540 * Header for /proc/<pid>/lstatus /proc/<pid>/lpsinfo /proc/<pid>/lusage
541 */
542 typedef struct prheader {
543     long    pr_nent;    /* number of entries */
544     long    pr_entsize; /* size of each entry, in bytes */
545 } prheader_t;

547 /*
548 * Macros for manipulating sets of flags.
549 * sp must be a pointer to one of sigset_t, fltset_t, or sysset_t.
550 * flag must be a member of the enumeration corresponding to *sp.
551 */

553 /* turn on all flags in set */
554 #define prfillset(sp) \
555     { register int _i = sizeof (*(sp))/sizeof (uint32_t); \
556       while (_i) ((uint32_t *) (sp)) [--_i] = (uint32_t) 0xFFFFFFFF; }

558 /* turn off all flags in set */
559 #define prdelset(sp, flag) \
560     { register int _i = sizeof (*(sp))/sizeof (uint32_t); \
561       while (_i) ((uint32_t *) (sp)) [--_i] = (uint32_t) 0; }

563 /* turn on specified flag in set */
564 #define praddset(sp, flag) \
565     ((void) (((unsigned) ((flag)-1) < 32*sizeof (*(sp))/sizeof (uint32_t)) ? \
566     (((uint32_t *) (sp)) [((flag)-1)/32] |= (1U << (((flag)-1)%32))) : 0))

568 /* turn off specified flag in set */
569 #define prdelset(sp, flag) \
570     ((void) (((unsigned) ((flag)-1) < 32*sizeof (*(sp))/sizeof (uint32_t)) ? \
571     (((uint32_t *) (sp)) [((flag)-1)/32] &= ~(1U << (((flag)-1)%32))) : 0))

573 /* query: != 0 iff flag is turned on in set */
574 #define prismember(sp, flag) \
575     (((unsigned) ((flag)-1) < 32*sizeof (*(sp))/sizeof (uint32_t)) && \
576     (((uint32_t *) (sp)) [((flag)-1)/32] & (1U << (((flag)-1)%32))))

578 #if defined(_SYS_CALL32)

580 /*
581 * dev32_t version of PRNODEV
582 */
583 #define PRNODEV32 (dev32_t) (-1)

585 /*

```

```

586 * Kernel view of /proc structures for _ILP32 programs.
587 */

589 /*
590 * _ILP32 lwp status file. /proc/<pid>/lwp/<lwpid>/lwpstatus
591 */
592 typedef struct lwpstatus32 {
593     int    pr_flags;    /* flags */
594     id32_t pr_lwpid;    /* specific lwp identifier */
595     short  pr_why;      /* reason for lwp stop, if stopped */
596     short  pr_what;     /* more detailed reason */
597     short  pr_cursig;   /* current signal, if any */
598     short  pr_pad1;
599     siginfo32_t pr_info; /* info associated with signal or fault */
600     sigset_t pr_lwppend; /* set of signals pending to the lwp */
601     sigset_t pr_lwphold; /* set of signals blocked by the lwp */
602     struct sigaction32 pr_action; /* signal action for current signal */
603     stack32_t pr_altstack; /* alternate signal stack info */
604     caddr32_t pr_oldcontext; /* address of previous ucontext */
605     short  pr_syscall;   /* system call number (if in syscall) */
606     short  pr_sysarg;    /* number of arguments to this syscall */
607     int    pr_errno;     /* errno for failed syscall, 0 if successful */
608     int32_t pr_sysarg[PRSYSARGS]; /* arguments to this syscall */
609     int32_t pr_rval1;    /* primary syscall return value */
610     int32_t pr_rval2;    /* second syscall return value, if any */
611     char   pr_clname[PRCLSZ]; /* scheduling class name */
612     timestruc32_t pr_tstamp; /* real-time time stamp of stop */
613     timestruc32_t pr_utime; /* lwp user cpu time */
614     timestruc32_t pr_stime; /* lwp system cpu time */
615     int    pr_filler[11 - 2 * sizeof (timestruc32_t) / sizeof (int)];
616     int    pr_errpriv;   /* missing privilege */
617     caddr32_t pr_ustack; /* address of stack boundary data (stack32_t) */
618     uint32_t pr_instr;   /* current instruction */
619     prgregset32_t pr_reg; /* general registers */
620     prfpregset32_t pr_fpreg; /* floating-point registers */
621 } lwpstatus32_t;

623 /*
624 * _ILP32 process status file. /proc/<pid>/status
625 */
626 typedef struct pstatus32 {
627     int    pr_flags;    /* flags */
628     int    pr_nlwp;     /* number of active lwps in the process */
629     pid32_t pr_pid;     /* process id */
630     pid32_t pr_ppid;    /* parent process id */
631     pid32_t pr_pgid;    /* process group id */
632     pid32_t pr_sid;     /* session id */
633     id32_t pr_aslwpid;  /* historical; now always zero */
634     id32_t pr_agentid;  /* lwp id of the /proc agent lwp, if any */
635     sigset_t pr_sigpend; /* set of process pending signals */
636     caddr32_t pr_brkbase; /* address of the process heap */
637     size32_t pr_brksize; /* size of the process heap, in bytes */
638     caddr32_t pr_stkbase; /* address of the process stack */
639     size32_t pr_stksize; /* size of the process stack, in bytes */
640     timestruc32_t pr_utime; /* process user cpu time */
641     timestruc32_t pr_stime; /* process system cpu time */
642     timestruc32_t pr_cutime; /* sum of children's user times */
643     timestruc32_t pr_cstime; /* sum of children's system times */
644     sigset_t pr_sigtrace; /* set of traced signals */
645     fltset_t pr_filttrace; /* set of traced faults */
646     sysset_t pr_sysentry; /* set of system calls traced on entry */
647     sysset_t pr_sysexit;  /* set of system calls traced on exit */
648     char   pr_dmodel;     /* data model of the process */
649     char   pr_pad[3];
650     id32_t pr_taskid;    /* task id */
651     id32_t pr_projid;    /* project id */

```

```

652 int pr_nzomb; /* number of zombie lwps in the process */
653 id32_t pr_zoneid; /* zone id */
654 int pr_filler[15]; /* reserved for future use */
655 lwpstatus32_t pr_lwp; /* status of the representative lwp */
656 } pstatus32_t;

658 /*
659 * _ILP32 lwp ps(1) information file. /proc/<pid>/lwp/<lwpid>/lwpsinfo
660 */
661 typedef struct lwpsinfo32 {
662     int pr_flag; /* lwp flags */
663     id32_t pr_lwpid; /* lwp id */
664     caddr32_t pr_addr; /* internal address of lwp */
665     caddr32_t pr_wchan; /* wait addr for sleeping lwp */
666     char pr_stype; /* synchronization event type */
667     char pr_state; /* numeric lwp state */
668     char pr_sname; /* printable character for pr_state */
669     char pr_nice; /* nice for cpu usage */
670     short pr_syscall; /* system call number (if in syscall) */
671     char pr_oldpri; /* pre-SVR4, low value is high priority */
672     char pr_cpu; /* pre-SVR4, cpu usage for scheduling */
673     int pr_pri; /* priority, high value is high priority */
674     /* The following percent number is a 16-bit binary */
675     /* fraction [0 .. 1] with the binary point to the */
676     /* right of the high-order bit (1.0 == 0x8000) */
677     ushort_t pr_pctcpu; /* % of recent cpu time used by this lwp */
678     ushort_t pr_pad;
679     timestruc32_t pr_start; /* lwp start time, from the epoch */
680     timestruc32_t pr_time; /* usr+sys cpu time for this lwp */
681     char pr_clname[PRCLSZ]; /* scheduling class name */
682     char pr_name[PRFNSZ]; /* name of system lwp */
683     processorid_t pr_onpro; /* processor which last ran this lwp */
684     processorid_t pr_bindpro; /* processor to which lwp is bound */
685     psetid_t pr_bindpset; /* processor set to which lwp is bound */
686     int pr_lgrp; /* lwp home lgroup */
687     int pr_filler[4]; /* reserved for future use */
688 } lwpsinfo32_t;

690 /*
691 * _ILP32 process ps(1) information file. /proc/<pid>/psinfo
692 */
693 typedef struct psinfo32 {
694     int pr_flag; /* process flags */
695     int pr_nlwp; /* number of active lwps in the process */
696     pid32_t pr_pid; /* unique process id */
697     pid32_t pr_ppid; /* process id of parent */
698     pid32_t pr_pgid; /* pid of process group leader */
699     pid32_t pr_sid; /* session id */
700     uid32_t pr_uid; /* real user id */
701     uid32_t pr_euid; /* effective user id */
702     gid32_t pr_gid; /* real group id */
703     gid32_t pr_egid; /* effective group id */
704     caddr32_t pr_addr; /* address of process */
705     size32_t pr_size; /* size of process image in Kbytes */
706     size32_t pr_rsize; /* resident set size in Kbytes */
707     size32_t pr_pad1;
708     dev32_t pr_ttydev; /* controlling tty device (or PRNODEV) */
709     ushort_t pr_pctcpu; /* % of recent cpu time used by all lwps */
710     ushort_t pr_pctmem; /* % of system memory used by process */
711     timestruc32_t pr_start; /* process start time, from the epoch */
712     timestruc32_t pr_time; /* usr+sys cpu time for this process */
713     timestruc32_t pr_ctime; /* usr+sys cpu time for reaped children */
714     char pr_fname[PRFNSZ]; /* name of execed file */
715     char pr_psargs[PRARGSZ]; /* initial characters of arg list */
716     int pr_wstat; /* if zombie, the wait() status */
717     int pr_argc; /* initial argument count */

```

```

718 caddr32_t pr_argv; /* address of initial argument vector */
719 caddr32_t pr_envp; /* address of initial environment vector */
720 char pr_dmodel; /* data model of the process */
721 char pr_pad2[3];
722 id32_t pr_taskid; /* task id */
723 id32_t pr_projid; /* project id */
724 int pr_nzomb; /* number of zombie lwps in the process */
725 id32_t pr_poolid; /* pool id */
726 id32_t pr_zoneid; /* zone id */
727 id32_t pr_contract; /* process contract */
728 int pr_filler[1]; /* reserved for future use */
729 lwpsinfo32_t pr_lwp; /* information for representative lwp */
730 } psinfo32_t;

732 /*
733 * _ILP32 Memory-management interface. /proc/<pid>/map /proc/<pid>/rmap
734 */
735 typedef struct prmap32 {
736     caddr32_t pr_vaddr; /* virtual address of mapping */
737     size32_t pr_size; /* size of mapping in bytes */
738     char pr_mapname[64]; /* name in /proc/<pid>/object */
739     offset_t pr_offset; /* offset into mapped object, if any */
740     int pr_mflags; /* protection and attribute flags */
741     int pr_pagesize; /* pagesize (bytes) for this mapping */
742     int pr_shmid; /* SysV shmid, -1 if not SysV shared memory */
743     int pr_filler[1]; /* filler for future expansion */
744 } prmap32_t;

746 /*
747 * _ILP32 HAT memory-map interface. /proc/<pid>/xmap
748 */
749 typedef struct prxmap32 {
750     caddr32_t pr_vaddr; /* virtual address of mapping */
751     size32_t pr_size; /* size of mapping in bytes */
752     char pr_mapname[PRMAPSZ]; /* name in /proc/<pid>/object */
753     offset_t pr_offset; /* offset into mapped object, if any */
754     int pr_mflags; /* protection and attribute flags (see below) */
755     int pr_pagesize; /* pagesize (bytes) for this mapping */
756     int pr_shmid; /* SysV shmid, -1 if not SysV shared memory */
757     dev32_t pr_dev; /* st_dev from stat64() of mapped object, or PRNODEV */
758     uint64_t pr_ino; /* st_ino from stat64() of mapped object, if any */
759     uint32_t pr_rss; /* pages of resident memory */
760     uint32_t pr_anon; /* pages of resident anonymous memory */
761     uint32_t pr_locked; /* pages of locked memory */
762     uint32_t pr_pad; /* currently unused */
763     uint64_t pr_hatpagesize; /* pagesize of the hat mapping */
764     uint32_t pr_filler[6]; /* filler for future expansion */
765 } prxmap32_t;

767 /*
768 * _ILP32 Process credentials. PCSCRED and /proc/<pid>/cred
769 */
770 typedef struct prcred32 {
771     uid32_t pr_euid; /* effective user id */
772     uid32_t pr_ruid; /* real user id */
773     uid32_t pr_suid; /* saved user id (from exec) */
774     gid32_t pr_egid; /* effective group id */
775     gid32_t pr_rgid; /* real group id */
776     gid32_t pr_sgid; /* saved group id (from exec) */
777     int pr_ngroups; /* number of supplementary groups */
778     gid32_t pr_groups[1]; /* array of supplementary groups */
779 } prcred32_t;

781 /*
782 * _ILP32 Watchpoint interface. PCWATCH and /proc/<pid>/watch
783 */

```

```

784 typedef struct prwatch32 {
785     caddr32_t pr_vaddr;    /* virtual address of watched area */
786     size32_t pr_size;     /* size of watched area in bytes */
787     int pr_wflags;       /* watch type flags */
788     int pr_pad;
789 } prwatch32_t;

791 /*
792 * _ILP32 PCREAD/PCWRITE I/O interface.
793 */
794 typedef struct priovec32 {
795     caddr32_t pio_base;   /* buffer in controlling process */
796     size32_t pio_len;     /* size of read/write request */
797     off32_t pio_offset;  /* virtual address in target process */
798 } priovec32_t;

800 /*
801 * _ILP32 Resource usage. /proc/<pid>/usage /proc/<pid>/lwp/<lwpid>/lwpusage
802 */
803 typedef struct prusage32 {
804     id32_t pr_lwpid;      /* lwp id. 0: process or defunct */
805     int32_t pr_count;     /* number of contributing lwps */
806     timestruc32_t pr_tstamp; /* current time stamp */
807     timestruc32_t pr_create; /* process/lwp creation time stamp */
808     timestruc32_t pr_term;  /* process/lwp termination time stamp */
809     timestruc32_t pr_rtime; /* total lwp real (elapsed) time */
810     timestruc32_t pr_utime; /* user level cpu time */
811     timestruc32_t pr_stime; /* system call cpu time */
812     timestruc32_t pr_ttime; /* other system trap cpu time */
813     timestruc32_t pr_tftime; /* text page fault sleep time */
814     timestruc32_t pr_dftime; /* data page fault sleep time */
815     timestruc32_t pr_kftime; /* kernel page fault sleep time */
816     timestruc32_t pr_ltime; /* user lock wait sleep time */
817     timestruc32_t pr_slptime; /* all other sleep time */
818     timestruc32_t pr_wtime; /* wait-cpu (latency) time */
819     timestruc32_t pr_stoptime; /* stopped time */
820     timestruc32_t pr_filltime[6]; /* filler for future expansion */
821     uint32_t pr_minf;      /* minor page faults */
822     uint32_t pr_majf;     /* major page faults */
823     uint32_t pr_nswap;    /* swaps */
824     uint32_t pr_inblk;    /* input blocks */
825     uint32_t pr_oublk;    /* output blocks */
826     uint32_t pr_msnd;     /* messages sent */
827     uint32_t pr_mrcv;     /* messages received */
828     uint32_t pr_sigs;     /* signals received */
829     uint32_t pr_vctx;    /* voluntary context switches */
830     uint32_t pr_ictx;    /* involuntary context switches */
831     uint32_t pr_sysc;     /* system calls */
832     uint32_t pr_ioch;    /* chars read and written */
833     uint32_t pr_filler[10]; /* filler for future expansion */
834 } prusage32_t;

836 /*
837 * _ILP32 Page data file. /proc/<pid>/pagedata
838 */

840 /* _ILP32 page data file header */
841 typedef struct prpageheader32 {
842     timestruc32_t pr_tstamp; /* real time stamp */
843     int32_t pr_nmap; /* number of address space mappings */
844     int32_t pr_npage; /* total number of pages */
845 } prpageheader32_t;

847 /* _ILP32 page data mapping header */
848 typedef struct prsmap32 {
849     caddr32_t pr_vaddr; /* virtual address of mapping */

```

```

850     size32_t pr_npage; /* number of pages in mapping */
851     char pr_mapname[64]; /* name in /proc/<pid>/object */
852     offset_t pr_offset; /* offset into mapped object, if any */
853     int pr_mflags; /* protection and attribute flags */
854     int pr_pagesize; /* pagesize (bytes) for this mapping */
855     int pr_shmid; /* SysV shmid, -1 if not SysV shared memory */
856     int pr_filler[1]; /* filler for future expansion */
857 } prsmap32_t;

859 /*
860 * _ILP32 Header for /proc/<pid>/lstatus /proc/<pid>/lpsinfo /proc/<pid>/lusage
861 */
862 typedef struct prheader32 {
863     int32_t pr_nent; /* number of entries */
864     int32_t pr_entsize; /* size of each entry, in bytes */
865 } prheader32_t;

867 #endif /* _SYSCALL32 */

869 #endif /* !_KERNEL && _STRUCTURED_PROC == 0 */

871 #ifdef __cplusplus
872 }
873 #endif

875 #endif /* _SYS_PROCFS_H */

```



```

*****
4726 Thu Jun 30 22:03:45 2016
new/usr/src/uts/common/sys/prsystem.h
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */

22 /*
23  * Copyright 2009 Sun Microsystems, Inc. All rights reserved.
24  * Use is subject to license terms.
25 */

27 /*      Copyright (c) 1984, 1986, 1987, 1988, 1989 AT&T */
28 /*      All Rights Reserved      */

30 /*
31  * Copyright (c) 2013, Joyent, Inc. All rights reserved.
32 */

34 #ifndef _SYS_PRSYSTEM_H
35 #define _SYS_PRSYSTEM_H

37 #include <sys/isa_defs.h>
38 #include <sys/zone.h>

40 #ifdef __cplusplus
41 extern "C" {
42 #endif

44 #if defined(_KERNEL)

46 extern kmutex_t pr_pidlock;
47 extern kcondvar_t *pr_pid_cv;

49 struct prfpregset;
50 struct pstatus;
51 struct lwpstatus;
52 struct psinfo;
53 struct lwpsinfo;
54 struct prcred;
55 struct prpriv;
56 struct prsecflags;
57 #endif /* ! codereview */

```

```

59 struct seg;
60 struct regs;
61 struct watched_page;

63 /*
64  * These are functions in the proefs module that are
65  * called from the kernel proper and from other modules.
66 */
67 extern uint_t pr_getprot(struct seg *, int, void **,
68     caddr_t *, caddr_t *, caddr_t);
69 extern void pr_getprot_done(void **);
70 extern size_t pr_getsegsz(struct seg *, int);
71 extern int pr_isobject(struct vnode *);
72 extern int pr_isself(struct vnode *);
73 extern void prinvalidate(struct user *);
74 extern void prgetstatus(proc_t *, struct pstatus *, zone_t *);
75 extern void prgetlwpstatus(kthread_t *, struct lwpstatus *, zone_t *);
76 extern void prgetpsinfo(proc_t *, struct psinfo *);
77 extern void prgetlwpsinfo(kthread_t *, struct lwpsinfo *);
78 extern void prgetprfpregs(klwp_t *, struct prfpregset *);
79 extern void prgetprxregs(klwp_t *, caddr_t);
80 extern int prgetprxregsize(proc_t *);
81 #if defined(__lint)
82 /* Work around lint confusion between old and new prcred definitions */
83 extern void prgetcred();
84 #else
85 extern void prgetcred(proc_t *, struct prcred *);
86 #endif
87 extern void prgetpriv(proc_t *, struct prpriv *);
88 extern size_t prgetprivsize(void);
89 extern void prgetsecflags(proc_t *, struct prsecflags *);
90 #endif /* ! codereview */
91 extern int prnsegs(struct as *, int);
92 extern void prxit(proc_t *);
93 extern void prfree(proc_t *);
94 extern void prlwpexit(kthread_t *);
95 extern void prlwpfree(proc_t *, lwpent_t *);
96 extern void prxecstart(void);
97 extern void prxecend(void);
98 extern void prrelvm(void);
99 extern void prbarrier(proc_t *);
100 extern void prstop(int, int);
101 extern void prunstop(void);
102 extern void prnotify(struct vnode *);
103 extern void prstep(klwp_t *, int);
104 extern void prnostep(klwp_t *);
105 extern void prdostep(void);
106 extern int prundostep(void);
107 extern int prhasfp(void);
108 extern int prhasx(proc_t *);
109 extern caddr_t prmapin(struct as *, caddr_t, int);
110 extern void prmapout(struct as *, caddr_t, caddr_t, int);
111 extern int pr_watch_emul(struct regs *, caddr_t, enum seg_rw);
112 extern void pr_free_watched_pages(proc_t *);
113 extern int pr_allstopped(proc_t *, int);
114 #if defined(__sparc)
115 struct _gwindows;
116 extern int prnwindows(klwp_t *);
117 extern void prgetwindows(klwp_t *, struct _gwindows *);
118 #if defined(__sparcv9) /* 32-bit adb macros should not see these defs */
119 extern void prgetasregs(klwp_t *, asrset_t);
120 extern void prsetasregs(klwp_t *, asrset_t);
121 #endif /* __sparcv9 */
122 #endif /* __sparc */
123 #if defined(__x86)
124 struct ssd;

```

```
125 extern int prnldt(proc_t *);
126 extern void prgetldt(proc_t *, struct ssd *);
127 #endif /* __x86 */

129 #ifdef _SYSCALL32_IMPL
130 struct prfpregset32;
131 struct pstatus32;
132 struct lwpstatus32;
133 struct psinfo32;
134 struct lwpsinfo32;
135 extern void prgetstatus32(proc_t *, struct pstatus32 *, zone_t *);
136 extern void prgetlwpstatus32(kthread_t *, struct lwpstatus32 *, zone_t *);
137 extern void prgetpsinfo32(proc_t *, struct psinfo32 *);
138 extern void prgetlwpsinfo32(kthread_t *, struct lwpsinfo32 *);
139 extern void lwpsinfo_kto32(const struct lwpsinfo *src, struct lwpsinfo32 *dest);
140 extern void psinfo_kto32(const struct psinfo *src, struct psinfo32 *dest);
141 extern void prgetprfpregs32(klwp_t *, struct prfpregset32 *);
142 #if defined(__sparc)
143 struct gwindows32;
144 void prgetwindows32(klwp_t *, struct gwindows32 *);
145 #endif /* __sparc */
146 #endif /* _SYSCALL32_IMPL */

148 #endif /* defined (_KERNEL) */

150 #ifdef __cplusplus
151 }
152 #endif

154 #endif /* _SYS_PRSYSTEM_H */
```

```

*****
3060 Thu Jun 30 22:03:46 2016
new/usr/src/uts/common/sys/secflags.h
Code review comments from jeffpc
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
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 /* Copyright 2014, Richard Lowe */

14 #ifndef _SYS_SECFLAGS_H
15 #define _SYS_SECFLAGS_H

17 #ifdef __cplusplus
18 extern "C" {
19 #endif

21 #include <sys/types.h>
22 #include <sys/procset.h>

24 struct proc;
25 typedef uint64_t secflagset_t;

27 typedef struct psecflags {
28     secflagset_t psf_effective;
29     secflagset_t psf_inherit;
30     secflagset_t psf_lower;
31     secflagset_t psf_upper;
32 } psecflags_t;

34 typedef struct secflagdelta {
35     secflagset_t psd_add;           /* Flags to add */
36     secflagset_t psd_rem;          /* Flags to remove */
37     secflagset_t psd_assign;       /* Flags to assign */
38     boolean_t psd_ass_active;      /* Need to assign */
39 } secflagdelta_t;

41 typedef enum {
42     PSF_EFFECTIVE = 0,
43     PSF_INHERIT,
44     PSF_LOWER,
45     PSF_UPPER
46 } psecflagwhich_t;

49 /*
50  * p_secflags codes
51  *
52  * These flags indicate the extra security-related features enabled for a
53  * given process.
54  */
55 typedef enum {
56     PROC_SEC_ASLR = 0,
57     PROC_SEC_FORBIDNULLMAP,

```

```

58     PROC_SEC_NOEXECSTACK
59 } secflag_t;

61 extern secflagset_t secflag_to_bit(secflag_t);
62 extern boolean_t secflag_isset(secflagset_t, secflag_t);
63 extern void secflag_clear(secflagset_t *, secflag_t);
64 extern void secflag_set(secflagset_t *, secflag_t);
65 extern boolean_t secflags_isempty(secflagset_t);
66 extern void secflags_zero(secflagset_t *);
67 extern void secflags_fullset(secflagset_t *);
68 extern void secflags_copy(secflagset_t *, const secflagset_t *);
69 extern boolean_t secflags_issubset(secflagset_t, secflagset_t);
70 extern boolean_t secflags_issuperset(secflagset_t, secflagset_t);
71 extern boolean_t secflags_intersection(secflagset_t, secflagset_t);
72 extern void secflags_union(secflagset_t *, const secflagset_t *);
73 extern void secflags_difference(secflagset_t *, const secflagset_t *);
74 extern boolean_t psecflags_validate_delta(const psecflags_t *,
75     const secflagdelta_t *);
76 extern boolean_t psecflags_validate(const psecflags_t *);
77 extern void psecflags_default(psecflags_t *sf);
78 extern const char *secflag_to_str(secflag_t);
79 extern boolean_t secflag_by_name(const char *, secflag_t *);
80 extern void secflags_to_str(secflagset_t, char *, size_t);

82 /* All valid bits */
83 #define PROC_SEC_MASK (secflag_to_bit(PROC_SEC_ASLR) | \
84     secflag_to_bit(PROC_SEC_FORBIDNULLMAP) | \
85     secflag_to_bit(PROC_SEC_NOEXECSTACK))

87 #if !defined(_KERNEL)
88 extern int secflags_parse(const secflagset_t *, const char *, secflagdelta_t *);
89 extern int psecflags(idtype_t, id_t, psecflagwhich_t, secflagdelta_t *);
90 #endif

92 #if defined(_KERNEL)
93 extern boolean_t secflag_enabled(struct proc *, secflag_t);
94 extern void secflags_promote(struct proc *);
95 extern void secflags_apply_delta(secflagset_t *, const secflagdelta_t *);
96 #endif

98 #ifdef __cplusplus
99 }
100 #endif

102 #endif /* _SYS_SECFLAGS_H */
103 #endif /* ! codereview */

```

```

*****
13687 Thu Jun 30 22:03:46 2016
new/usr/src/uts/common/sys/syscall.h
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */

22 /*
23  * Copyright 2014 Garrett D'Amore <garrett@damore.org>
24  * Copyright (c) 1989, 2010, Oracle and/or its affiliates. All rights reserved.
25  * Copyright (c) 2013 by Delphix. All rights reserved.
26  * Copyright (c) 2015, Joyent, Inc. All rights reserved.
27 */

29 /*      Copyright (c) 1984, 1986, 1987, 1988, 1989 AT&T */
30 /*      All Rights Reserved      */

32 #ifndef _SYS_SYSCALL_H
33 #define _SYS_SYSCALL_H

35 #ifdef __cplusplus
36 extern "C" {
37 #endif

39 /*
40  *      system call numbers
41  *      syscall(SYS_xxxx, ...)
42  */

44 /* syscall enumeration MUST begin with 1 */

46 /*
47  * SunOS/SPARC uses 0 for the indirect system call SYS_syscall
48  * but this doesn't count because it is just another way
49  * to specify the real system call number.
50  */

52 #define SYS_syscall    0
53 #define SYS_exit      1
54 #define SYS_pseclags  2
55 #endif /* ! codereview */
56 #define SYS_read      3
57 #define SYS_write     4
58 #define SYS_open      5

```

```

59 #define SYS_close     6
60 #define SYS_linkat   7
61 #define SYS_link     9
62 #define SYS_unlink  10
63 #define SYS_symlinkat 11
64 #define SYS_chdir   12
65 #define SYS_time    13
66 #define SYS_mknod   14
67 #define SYS_chmod   15
68 #define SYS_chown   16
69 #define SYS_brk     17
70 #define SYS_stat    18
71 #define SYS_lseek   19
72 #define SYS_getpid  20
73 #define SYS_mount   21
74 #define SYS_readlinkat 22
75 #define SYS_setuid  23
76 #define SYS_getuid  24
77 #define SYS_stime   25
78 #define SYS_pcsample 26
79 #define SYS_alarm   27
80 #define SYS_fstat   28
81 #define SYS_pause   29
82 #define SYS_stty    31
83 #define SYS_gtty    32
84 #define SYS_access  33
85 #define SYS_nice    34
86 #define SYS_statfs  35
87 #define SYS_sync    36
88 #define SYS_kill    37
89 #define SYS_fstatfs 38
90 #define SYS_pgrpsys 39
91 /*
92  * subcodes:
93  *      getpgrp()      :: syscall(39,0)
94  *      setpgrp()     :: syscall(39,1)
95  *      getsid(pid)   :: syscall(39,2,pid)
96  *      setsid()      :: syscall(39,3)
97  *      getpgid(pid)  :: syscall(39,4,pid)
98  *      setpgid(pid,pgid) :: syscall(39,5,pid,pgid)
99  */
100 #define SYS_ucopystr 40
101 #define SYS_pipe     42
102 #define SYS_times    43
103 #define SYS_profil   44
104 #define SYS_faccessat 45
105 #define SYS_setgid   46
106 #define SYS_getgid   47
107 #define SYS_mknodat  48
108 #define SYS_msgsys  49
109 /*
110  * subcodes:
111  *      msgget(...)  :: msgsys(0, ...)
112  *      msgctl(...)  :: msgsys(1, ...)
113  *      msgrcv(...)  :: msgsys(2, ...)
114  *      msgsnd(...)  :: msgsys(3, ...)
115  *      msgids(...)  :: msgsys(4, ...)
116  *      msgsnap(...) :: msgsys(5, ...)
117  *      see <sys/msg.h>
118  */
119 #define SYS_sysi86    50
120 /*
121  * subcodes:
122  *      sysi86(code, ...)
123  */
124 #define SYS_acct     51

```

```

125 #define SYS_shmsys      52
126 /*
127  * subcodes:
128  *   shmctl (...) :: shmsys(0, ...)
129  *   shmctl (...) :: shmsys(1, ...)
130  *   shmdt (...) :: shmsys(2, ...)
131  *   shmget (...) :: shmsys(3, ...)
132  *   shmids (...) :: shmsys(4, ...)
133  *   see <sys/shm.h>
134 */
135 #define SYS_semsys      53
136 /*
137  * subcodes:
138  *   semctl (...) :: semsys(0, ...)
139  *   semget (...) :: semsys(1, ...)
140  *   semop (...) :: semsys(2, ...)
141  *   semids (...) :: semsys(3, ...)
142  *   semtimedop (...) :: semsys(4, ...)
143  *   see <sys/sem.h>
144 */
145 #define SYS_ioctl      54
146 #define SYS_uadmin     55
147 #define SYS_fchownat   56
148 #define SYS_utssys     57
149 /*
150  * subcodes (third argument):
151  *   uname(obuf) (obsolete) :: syscall(57, obuf, ign, 0)
152  *   subcode 1 unused
153  *   ustat(dev, obuf) :: syscall(57, obuf, dev, 2)
154  *   fusers(path, flags, obuf) :: syscall(57, path, flags, 3, obuf)
155  *   see <sys/utssys.h>
156 */
157 #define SYS_fdsync     58
158 #define SYS_execve     59
159 #define SYS_umask      60
160 #define SYS_chroot     61
161 #define SYS_fcntl      62
162 #define SYS_ulimit     63
163 #define SYS_renameat   64
164 #define SYS_unlinkat   65
165 #define SYS_fstatat    66
166 #define SYS_fstatat64  67
167 #define SYS_openat     68
168 #define SYS_openat64   69
169 #define SYS_tasksys    70
170 /*
171  * subcodes:
172  *   settaskid (...) :: tasksys(0, ...)
173  *   gettaskid (...) :: tasksys(1, ...)
174  *   getprojid (...) :: tasksys(2, ...)
175 */
176 #define SYS_acctctl    71
177 #define SYS_exacctsys  72
178 /*
179  * subcodes:
180  *   getacct (...) :: exacct(0, ...)
181  *   putacct (...) :: exacct(1, ...)
182  *   wracct (...) :: exacct(2, ...)
183 */
184 #define SYS_getpagesizes 73
185 /*
186  * subcodes:
187  *   getpagesizes2 (...) :: getpagesizes(0, ...)
188  *   getpagesizes (...) :: getpagesizes(1, ...) legacy
189 */
190 #define SYS_rctlsys    74

```

```

191 /*
192  * subcodes:
193  *   getrctl (...) :: rctlsys(0, ...)
194  *   setrctl (...) :: rctlsys(1, ...)
195  *   rctllist (...) :: rctlsys(2, ...)
196  *   rctlctl (...) :: rctlsys(3, ...)
197 */
198 #define SYS_sidsys     75
199 /*
200  * subcodes:
201  *   allocids (...) :: sidsys(0, ...)
202  *   idmap_reg (...) :: sidsys(1, ...)
203  *   idmap_unreg (...) :: sidsys(2, ...)
204 */
205 #define SYS_lwp_park   77
206 /*
207  * subcodes:
208  *   _lwp_park(timespec_t *, lwpid_t) :: syslwp_park(0, ...)
209  *   _lwp_unpark(lwpid_t, int) :: syslwp_park(1, ...)
210  *   _lwp_unpark_all(lwpid_t *, int) :: syslwp_park(2, ...)
211  *   _lwp_unpark_cancel(lwpid_t *, int) :: syslwp_park(3, ...)
212  *   _lwp_set_park(lwpid_t *, int) :: syslwp_park(4, ...)
213 */
214 #define SYS_sendfilev 78
215 /*
216  * subcodes :
217  *   sendfilev() :: sendfilev(0, ...)
218  *   sendfilev64() :: sendfilev(1, ...)
219 */
220 #define SYS_rmdir     79
221 #define SYS_mkdir     80
222 #define SYS_getdents  81
223 #define SYS_privsys   82
224 /*
225  * subcodes:
226  *   setppriv (...) :: privsys(0, ...)
227  *   getppriv (...) :: privsys(1, ...)
228  *   getimplinfo (...) :: privsys(2, ...)
229  *   setpflags (...) :: privsys(3, ...)
230  *   getpflags (...) :: privsys(4, ...)
231  *   issetugid(); :: privsys(5)
232 */
233 #define SYS_ucredsys  83
234 /*
235  * subcodes:
236  *   ucred_get (...) :: ucredsys(0, ...)
237  *   getpeerucred (...) :: ucredsys(1, ...)
238 */
239 #define SYS_sysfs     84
240 /*
241  * subcodes:
242  *   sysfs(code, ...)
243  *   see <sys/fstyp.h>
244 */
245 #define SYS_getmsg     85
246 #define SYS_putmsg     86
247 #define SYS_lstat      88
248 #define SYS_symlink    89
249 #define SYS_readlink   90
250 #define SYS_setgroups  91
251 #define SYS_getgroups  92
252 #define SYS_fchmod     93
253 #define SYS_fchown     94
254 #define SYS_sigprocmask 95
255 #define SYS_sigsuspend 96
256 #define SYS_sigaltstack 97

```

```

257 #define SYS_sigaction 98
258 #define SYS_sigpending 99
259 /*
260  * subcodes:
261  *
262  *     sigpending(...) :: syscall(99, 1, ...)
263  *     sigfillset(...) :: syscall(99, 2, ...)
264  */
265 #define SYS_context 100
266 /*
267  * subcodes:
268  *     getcontext(...) :: syscall(100, 0, ...)
269  *     setcontext(...) :: syscall(100, 1, ...)
270  */
271 #define SYS_fchmodat 101
272 #define SYS_mkdirat 102
273 #define SYS_statvfs 103
274 #define SYS_fstatvfs 104
275 #define SYS_getloadavg 105
276 #define SYS_nfssys 106
277 #define SYS_waitid 107
278 #define SYS_waitsys SYS_waitid /* historical */
279 #define SYS_sigsendsys 108
280 #define SYS_hrtsys 109
281 #define SYS_utimesys 110
282 /*
283  * subcodes:
284  *     futimens(...) :: syscall(110, 0, ...)
285  *     utimensat(...) :: syscall(110, 1, ...)
286  */
287 #define SYS_sigresend 111
288 #define SYS_priocntlsys 112
289 #define SYS_pathconf 113
290 #define SYS_mincore 114
291 #define SYS_mmap 115
292 #define SYS_mprotect 116
293 #define SYS_munmap 117
294 #define SYS_fpathconf 118
295 #define SYS_vfork 119
296 #define SYS_fchdir 120
297 #define SYS_readv 121
298 #define SYS_writev 122
299 #define SYS_preadv 123
300 #define SYS_pwritev 124
301 #define SYS_getrandom 126
302 #define SYS_mmapobj 127
303 #define SYS_setrlimit 128
304 #define SYS_getrlimit 129
305 #define SYS_lchown 130
306 #define SYS_memcntl 131
307 #define SYS_getpmsg 132
308 #define SYS_putpmsg 133
309 #define SYS_rename 134
310 #define SYS_uname 135
311 #define SYS_setegid 136
312 #define SYS_sysconfig 137
313 #define SYS_adjtime 138
314 #define SYS_systeminfo 139
315 #define SYS_sharefs 140
316 #define SYS_seteuid 141
317 #define SYS_forksys 142
318 /*
319  * subcodes:
320  *     forkx(flags) :: forksys(0, flags)
321  *     forkallx(flags) :: forksys(1, flags)
322  *     vforkx(flags) :: forksys(2, flags)

```

```

323  */
324 #define SYS_sigtimedwait 144
325 #define SYS_lwp_info 145
326 #define SYS_yield 146
327 #define SYS_lwp_sema_post 148
328 #define SYS_lwp_sema_trywait 149
329 #define SYS_lwp_detach 150
330 #define SYS_corectl 151
331 #define SYS_modctl 152
332 #define SYS_fchroot 153
333 #define SYS_vhangup 155
334 #define SYS_gettimeofday 156
335 #define SYS_getitimer 157
336 #define SYS_setitimer 158
337 #define SYS_lwp_create 159
338 #define SYS_lwp_exit 160
339 #define SYS_lwp_suspend 161
340 #define SYS_lwp_continue 162
341 #define SYS_lwp_kill 163
342 #define SYS_lwp_self 164
343 #define SYS_lwp_sigmask 165
344 #define SYS_lwp_private 166
345 #define SYS_lwp_wait 167
346 #define SYS_lwp_mutex_wakeup 168
347 #define SYS_lwp_cond_wait 170
348 #define SYS_lwp_cond_signal 171
349 #define SYS_lwp_cond_broadcast 172
350 #define SYS_pread 173
351 #define SYS_pwrite 174
352 #define SYS_llseek 175
353 #define SYS_inst_sync 176
354 #define SYS_brand 177
355 #define SYS_kaio 178
356 /*
357  * subcodes:
358  *     aioread(...) :: kaio(AIOREAD, ...)
359  *     aiowrite(...) :: kaio(AIOWRITE, ...)
360  *     aiowait(...) :: kaio(AIOWAIT, ...)
361  *     aiocancel(...) :: kaio(AIOCANCEL, ...)
362  *     aionotify() :: kaio(AIONOTIFY)
363  *     aioinit() :: kaio(AIOINIT)
364  *     aiostart() :: kaio(AIOSTART)
365  *     see <sys/aio.h>
366  */
367 #define SYS_cpc 179
368 #define SYS_lgrpsys 180
369 #define SYS_meminfosys SYS_lgrpsys
370 /*
371  * subcodes:
372  *     meminfo(...) :: meminfosys(MISYS_MEMINFO, ...)
373  */
374 #define SYS_rusagesys 181
375 /*
376  * subcodes:
377  *     getrusage(...) :: rusagesys(RUSAGESYS_GETRUSAGE, ...)
378  *     getvmusage(...) :: rusagesys(RUSAGESYS_GETVMUSAGE, ...)
379  */
380 #define SYS_port 182
381 /*
382  * subcodes:
383  *     port_create(...) :: portfs(PORT_CREATE, ...)
384  *     port_associate(...) :: portfs(PORT_ASSOCIATE, ...)
385  *     port_dissociate(...) :: portfs(PORT DISSOCIATE, ...)
386  *     port_send(...) :: portfs(PORT_SEND, ...)
387  *     port_sendn(...) :: portfs(PORT_SENDR, ...)
388  *     port_get(...) :: portfs(PORT_GET, ...)

```

```

389 *      port_getn(...) :: portfs(PORT_GETN, ...)
390 *      port_alert(...) :: portfs(PORT_ALERT, ...)
391 *      port_dispatch(...) :: portfs(PORT_DISPATCH, ...)
392 */
393 #define SYS_pollsys      183
394 #define SYS_labelsys    184
395 #define SYS_acl          185
396 #define SYS_auditsys    186
397 #define SYS_processor_bind 187
398 #define SYS_processor_info 188
399 #define SYS_p_online    189
400 #define SYS_sigqueue    190
401 #define SYS_clock_gettime 191
402 #define SYS_clock_settime 192
403 #define SYS_clock_getres 193
404 #define SYS_timer_create 194
405 #define SYS_timer_delete 195
406 #define SYS_timer_settime 196
407 #define SYS_timer_gettime 197
408 #define SYS_timer_getoverrun 198
409 #define SYS_nanosleep 199
410 #define SYS_facl        200
411 #define SYS_door        201
412 /*
413  * Door Subcodes:
414  *      0      door_create
415  *      1      door_revoke
416  *      2      door_info
417  *      3      door_call
418  *      4      door_return
419  */
420 #define SYS_setreuid     202
421 #define SYS_setregid     203
422 #define SYS_install_ustrap 204
423 #define SYS_signotify    205
424 #define SYS_schedctl    206
425 #define SYS_pset        207
426 #define SYS_sparc_ustrap_install 208
427 #define SYS_resolvepath 209
428 #define SYS_lwp_mutex_timedlock 210
429 #define SYS_lwp_sema_timedwait 211
430 #define SYS_lwp_rwlock_sys 212
431 /*
432  * subcodes:
433  *      lwp_rwlock_rdlock(...) :: syscall(212, 0, ...)
434  *      lwp_rwlock_wrlock(...) :: syscall(212, 1, ...)
435  *      lwp_rwlock_tryrdlock(...) :: syscall(212, 2, ...)
436  *      lwp_rwlock_trywrlock(...) :: syscall(212, 3, ...)
437  *      lwp_rwlock_unlock(...) :: syscall(212, 4, ...)
438  */
439 /* system calls for large file (> 2 gigabyte) support */
440 #define SYS_getdents64   213
441 #define SYS_mmap64      214
442 #define SYS_stat64      215
443 #define SYS_lstat64     216
444 #define SYS_fstat64     217
445 #define SYS_statvfs64   218
446 #define SYS_fstatvfs64  219
447 #define SYS_setrlimit64 220
448 #define SYS_getrlimit64 221
449 #define SYS_pread64     222
450 #define SYS_pwrite64    223
451 #define SYS_open64      225
452 #define SYS_rpcsys      226
453 #define SYS_zone        227
454 /*

```

```

455 * subcodes:
456 *      zone_create(...) :: zone(ZONE_CREATE, ...)
457 *      zone_destroy(...) :: zone(ZONE_DESTROY, ...)
458 *      zone_getattr(...) :: zone(ZONE_GETATTR, ...)
459 *      zone_enter(...) :: zone(ZONE_ENTER, ...)
460 *      zone_list(...) :: zone(ZONE_LIST, ...)
461 *      zone_shutdown(...) :: zone(ZONE_SHUTDOWN, ...)
462 *      zone_lookup(...) :: zone(ZONE_LOOKUP, ...)
463 *      zone_boot(...) :: zone(ZONE_BOOT, ...)
464 *      zone_version(...) :: zone(ZONE_VERSION, ...)
465 *      zone_setattr(...) :: zone(ZONE_SETATTR, ...)
466 *      zone_add_datalink(...) :: zone(ZONE_ADD_DATA_LINK, ...)
467 *      zone_remove_datalink(...) :: zone(ZONE_DEL_DATA_LINK, ...)
468 *      zone_check_datalink(...) :: zone(ZONE_CHECK_DATA_LINK, ...)
469 *      zone_list_datalink(...) :: zone(ZONE_LIST_DATA_LINK, ...)
470 */
471 #define SYS_autofssys    228
472 #define SYS_getcwd      229
473 #define SYS_so_socket    230
474 #define SYS_so_socketpair 231
475 #define SYS_bind        232
476 #define SYS_listen      233
477 #define SYS_accept      234
478 #define SYS_connect     235
479 #define SYS_shutdown    236
480 #define SYS_recv        237
481 #define SYS_recvfrom    238
482 #define SYS_recvmsg     239
483 #define SYS_send        240
484 #define SYS_sendmsg     241
485 #define SYS_sendto     242
486 #define SYS_getpeername 243
487 #define SYS_getsockname 244
488 #define SYS_getsockopt  245
489 #define SYS_setsockopt  246
490 #define SYS_sockconfig  247
491 /*
492  * NTP codes
493  */
494 #define SYS_ntp_gettime  248
495 #define SYS_ntp_adjtime  249
496 #define SYS_lwp_mutex_unlock 250
497 #define SYS_lwp_mutex_trylock 251
498 #define SYS_lwp_mutex_register 252
499 #define SYS_cladm        253
500 #define SYS_uucopy       254
501 #define SYS_umount2     255
502
503 #ifndef _ASM
504
505 typedef struct { /* syscall set type */
506     unsigned int    word[16];
507 } sysset_t;
508
509 typedef struct { /* return values from system call */
510     long    sys_rval1; /* primary return value from system call */
511     long    sys_rval2; /* second return value from system call */
512 } sysret_t;
513
514 #if !defined(_KERNEL)
515
516 extern long    syscall(int, ...);
517 extern long    __syscall(sysret_t *, int, ...);
518 extern int     __set_errno(int);
519
520 #endif /* _KERNEL */

```

```
522 #endif /* _ASM */
524 #ifdef __cplusplus
525 }
526 #endif
528 #endif /* _SYS_SYSCALL_H */
```



```

*****
3459 Thu Jun 30 22:03:47 2016
new/usr/src/uts/common/sys/tsol/priv.h
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
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 */

26 #ifndef _SYS_TSOL_PRIV_H
27 #define _SYS_TSOL_PRIV_H

29 #pragma ident      "%Z%M% %I%      %E% SMI"

29 #include <sys/priv.h>

31 #ifdef __cplusplus
32 extern "C" {
33 #endif

35 typedef enum priv_ftype {
36     PRIV_ALLOWED,
37     PRIV_FORCED
38 } priv_ftype_t;

40 /*
41  * Privilege macros.
42  *
43  * These names are here for compatibility reasons, and thus do not match
44  * priv_impl.h
45  *#endif /* ! codereview */
46  */

48 /*
49  * PRIV_ASSERT(a, b) setst.privilege "b" in privilege set "a".
50  */
51 #define PRIV_ASSERT(a, b) (priv_addset(a, b))

53 /*
54  * PRIV_CLEAR(a,b) clearst.privilege "b" in privilege set "a".
55  */
56 #define PRIV_CLEAR(a, b) (priv_delset(a, b))

```

```

58 /*
59  * PRIV_EQUAL(set_a, set_b) is true if set_a and set_b are identical.
60  */
61 #define PRIV_EQUAL(a, b) (priv_isequalset(a, b))
62 #define PRIV_EMPTY(a) (priv_emptyset(a))
63 #define PRIV_FILL(a) (priv_fillset(a))

65 /*
66  * PRIV_ISASSERT tests if privilege 'b' is asserted in privilege set 'a'.
67  */
68 #define PRIV_ISASSERT(a, b) (priv_ismember(a, b))
69 #define PRIV_ISEMPY(a) (priv_isemptyset(a))
70 #define PRIV_ISFULL(a) (priv_isfullset(a))

72 /*
73  * This macro returns 1 if all privileges asserted in privilege set "a"
74  * are also asserted in privilege set "b" (i.e. if a is a subset of b)
75  */
76 #define PRIV_ISSUBSET(a, b) (priv_issubset(a, b))

78 /*
79  * Takes intersection of "a" and "b" and stores in "b".
80  */
81 #define PRIV_INTERSECT(a, b) (priv_intersect(a, b))

83 /*
84  * Replaces "a" with inverse of "a".
85  */
86 #define PRIV_INVERSE(a) (priv_inverse(a))

88 /*
89  * Takes union of "a" and "b" and stores in "b".
90  */
91 #define PRIV_UNION(a, b) (priv_union(a, b))

94 #define PRIV_FILE_UPGRADE_SL ((const char *)"file_upgrade_sl")
95 #define PRIV_FILE_DOWNGRADE_SL ((const char *)"file_downgrade_sl")
96 #
97 #define PRIV_PROC_AUDIT_TCB ((const char *)"proc_audit")
98 #define PRIV_PROC_AUDIT_APPL ((const char *)"proc_audit")
99 #
100 #define PRIV_SYS_TRANS_LABEL ((const char *)"sys_trans_label")
101 #define PRIV_WIN_COLORMAP ((const char *)"win_colormap")
102 #define PRIV_WIN_CONFIG ((const char *)"win_config")
103 #define PRIV_WIN_DAC_READ ((const char *)"win_dac_read")
104 #define PRIV_WIN_DAC_WRITE ((const char *)"win_dac_write")
105 #define PRIV_WIN_DGA ((const char *)"win_dga")
106 #define PRIV_WIN_DEVICES ((const char *)"win_devices")
107 #define PRIV_WIN_DOWNGRADE_SL ((const char *)"win_downgrade_sl")
108 #define PRIV_WIN_FONTPATH ((const char *)"win_fontpath")
109 #define PRIV_WIN_MAC_READ ((const char *)"win_mac_read")
110 #define PRIV_WIN_MAC_WRITE ((const char *)"win_mac_write")
111 #define PRIV_WIN_SELECTION ((const char *)"win_selection")
112 #define PRIV_WIN_UPGRADE_SL ((const char *)"win_upgrade_sl")

114 #ifdef __cplusplus
115 }
116 #endif

118 #endif /* _SYS_TSOL_PRIV_H */

```

```

*****
27284 Thu Jun 30 22:03:48 2016
new/usr/src/uts/common/sys/zone.h
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
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 (c) 2003, 2010, Oracle and/or its affiliates. All rights reserved.
23  * Copyright 2015 Joyent, Inc. All rights reserved.
24  * Copyright 2014 Nexenta Systems, Inc. All rights reserved.
25  * Copyright 2014 Igor Kozhukhov <ikozhukhov@gmail.com>.
26  */

28 #ifndef _SYS_ZONE_H
29 #define _SYS_ZONE_H

31 #include <sys/types.h>
32 #include <sys/mutex.h>
33 #include <sys/param.h>
34 #include <sys/rctl.h>
35 #include <sys/ipc_rctl.h>
36 #include <sys/pset.h>
37 #include <sys/tsol/label.h>
38 #include <sys/cred.h>
39 #include <sys/netstack.h>
40 #include <sys/uadmin.h>
41 #include <sys/ksynch.h>
42 #include <sys/socket_impl.h>
43 #include <sys/secflags.h>
44 #endif /* !codereview */
45 #include <netinet/in.h>

47 #ifdef __cplusplus
48 extern "C" {
49 #endif

51 /*
52  * NOTE
53  *
54  * The contents of this file are private to the implementation of
55  * Solaris and are subject to change at any time without notice.
56  * Applications and drivers using these interfaces may fail to
57  * run on future releases.
58  */

```

```

60 /* Available both in kernel and for user space */

62 /* zone id restrictions and special ids */
63 #define MAX_ZONEID 9999
64 #define MIN_USERZONEID 1 /* lowest user-creatable zone ID */
65 #define MIN_ZONEID 0 /* minimum zone ID on system */
66 #define GLOBAL_ZONEID 0
67 #define ZONEID_WIDTH 4 /* for printf */

69 /*
70  * Special zoneid_t token to refer to all zones.
71  */
72 #define ALL_ZONES (-1)

74 /* system call subcodes */
75 #define ZONE_CREATE 0
76 #define ZONE_DESTROY 1
77 #define ZONE_GETATTR 2
78 #define ZONE_ENTER 3
79 #define ZONE_LIST 4
80 #define ZONE_SHUTDOWN 5
81 #define ZONE_LOOKUP 6
82 #define ZONE_BOOT 7
83 #define ZONE_VERSION 8
84 #define ZONE_SETATTR 9
85 #define ZONE_ADD_DATALINK 10
86 #define ZONE_DEL_DATALINK 11
87 #define ZONE_CHECK_DATALINK 12
88 #define ZONE_LIST_DATALINK 13

90 /* zone attributes */
91 #define ZONE_ATTR_ROOT 1
92 #define ZONE_ATTR_NAME 2
93 #define ZONE_ATTR_STATUS 3
94 #define ZONE_ATTR_PRIVSET 4
95 #define ZONE_ATTR_UNIQID 5
96 #define ZONE_ATTR_POOLID 6
97 #define ZONE_ATTR_INITPID 7
98 #define ZONE_ATTR_SLBL 8
99 #define ZONE_ATTR_INITNAME 9
100 #define ZONE_ATTR_BOOTARGS 10
101 #define ZONE_ATTR_BRAND 11
102 #define ZONE_ATTR_PHYS_MCAP 12
103 #define ZONE_ATTR_SCHED_CLASS 13
104 #define ZONE_ATTR_FLAGS 14
105 #define ZONE_ATTR_HOSTID 15
106 #define ZONE_ATTR_FS_ALLOWED 16
107 #define ZONE_ATTR_NETWORK 17
108 #define ZONE_ATTR_INITNORESTART 20
109 #define ZONE_ATTR_SECFLAGS 21
110 #endif /* !codereview */

112 /* Start of the brand-specific attribute namespace */
113 #define ZONE_ATTR_BRAND_ATTRS 32768

115 #define ZONE_FS_ALLOWED_MAX 1024

117 #define ZONE_EVENT_CHANNEL "com.sun:zones:status"
118 #define ZONE_EVENT_STATUS_CLASS "status"
119 #define ZONE_EVENT_STATUS_SUBCLASS "change"

121 #define ZONE_EVENT_UNINITIALIZED "uninitialized"
122 #define ZONE_EVENT_INITIALIZED "initialized"
123 #define ZONE_EVENT_READY "ready"
124 #define ZONE_EVENT_RUNNING "running"

```

```

125 #define ZONE_EVENT_SHUTTING_DOWN      "shutting_down"

127 #define ZONE_CB_NAME                   "zonename"
128 #define ZONE_CB_NEWSTATE               "newstate"
129 #define ZONE_CB_OLDSTATE               "oldstate"
130 #define ZONE_CB_TIMESTAMP              "when"
131 #define ZONE_CB_ZONEID                 "zoneid"

133 /*
134 * Exit values that may be returned by scripts or programs invoked by various
135 * zone commands.
136 *
137 * These are defined as:
138 *
139 *     ZONE_SUBPROC_OK
140 *     =====
141 *     The subprocess completed successfully.
142 *
143 *     ZONE_SUBPROC_USAGE
144 *     =====
145 *     The subprocess failed with a usage message, or a usage message should
146 *     be output in its behalf.
147 *
148 *     ZONE_SUBPROC_NOTCOMPLETE
149 *     =====
150 *     The subprocess did not complete, but the actions performed by the
151 *     subprocess require no recovery actions by the user.
152 *
153 *     For example, if the subprocess were called by "zoneadm install," the
154 *     installation of the zone did not succeed but the user need not perform
155 *     a "zoneadm uninstall" before attempting another install.
156 *
157 *     ZONE_SUBPROC_FATAL
158 *     =====
159 *     The subprocess failed in a fatal manner, usually one that will require
160 *     some type of recovery action by the user.
161 *
162 *     For example, if the subprocess were called by "zoneadm install," the
163 *     installation of the zone did not succeed and the user will need to
164 *     perform a "zoneadm uninstall" before another install attempt is
165 *     possible.
166 *
167 *     The non-success exit values are large to avoid accidental collision
168 *     with values used internally by some commands (e.g. "Z_ERR" and
169 *     "Z_USAGE" as used by zoneadm.)
170 */
171 #define ZONE_SUBPROC_OK                  0
172 #define ZONE_SUBPROC_USAGE               253
173 #define ZONE_SUBPROC_NOTCOMPLETE        254
174 #define ZONE_SUBPROC_FATAL              255

176 #ifndef _SYSCALL32
177 typedef struct {
178     caddr32_t zone_name;
179     caddr32_t zone_root;
180     caddr32_t zone_privs;
181     size32_t zone_privssz;
182     caddr32_t rctlbuf;
183     size32_t rctlbufsz;
184     caddr32_t extended_error;
185     caddr32_t zfsbuf;
186     size32_t zfsbufsz;
187     int match;
188     uint32_t doi;
189     caddr32_t label;
190     int flags;

```

```

191 } zone_def32;
192 #endif
193 typedef struct {
194     const char *zone_name;
195     const char *zone_root;
196     const struct priv_set *zone_privs;
197     size_t zone_privssz;
198     const char *rctlbuf;
199     size_t rctlbufsz;
200     int *extended_error;
201     const char *zfsbuf;
202     size_t zfsbufsz;
203     int match;
204     uint32_t doi;
205     const bslabel_t *label;
206     int flags;
207 } zone_def;

209 /* extended error information */
210 #define ZE_UNKNOWN      0 /* No extended error info */
211 #define ZE_CHROOTED    1 /* tried to zone_create from chroot */
212 #define ZE_AREMOUNTS   2 /* there are mounts within the zone */
213 #define ZE_LABELINUSE  3 /* label is already in use by some other zone */

215 /*
216 * zone_status values
217 *
218 * You must modify zone_status_names in mdb(1M)'s genunix module
219 * (genunix/zone.c) when you modify this enum.
220 */
221 typedef enum {
222     ZONE_IS_UNINITIALIZED = 0,
223     ZONE_IS_INITIALIZED,
224     ZONE_IS_READY,
225     ZONE_IS_BOOTING,
226     ZONE_IS_RUNNING,
227     ZONE_IS_SHUTTING_DOWN,
228     ZONE_IS_EMPTY,
229     ZONE_IS_DOWN,
230     ZONE_IS_DYING,
231     ZONE_IS_DEAD
232 } zone_status_t;
233 #define ZONE_MIN_STATE      ZONE_IS_UNINITIALIZED
234 #define ZONE_MAX_STATE     ZONE_IS_DEAD

236 /*
237 * Valid commands which may be issued by zoneadm to zoneadm. The kernel also
238 * communicates with zoneadm, but only uses Z_REBOOT and Z_HALT.
239 */
240 typedef enum zone_cmd {
241     Z_READY, Z_BOOT, Z_FORCEBOOT, Z_REBOOT, Z_HALT, Z_NOTE_UNINSTALLING,
242     Z_MOUNT, Z_FORCEMOUNT, Z_UNMOUNT, Z_SHUTDOWN
243 } zone_cmd_t;

245 /*
246 * The structure of a request to zoneadm.
247 */
248 typedef struct zone_cmd_arg {
249     uint64_t    uniqid; /* unique "generation number" */
250     zone_cmd_t cmd;    /* requested action */
251     uint32_t    _pad;  /* need consistent 32/64 bit alignmt */
252     char locale[MAXPATHLEN]; /* locale in which to render messages */
253     char bootbuf[BOOTARGS_MAX]; /* arguments passed to zone_boot() */
254 } zone_cmd_arg_t;

256 /*

```

```

257 * Structure of zoneadm's response to a request. A NULL return value means
258 * the caller should attempt to restart zoneadm and retry.
259 */
260 typedef struct zone_cmd_rval {
261     int rval; /* return value of request */
262     char errbuf[1]; /* variable-sized buffer containing error messages */
263 } zone_cmd_rval_t;

265 /*
266 * The zone support infrastructure uses the zone name as a component
267 * of unix domain (AF_UNIX) sockets, which are limited to 108 characters
268 * in length, so ZONENAME_MAX is limited by that.
269 */
270 #define ZONENAME_MAX 64

272 #define GLOBAL_ZONENAME "global"

274 /*
275 * Extended Regular expression (see regex(5)) which matches all valid zone
276 * names.
277 */
278 #define ZONENAME_REGEX "[a-zA-Z0-9][_\.a-zA-Z0-9]{0,62}"

280 /*
281 * Where the zones support infrastructure places temporary files.
282 */
283 #define ZONES_TMPDIR "/var/run/zones"

285 /*
286 * The path to the door used by clients to communicate with zoneadm.
287 */
288 #define ZONE_DOOR_PATH ZONES_TMPDIR "%s.zoneadm_door"

291 /* zone_flags */
292 /*
293 * Threads that read or write the following flag must hold zone_lock.
294 */
295 #define ZF_REFCOUNTS_LOGGED 0x1 /* a thread logged the zone's refs */

297 /*
298 * The following threads are set when the zone is created and never changed.
299 * Threads that test for these flags don't have to hold zone_lock.
300 */
301 #define ZF_HASHED_LABEL 0x2 /* zone has a unique label */
302 #define ZF_IS_SCRATCH 0x4 /* scratch zone */
303 #define ZF_NET_EXCL 0x8 /* Zone has an exclusive IP stack */

306 /* zone_create flags */
307 #define ZCF_NET_EXCL 0x1 /* Create a zone with exclusive IP */

309 /* zone network properties */
310 #define ZONE_NETWORK_ADDRESS 1
311 #define ZONE_NETWORK_DEFROUTER 2

313 #define ZONE_NET_ADDRNAME "address"
314 #define ZONE_NET_RTRNAME "route"

316 typedef struct zone_net_data {
317     int zn_type;
318     int zn_len;
319     datalink_id_t zn_linkid;
320     uint8_t zn_val[1];
321 } zone_net_data_t;

```

```

324 #ifndef _KERNEL

326 /*
327 * We need to protect the definition of 'list_t' from userland applications and
328 * libraries which may be defining their own versions.
329 */
330 #include <sys/list.h>
331 #include <sys/loadavg.h>

333 #define GLOBAL_ZONEUNIQID 0 /* uniqid of the global zone */

335 struct pool;
336 struct brand;

338 /*
339 * Each of these constants identifies a kernel subsystem that acquires and
340 * releases zone references. Each subsystem that invokes
341 * zone_hold_ref() and zone_rele_ref() should specify the
342 * zone_ref_subsys_t constant associated with the subsystem. Tracked holds
343 * help users and developers quickly identify subsystems that stall zone
344 * shutdowns indefinitely.
345 *
346 * NOTE: You must modify zone_ref_subsys_names in usr/src/uts/common/os/zone.c
347 * when you modify this enumeration.
348 */
349 typedef enum zone_ref_subsys {
350     ZONE_REF_NFS, /* NFS */
351     ZONE_REF_NFSV4, /* NFSv4 */
352     ZONE_REF_SMBFS, /* SMBFS */
353     ZONE_REF_MNTFS, /* MNTFS */
354     ZONE_REF_LOFI, /* LOFI devices */
355     ZONE_REF_VFS, /* VFS infrastructure */
356     ZONE_REF_IPC, /* IPC infrastructure */
357     ZONE_REF_NUM_SUBSYS /* This must be the last entry. */
358 } zone_ref_subsys_t;

360 /*
361 * zone_ref represents a general-purpose references to a zone. Each zone's
362 * references are linked into the zone's zone_t::zone_ref_list. This allows
363 * debuggers to walk zones' references.
364 */
365 typedef struct zone_ref {
366     struct zone *zref_zone; /* the zone to which the reference refers */
367     list_node_t zref_linkage; /* linkage for zone_t::zone_ref_list */
368 } zone_ref_t;

370 /*
371 * Structure to record list of ZFS datasets exported to a zone.
372 */
373 typedef struct zone_dataset {
374     char *zd_dataset;
375     list_node_t zd_linkage;
376 } zone_dataset_t;

378 /*
379 * structure for zone kstats
380 */
381 typedef struct zone_kstat {
382     kstat_named_t zk_zonename;
383     kstat_named_t zk_usage;
384     kstat_named_t zk_value;
385 } zone_kstat_t;

387 struct cpucap;

```

```

389 typedef struct {
390     kstat_named_t    zm_zonename;
391     kstat_named_t    zm_pgpgin;
392     kstat_named_t    zm_anonpgin;
393     kstat_named_t    zm_execpgin;
394     kstat_named_t    zm_fspgin;
395     kstat_named_t    zm_anon_alloc_fail;
396 } zone_mcap_kstat_t;

398 typedef struct {
399     kstat_named_t    zm_zonename;    /* full name, kstat truncates name */
400     kstat_named_t    zm_untime;
401     kstat_named_t    zm_stime;
402     kstat_named_t    zm_wtime;
403     kstat_named_t    zm_avenrun1;
404     kstat_named_t    zm_avenrun5;
405     kstat_named_t    zm_avenrun15;
406     kstat_named_t    zm_ffcap;
407     kstat_named_t    zm_ffnproc;
408     kstat_named_t    zm_ffnomem;
409     kstat_named_t    zm_ffmisc;
410     kstat_named_t    zm_nested_intp;
411     kstat_named_t    zm_init_pid;
412     kstat_named_t    zm_boot_time;
413 } zone_misc_kstat_t;

415 typedef struct zone {
416     /*
417      * zone_name is never modified once set.
418      */
419     char            *zone_name;    /* zone's configuration name */
420     /*
421      * zone_nodename and zone_domain are never freed once allocated.
422      */
423     char            *zone_nodename; /* utsname.nodename equivalent */
424     char            *zone_domain;   /* srpc_domain equivalent */
425     /*
426      * zone_hostid is used for per-zone hostid emulation.
427      * Currently it isn't modified after it's set (so no locks protect
428      * accesses), but that might have to change when we allow
429      * administrators to change running zones' properties.
430      *
431      * The global zone's zone_hostid must always be HW_INVALID_HOSTID so
432      * that zone_get_hostid() will function correctly.
433      */
434     uint32_t        zone_hostid;    /* zone's hostid, HW_INVALID_HOSTID */
435     /* if not emulated */
436     /*
437      * zone_lock protects the following fields of a zone_t:
438      *   zone_ref
439      *   zone_cred_ref
440      *   zone_subsys_ref
441      *   zone_ref_list
442      *   zone_ntasks
443      *   zone_flags
444      *   zone_zsd
445      *   zone_pfexecd
446      */
447     kmutex_t        zone_lock;
448     /*
449      * zone_linkage is the zone's linkage into the active or
450      * death-row list. The field is protected by zonehash_lock.
451      */
452     list_node_t     zone_linkage;
453     zoneid_t        zone_id;        /* ID of zone */
454     uint_t          zone_ref;       /* count of zone_hold()s on zone */

```

```

455     uint_t          zone_cred_ref;  /* count of zone_hold_cred()s on zone */
456     /*
457      * Fixed-sized array of subsystem-specific reference counts
458      * The sum of all of the counts must be less than or equal to zone_ref.
459      * The array is indexed by the counts' subsystems' zone_ref_subsys_t
460      * constants.
461      */
462     uint_t          zone_subsys_ref[ZONE_REF_NUM_SUBSYS];
463     list_t          zone_ref_list;  /* list of zone_ref_t structs */
464     /*
465      * zone_rootvp and zone_rootpath can never be modified once set.
466      */
467     struct vnode    *zone_rootvp;  /* zone's root vnode */
468     char            *zone_rootpath; /* Path to zone's root + '/' */
469     ushort_t        zone_flags;    /* misc flags */
470     zone_status_t   zone_status;    /* protected by zone_status_lock */
471     uint_t          zone_ntasks;    /* number of tasks executing in zone */
472     kmutex_t        zone_nlwps_lock; /* protects zone_nlwps, and *nlwps */
473     /* counters in projects and tasks */
474     /* that are within the zone */
475     rctl_qty_t      zone_nlwps;     /* number of nlwps in zone */
476     rctl_qty_t      zone_nlwps_ctl; /* protected by zone_rctls->rcls_lock */
477     rctl_qty_t      zone_shmmax;    /* System V shared memory usage */
478     ipc_rqty_t      zone_ipc;       /* System V IPC id resource usage */

480     uint_t          zone_rootpathlen; /* strlen(zone_rootpath) + 1 */
481     uint32_t        zone_shares;    /* FSS shares allocated to zone */
482     rctl_set_t      *zone_rctls;    /* zone-wide (zone.*) rctls */
483     kmutex_t        zone_mem_lock;  /* protects zone_locked_mem and */
484     /* kpd_locked_mem for all */
485     /* projects in zone. */
486     /* Also protects zone_max_swap */
487     /* grab after p_lock, before rcs_lock */
488     rctl_qty_t      zone_locked_mem; /* bytes of locked memory in */
489     /* zone */
490     rctl_qty_t      zone_locked_mem_ctl; /* Current locked memory */
491     /* limit. Protected by */
492     /* zone_rctls->rcls_lock */
493     rctl_qty_t      zone_max_swap;  /* bytes of swap reserved by zone */
494     rctl_qty_t      zone_max_swap_ctl; /* current swap limit. */
495     /* Protected by */
496     /* zone_rctls->rcls_lock */
497     kmutex_t        zone_rctl_lock; /* protects zone_max_lofi */
498     rctl_qty_t      zone_max_lofi;  /* lofi devs for zone */
499     rctl_qty_t      zone_max_lofi_ctl; /* current lofi limit. */
500     /* Protected by */
501     /* zone_rctls->rcls_lock */
502     list_t          zone_zsd;        /* list of Zone-Specific Data values */
503     kcondvar_t      zone_cv;        /* used to signal state changes */
504     struct proc     *zone_zsched;    /* Dummy kernel "zsched" process */
505     pid_t          zone_proc_initpid; /* pid of "init" for this zone */
506     char            *zone_initname;  /* fs path to 'init' */
507     int             zone_boot_err;   /* for zone_boot() if boot fails */
508     char            *zone_bootargs;  /* arguments passed via zone_boot() */
509     uint64_t        zone_phys_mcap;  /* physical memory cap */
510     /*
511      * zone_kthreads is protected by zone_status_lock.
512      */
513     kthread_t       *zone_kthreads;  /* kernel threads in zone */
514     struct priv_set *zone_privset;   /* limit set for zone */
515     /*
516      * zone_vfslist is protected by vfs_list_lock().
517      */
518     struct vfs      *zone_vfslist;   /* list of FS's mounted in zone */
519     uint64_t        zone_uniqid;     /* unique zone generation number */
520     struct cred     *zone_kcred;     /* kcred-like, zone-limited cred */

```

```

521 /*
522  * zone_pool is protected by pool_lock().
523  */
524 struct pool      *zone_pool;    /* pool the zone is bound to */
525 hrtime_t        zone_pool_mod; /* last pool bind modification time */
526 /* zone_psetid is protected by cpu_lock */
527 psetid_t        zone_psetid;   /* pset the zone is bound to */

529 time_t          zone_boot_time; /* Similar to boot_time */

531 /*
532  * The following two can be read without holding any locks. They are
533  * updated under cpu_lock.
534  */
535 int             zone_ncpus;     /* zone's idea of ncpus */
536 int             zone_ncpus_online; /* zone's idea of ncpus_online */
537 /*
538  * List of ZFS datasets exported to this zone.
539  */
540 list_t          zone_datasets; /* list of datasets */

542 ts_label_t      *zone_slabel;  /* zone sensitivity label */
543 int             zone_match;     /* require label match for packets */
544 tsol_mlp_list_t zone_mlps;     /* MLPs on zone-private addresses */

546 boolean_t       zone_restart_init; /* Restart init if it dies? */
547 struct brand    *zone_brand;     /* zone's brand */
548 void            *zone_brand_data; /* store brand specific data */
549 id_t            zone_defaultcid; /* dflt scheduling class id */
550 kstat_t         *zone_swapresv_kstat;
551 kstat_t         *zone_lockedmem_kstat;
552 /*
553  * zone_dl_list is protected by zone_lock
554  */
555 list_t          zone_dl_list;
556 netstack_t      *zone_netstack;
557 struct cpucap   *zone_cpucap;    /* CPU caps data */
558 /*
559  * Solaris Auditing per-zone audit context
560  */
561 struct au_kcontext *zone_audit_kctx;
562 /*
563  * For private use by mntfs.
564  */
565 struct mntelem   *zone_mntfs_db;
566 krwlock_t        zone_mntfs_db_lock;

568 struct klpd_reg  *zone_pfexecd;

570 char            *zone_fs_allowed;
571 rctl_qty_t      zone_nprocs;     /* number of processes in the zone */
572 rctl_qty_t      zone_nprocs_ctl; /* current limit protected by */
573 /* zone_rctls->rcls_lock */
574 kstat_t         *zone_nprocs_kstat;

576 kmutex_t        zone_mcap_lock; /* protects mcap statistics */
577 kstat_t         *zone_mcap_ksp;
578 zone_mcap_kstat_t *zone_mcap_stats;
579 uint64_t        zone_pgpgin;     /* pages paged in */
580 uint64_t        zone_anonpgin;   /* anon pages paged in */
581 uint64_t        zone_execpgin;   /* exec pages paged in */
582 uint64_t        zone_fspgin;     /* fs pages paged in */
583 uint64_t        zone_anon_alloc_fail; /* cnt of anon alloc fails */

585 psecflags_t     zone_secflags; /* default zone security-flags */

```

```

587 #endif /* ! codereview */
588 /*
589  * Misc. kstats and counters for zone cpu-usage aggregation.
590  * The zone_Xtime values are the sum of the micro-state accounting
591  * values for all threads that are running or have run in the zone.
592  * This is tracked in msacct.c as threads change state.
593  * The zone_stime is the sum of the LMS_SYSTEM times.
594  * The zone_utime is the sum of the LMS_USER times.
595  * The zone_wtime is the sum of the LMS_WAIT_CPU times.
596  * As with per-thread micro-state accounting values, these values are
597  * not scaled to nanosecs. The scaling is done by the
598  * zone_misc_kstat_update function when kstats are requested.
599  */
600 kmutex_t        zone_misc_lock; /* protects misc statistics */
601 kstat_t         *zone_misc_ksp;
602 zone_misc_kstat_t *zone_misc_stats;
603 uint64_t        zone_stime;     /* total system time */
604 uint64_t        zone_utime;     /* total user time */
605 uint64_t        zone_wtime;     /* total time waiting in runq */
606 /* fork-fail kstat tracking */
607 uint32_t        zone_ffcap;     /* hit an rctl cap */
608 uint32_t        zone_ffnoprocs; /* get proc/lwp error */
609 uint32_t        zone_ffnomem;   /* as dup/memory error */
610 uint32_t        zone_ffmisc;    /* misc. other error */

612 uint32_t        zone_nested_intp; /* nested interp. kstat */

614 struct loadavg_s zone_loadavg; /* loadavg for this zone */
615 uint64_t        zone_hp_avenrun[3]; /* high-precision avenrun */
616 int             zone_avenrun[3]; /* FSCALEd avg. run queue len */

618 /*
619  * FSS stats updated once per second by fss_decay_usage.
620  */
621 uint32_t        zone_fss_gen;   /* FSS generation cntr */
622 uint64_t        zone_run_ticks; /* tot # of ticks running */

624 /*
625  * DTrace-private per-zone state
626  */
627 int             zone_dtrace_getf; /* # of unprivileged getf(s) */

629 /*
630  * Synchronization primitives used to synchronize between mounts and
631  * zone creation/destruction.
632  */
633 int             zone_mounts_in_progress;
634 kcondvar_t      zone_mount_cv;
635 kmutex_t        zone_mount_lock;
636 } zone_t;

638 /*
639  * Special value of zone_psetid to indicate that pools are disabled.
640  */
641 #define ZONE_PS_INVALID PS_MYID

644 extern zone_t zone0;
645 extern zone_t *global_zone;
646 extern uint_t maxzones;
647 extern rctl_hndl_t rc_zone_nlwps;
648 extern rctl_hndl_t rc_zone_nprocs;

650 extern long zone(int, void *, void *, void *, void *);
651 extern void zone_zsd_init(void);
652 extern void zone_init(void);

```

```

653 extern void zone_hold(zone_t *);
654 extern void zone_rele(zone_t *);
655 extern void zone_init_ref(zone_ref_t *);
656 extern void zone_hold_ref(zone_t *, zone_ref_t *, zone_ref_subsys_t);
657 extern void zone_rele_ref(zone_ref_t *, zone_ref_subsys_t);
658 extern void zone_cred_hold(zone_t *);
659 extern void zone_cred_rele(zone_t *);
660 extern void zone_task_hold(zone_t *);
661 extern void zone_task_rele(zone_t *);
662 extern zone_t *zone_find_by_id(zoneid_t);
663 extern zone_t *zone_find_by_label(const ts_label_t *);
664 extern zone_t *zone_find_by_name(char *);
665 extern zone_t *zone_find_by_any_path(const char *, boolean_t);
666 extern zone_t *zone_find_by_path(const char *);
667 extern zoneid_t getzoneid(void);
668 extern zone_t *zone_find_by_id_nolock(zoneid_t);
669 extern int zone_datalink_walk(zoneid_t, int (*)(datalink_id_t, void *), void *);
670 extern int zone_check_datalink(zoneid_t *, datalink_id_t);
671 extern void zone_loadavg_update();

673 /*
674  * Zone-specific data (ZSD) APIs
675  */
676 /*
677  * The following is what code should be initializing its zone_key_t to if it
678  * calls zone_getspecific() without necessarily knowing that zone_key_create()
679  * has been called on the key.
680  */
681 #define ZONE_KEY_UNINITIALIZED 0

683 typedef uint_t zone_key_t;

685 extern void zone_key_create(zone_key_t *, void (*)(zoneid_t),
686 void (*)(zoneid_t, void *), void (*)(zoneid_t, void *));
687 extern int zone_key_delete(zone_key_t);
688 extern void *zone_getspecific(zone_key_t, zone_t *);
689 extern int zone_setspecific(zone_key_t, zone_t *, const void *);

691 /*
692  * The definition of a zsd_entry is truly private to zone.c and is only
693  * placed here so it can be shared with mdb.
694  *
695  * State maintained for each zone times each registered key, which tracks
696  * the state of the create, shutdown and destroy callbacks.
697  *
698  * zsd_flags is used to keep track of pending actions to avoid holding locks
699  * when calling the create/shutdown/destroy callbacks, since doing so
700  * could lead to deadlocks.
701  */
702 struct zsd_entry {
703     zone_key_t      zsd_key;      /* Key used to lookup value */
704     void            *zsd_data;    /* Caller-managed value */
705     /*
706      * Callbacks to be executed when a zone is created, shutdown, and
707      * destroyed, respectively.
708      */
709     void            (*zsd_create)(zoneid_t);
710     void            (*zsd_shutdown)(zoneid_t, void *);
711     void            (*zsd_destroy)(zoneid_t, void *);
712     list_node_t    zsd_linkage;
713     uint16_t        zsd_flags;    /* See below */
714     kcondvar_t      zsd_cv;
715 };

717 /*
718  * zsd_flags

```

```

719 */
720 #define ZSD_CREATE_NEEDED      0x0001
721 #define ZSD_CREATE_INPROGRESS 0x0002
722 #define ZSD_CREATE_COMPLETED  0x0004
723 #define ZSD_SHUTDOWN_NEEDED   0x0010
724 #define ZSD_SHUTDOWN_INPROGRESS 0x0020
725 #define ZSD_SHUTDOWN_COMPLETED 0x0040
726 #define ZSD_DESTROY_NEEDED    0x0100
727 #define ZSD_DESTROY_INPROGRESS 0x0200
728 #define ZSD_DESTROY_COMPLETED 0x0400

730 #define ZSD_CREATE_ALL \
731     (ZSD_CREATE_NEEDED|ZSD_CREATE_INPROGRESS|ZSD_CREATE_COMPLETED)
732 #define ZSD_SHUTDOWN_ALL \
733     (ZSD_SHUTDOWN_NEEDED|ZSD_SHUTDOWN_INPROGRESS|ZSD_SHUTDOWN_COMPLETED)
734 #define ZSD_DESTROY_ALL \
735     (ZSD_DESTROY_NEEDED|ZSD_DESTROY_INPROGRESS|ZSD_DESTROY_COMPLETED)

737 #define ZSD_ALL_INPROGRESS \
738     (ZSD_CREATE_INPROGRESS|ZSD_SHUTDOWN_INPROGRESS|ZSD_DESTROY_INPROGRESS)

740 /*
741  * Macros to help with zone visibility restrictions.
742  */

744 /*
745  * Is process in the global zone?
746  */
747 #define INGLOBALZONE(p) \
748     ((p)->p_zone == global_zone)

750 /*
751  * Can process view objects in given zone?
752  */
753 #define HASZONEACCESS(p, zoneid) \
754     ((p)->p_zone->zone_id == (zoneid) || INGLOBALZONE(p))

756 /*
757  * Convenience macro to see if a resolved path is visible from within a
758  * given zone.
759  *
760  * The basic idea is that the first (zone_rootpathlen - 1) bytes of the
761  * two strings must be equal. Since the rootpathlen has a trailing '/',
762  * we want to skip everything in the path up to (but not including) the
763  * trailing '/'.
764  */
765 #define ZONE_PATH_VISIBLE(path, zone) \
766     (strcmp((path), (zone)->zone_rootpath, \
767             (zone)->zone_rootpathlen - 1) == 0)

769 /*
770  * Convenience macro to go from the global view of a path to that seen
771  * from within said zone. It is the responsibility of the caller to
772  * ensure that the path is a resolved one (ie, no '..', or '.',s), and is
773  * in fact visible from within the zone.
774  */
775 #define ZONE_PATH_TRANSLATE(path, zone) \
776     (ASSERT(ZONE_PATH_VISIBLE(path, zone)), \
777      (path) + (zone)->zone_rootpathlen - 2)

779 /*
780  * Special processes visible in all zones.
781  */
782 #define ZONE_SPECIALPID(x)      ((x) == 0 || (x) == 1)

784 /*

```

```

785 * Zone-safe version of thread_create() to be used when the caller wants to
786 * create a kernel thread to run within the current zone's context.
787 */
788 extern kthread_t *zthread_create(caddr_t, size_t, void (*)(), void *, size_t,
789     pri_t);
790 extern void zthread_exit(void);

792 /*
793 * Functions for an external observer to register interest in a zone's status
794 * change. Observers will be woken up when the zone status equals the status
795 * argument passed in (in the case of zone_status_timedwait, the function may
796 * also return because of a timeout; zone_status_wait_sig may return early due
797 * to a signal being delivered; zone_status_timedwait_sig may return for any of
798 * the above reasons).
799 *
800 * Otherwise these behave identically to cv_timedwait(), cv_wait(), and
801 * cv_wait_sig() respectively.
802 */
803 extern clock_t zone_status_timedwait(zone_t *, clock_t, zone_status_t);
804 extern clock_t zone_status_timedwait_sig(zone_t *, clock_t, zone_status_t);
805 extern void zone_status_wait(zone_t *, zone_status_t);
806 extern int zone_status_wait_sig(zone_t *, zone_status_t);

808 /*
809 * Get the status of the zone (at the time it was called). The state may
810 * have progressed by the time it is returned.
811 */
812 extern zone_status_t zone_status_get(zone_t *);

814 /*
815 * Safely get the hostid of the specified zone (defaults to machine's hostid
816 * if the specified zone doesn't emulate a hostid). Passing NULL retrieves
817 * the global zone's (i.e., physical system's) hostid.
818 */
819 extern uint32_t zone_get_hostid(zone_t *);

821 /*
822 * Get the "kcred" credentials corresponding to the given zone.
823 */
824 extern struct cred *zone_get_kcred(zoneid_t);

826 /*
827 * Get/set the pool the zone is currently bound to.
828 */
829 extern struct pool *zone_pool_get(zone_t *);
830 extern void zone_pool_set(zone_t *, struct pool *);

832 /*
833 * Get/set the pset the zone is currently using.
834 */
835 extern psetid_t zone_pset_get(zone_t *);
836 extern void zone_pset_set(zone_t *, psetid_t);

838 /*
839 * Get the number of cpus/online-cpus visible from the given zone.
840 */
841 extern int zone_ncpus_get(zone_t *);
842 extern int zone_ncpus_online_get(zone_t *);

844 /*
845 * Returns true if the named pool/dataset is visible in the current zone.
846 */
847 extern int zone_dataset_visible(const char *, int *);

849 /*
850 * zone version of kadmin()

```

```

851 */
852 extern int zone_kadmin(int, int, const char *, cred_t *);
853 extern void zone_shutdown_global(void);

855 extern void mount_in_progress(zone_t *);
856 extern void mount_completed(zone_t *);

858 extern int zone_walk(int (*)(zone_t *, void *), void *);

860 extern rctl_hdl_t rc_zone_locked_mem;
861 extern rctl_hdl_t rc_zone_max_swap;
862 extern rctl_hdl_t rc_zone_max_lofi;

864 #endif /* _KERNEL */

866 #ifdef __cplusplus
867 }
868 #endif

870 #endif /* _SYS_ZONE_H */

```



```

*****
10217 Thu Jun 30 22:03:49 2016
new/usr/src/uts/common/syscall/ppriv.c
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
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 (c) 2003, 2010, Oracle and/or its affiliates. All rights reserved.
23  */

25 #include <sys/param.h>
26 #include <sys/types.h>
27 #include <sys/sysmacros.h>
28 #include <sys/system.h>
29 #include <sys/cred_impl.h>
30 #include <sys/errno.h>
31 #include <sys/klpd.h>
32 #include <sys/proc.h>
33 #include <sys/priv_impl.h>
34 #include <sys/policy.h>
35 #include <sys/ddi.h>
36 #include <sys/thread.h>
37 #include <sys/cmn_err.h>
38 #include <c2/audit.h>

40 /*
41  * System call support for manipulating privileges.
42  *
43  *
44  * setppriv(2) - set process privilege set
45  * getppriv(2) - get process privilege set
46  * getprivimplinfo(2) - get process privilege implementation information
47  * setpflags(2) - set process (privilege) flags
48  * getpflags(2) - get process (privilege) flags
49  */

51 /*
52  * setppriv (priv_op_t, priv_ptype_t, priv_set_t)
53  */
54 static int
55 setppriv(priv_op_t op, priv_ptype_t type, priv_set_t *in_pset)
56 {
57     priv_set_t    pset, *target;
58     cred_t        *cr, *pcr;

```

```

59     proc_t        *p;
60     boolean_t     donocd = B_FALSE;

62     if (!PRIV_VALIDSET(type) || !PRIV_VALIDDOP(op))
63         return (set_errno(EINVAL));

65     if (copyin(in_pset, &pset, sizeof (priv_set_t)))
66         return (set_errno(EFAULT));

68     p = ttoproc(curthread);
69     cr = cralloc();
70     mutex_enter(&p->p_crlock);

72 retry:
73     pcr = p->p_cred;

75     if (AU_AUDITING())
76         audit_setppriv(op, type, &pset, pcr);

78     /*
79      * Filter out unallowed request (bad op and bad type)
80      */
81     switch (op) {
82     case PRIV_ON:
83     case PRIV_SET:
84         /*
85          * Turning on privileges; the limit set cannot grow,
86          * other sets can but only as long as they remain subsets
87          * of P. Only immediately after exec holds that P <= L.
88          */
89         if (type == PRIV_LIMIT &&
90             !priv_issubset(&pset, &CR_LPRIV(pcr))) {
91             mutex_exit(&p->p_crlock);
92             crfree(cr);
93             return (set_errno(EPERM));
94         }
95         if (!priv_issubset(&pset, &CR_OPPRIV(pcr)) &&
96             !priv_issubset(&pset, priv_getset(pcr, type))) {
97             mutex_exit(&p->p_crlock);
98             /* Policy override should not grow beyond L either */
99             if (type != PRIV_INHERITABLE ||
100                 !priv_issubset(&pset, &CR_LPRIV(pcr)) ||
101                 secpolicy_require_privs(CRED(), &pset) != 0) {
102                 crfree(cr);
103                 return (set_errno(EPERM));
104             }
105             mutex_enter(&p->p_crlock);
106             if (pcr != p->p_cred)
107                 goto retry;
108             donocd = B_TRUE;
109         }
110         break;

112     case PRIV_OFF:
113         /* PRIV_OFF is always allowed */
114         break;
115     }

117     /*
118      * OK! everything is cool.
119      * Do cred COW.
120      */
121     crcopy_to(pcr, cr);

123     /*
124      * If we change the effective, permitted or limit set, we attain

```

```

125     * "privilege awareness".
126     */
127     if (type != PRIV_INHERITABLE)
128         priv_set_PA(cr);

130     target = &(CR_PRIVS(cr)->crprivs[type]);

132     switch (op) {
133     case PRIV_ON:
134         priv_union(&pset, target);
135         break;
136     case PRIV_OFF:
137         priv_inverse(&pset);
138         priv_intersect(target, &pset);

140         /*
141          * Fall-thru to set target and change other process
142          * privilege sets.
143          */
144         /*FALLTHRU*/

146     case PRIV_SET:
147         *target = pset;

149         /*
150          * Take privileges no longer permitted out
151          * of other effective sets as well.
152          * Limit set is enforced at exec() time.
153          */
154         if (type == PRIV_PERMITTED)
155             priv_intersect(&pset, &CR_EPRIV(cr));
156         break;
157     }

159     /*
160     * When we give up privileges not in the inheritable set,
161     * set SNOCD if not already set; first we compute the
162     * privileges removed from P using Diff = (~P') & P
163     * and then we check whether the removed privileges are
164     * a subset of I. If we retain uid 0, all privileges
165     * are required anyway so don't set SNOCD.
166     */
167     if (type == PRIV_PERMITTED && (p->p_flag & SNOCD) == 0 &&
168         cr->cr_uid != 0 && cr->cr_ruid != 0 && cr->cr_suid != 0) {
169         priv_set_t diff = CR_OPPRIV(cr);
170         priv_inverse(&diff);
171         priv_intersect(&CR_OPPRIV(pcr), &diff);
172         donocd = !priv_issubset(&diff, &CR_IPRIV(cr));
173     }

175     p->p_cred = cr;
176     mutex_exit(&p->p_crlock);

178     if (donocd) {
179         mutex_enter(&p->p_lock);
180         p->p_flag |= SNOCD;
181         mutex_exit(&p->p_lock);
182     }

184     /*
185     * The basic_test privilege should not be removed from E;
186     * if that has happened, then some programmer typically set the E/P to
187     * empty. That is not portable.
188     */
189     if ((type == PRIV_EFFECTIVE || type == PRIV_PERMITTED) &&
190         priv_basic_test >= 0 && !PRIV_ISMEMBER(target, priv_basic_test)) {

```

```

190         priv_basic_test >= 0 && !PRIV_ISASSERT(target, priv_basic_test)) {
191             proc_t *p = curproc;
192             pid_t pid = p->p_pid;
193             char *fn = PTOU(p)->u_comm;

195             cmn_err(CE_WARN, "%s[%d]: setppriv: basic_test privilege "
196                  "removed from E/P", fn, pid);
197         }

199         crset(p, cr);          /* broadcast to process threads */

201         return (0);
202     }

```

unchanged_portion_omitted

```

*****
2635 Thu Jun 30 22:03:49 2016
new/usr/src/uts/common/syscall/psecflags.c
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
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 /* Copyright 2015, Richard Lowe. */

14 #include <sys/ddi.h>
15 #include <sys/errno.h>
16 #include <sys/policy.h>
17 #include <sys/proc.h>
18 #include <sys/procset.h>
19 #include <sys/system.h>
20 #include <sys/types.h>
21 #include <c2/audit.h>

23 struct psdargs {
24     psecflagwhich_t which;
25     const secflagdelta_t *delta;
26 };

28 void
29 secflags_apply_delta(secflagset_t *set, const secflagdelta_t *delta)
30 {
31     if (delta->psd_ass_active) {
32         secflags_copy(set, &delta->psd_assign);
33     } else {
34         if (!secflags_isempty(delta->psd_add)) {
35             secflags_union(set, &delta->psd_add);
36         }
37         if (!secflags_isempty(delta->psd_rem)) {
38             secflags_difference(set, &delta->psd_rem);
39         }
40     }
41 }

44 static int
45 psecdo(proc_t *p, struct psdargs *args)
46 {
47     secflagset_t *set;
48     int ret = 0;

50     mutex_enter(&p->p_lock);

52     if (secpolicy_psecflags(CRED(), p, curproc) != 0) {
53         ret = EPERM;
54         goto out;
55     }

57     ASSERT(args->which != PSF_EFFECTIVE);

```

```

59     if (!psecflags_validate_delta(&p->p_secflags, args->delta)) {
60         ret = EINVAL;
61         goto out;
62     }

64     if (AU_AUDITING())
65         audit_psecflags(p, args->which, args->delta);

67     switch (args->which) {
68     case PSF_INHERIT:
69         set = &p->p_secflags.psf_inherit;
70         break;
71     case PSF_LOWER:
72         set = &p->p_secflags.psf_lower;
73         break;
74     case PSF_UPPER:
75         set = &p->p_secflags.psf_upper;
76         break;
77     }

79     secflags_apply_delta(set, args->delta);

81     /*
82      * Add any flag now in the lower that is not in the inheritable.
83      */
84     secflags_union(&p->p_secflags.psf_inherit, &p->p_secflags.psf_lower);

86 out:
87     mutex_exit(&p->p_lock);
88     return (ret);
89 }

91 int
92 psecflags(procset_t *psp, psecflagwhich_t which, secflagdelta_t *ap)
93 {
94     procset_t procset;
95     secflagdelta_t args;
96     int rv = 0;
97     struct psdargs psd = {
98         .which = which,
99     };

101     /* Can never change the effective flags */
102     if (psd.which == PSF_EFFECTIVE)
103         return (EINVAL);

105     if (copyin(psp, &procset, sizeof (procset)) != 0)
106         return (set_errno(EFAULT));

108     if (copyin(ap, &args, sizeof (secflagdelta_t)) != 0)
109         return (set_errno(EFAULT));

111     psd.delta = &args;

113     /* secflags are per-process, procset must be in terms of processes */
114     if ((procset.p_lidtype == P_LWPID) ||
115         (procset.p_ridtype == P_LWPID))
116         return (set_errno(EINVAL));

118     rv = dotoprocs(&procset, psecdo, (caddr_t)&psd);

120     return (rv ? set_errno(rv) : 0);
121 }
122 #endif /* ! codereview */

```

new/usr/src/uts/i86pc/os/mlsetup.c

1

```
*****
14166 Thu Jun 30 22:03:50 2016
new/usr/src/uts/i86pc/os/mlsetup.c
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
_____unchanged_portion_omitted_____

96 /*
97  * Setup routine called right before main(). Interposing this function
98  * before main() allows us to call it in a machine-independent fashion.
99  */
100 void
101 mlsetup(struct regs *rp)
102 {
103     u_longlong_t prop_value;
104     extern struct classfuncs sys_classfuncs;
105     extern disp_t cpu0_disp;
106     extern char t0stack[];
107     extern int post_fastreboot;
108     extern uint64_t plat_dr_options;

110     ASSERT_STACK_ALIGNED();

112     /*
113      * initialize cpu_self
114      */
115     cpu[0]->cpu_self = cpu[0];

117 #if defined(__xpv)
118     /*
119      * Point at the hypervisor's virtual cpu structure
120      */
121     cpu[0]->cpu_m.mcpu_vcpcu_info = &HYPERVISOR_shared_info->vcpcu_info[0];
122 #endif

124     /*
125      * check if we've got special bits to clear or set
126      * when checking cpu features
127      */

129     if (bootprop_getval("cpuid_feature_ecx_include", &prop_value) != 0)
130         cpuid_feature_ecx_include = 0;
131     else
132         cpuid_feature_ecx_include = (uint32_t)prop_value;

134     if (bootprop_getval("cpuid_feature_ecx_exclude", &prop_value) != 0)
135         cpuid_feature_ecx_exclude = 0;
136     else
137         cpuid_feature_ecx_exclude = (uint32_t)prop_value;

139     if (bootprop_getval("cpuid_feature_edx_include", &prop_value) != 0)
140         cpuid_feature_edx_include = 0;
141     else
142         cpuid_feature_edx_include = (uint32_t)prop_value;

144     if (bootprop_getval("cpuid_feature_edx_exclude", &prop_value) != 0)
145         cpuid_feature_edx_exclude = 0;
146     else
147         cpuid_feature_edx_exclude = (uint32_t)prop_value;

149     /*
150      * Initialize idt0, gdt0, ldt0_default, ktss0 and dftss.
```

new/usr/src/uts/i86pc/os/mlsetup.c

2

```
151     /*
152     init_desctbls();

154     /*
155      * lgrp_init() and possibly cpuid_pass1() need PCI config
156      * space access
157      */
158 #if defined(__xpv)
159     if (DOMAIN_IS_INITDOMAIN(xen_info))
160         pci_cfgspace_init();
161 #else
162     pci_cfgspace_init();
163     /*
164      * Initialize the platform type from CPU 0 to ensure that
165      * determine_platform() is only ever called once.
166      */
167     determine_platform();
168 #endif

170     /*
171      * The first lightweight pass (pass0) through the cpuid data
172      * was done in locore before mlsetup was called. Do the next
173      * pass in C code.
174      *
175      * The x86_featureset is initialized here based on the capabilities
176      * of the boot CPU. Note that if we choose to support CPUs that have
177      * different feature sets (at which point we would almost certainly
178      * want to set the feature bits to correspond to the feature
179      * minimum) this value may be altered.
180      */
181     cpuid_pass1(cpu[0], x86_featureset);

183 #if !defined(__xpv)
184     if ((get_hwenv() & HW_XEN_HVM) != 0)
185         xen_hvm_init();

187     /*
188      * Before we do anything with the TSCs, we need to work around
189      * Intel erratum BT81. On some CPUs, warm reset does not
190      * clear the TSC. If we are on such a CPU, we will clear TSC ourselves
191      * here. Other CPUs will clear it when we boot them later, and the
192      * resulting skew will be handled by tsc_sync_master()/_slave();
193      * note that such skew already exists and has to be handled anyway.
194      *
195      * We do this only on metal. This same problem can occur with a
196      * hypervisor that does not happen to virtualise a TSC that starts from
197      * zero, regardless of CPU type; however, we do not expect hypervisors
198      * that do not virtualise TSC that way to handle writes to TSC
199      * correctly, either.
200      */
201     if (get_hwenv() == HW_NATIVE &&
202         cpuid_getvendor(CPU) == X86_VENDOR_Intel &&
203         cpuid_getfamily(CPU) == 6 &&
204         (cpuid_getmodel(CPU) == 0x2d || cpuid_getmodel(CPU) == 0x3e) &&
205         is_x86_feature(x86_featureset, X86FSET_TSC)) {
206         (void) wrmsr(REG_TSC, 0UL);
207     }

209     /*
210      * Patch the tsc_read routine with appropriate set of instructions,
211      * depending on the processor family and architecture, to read the
212      * time-stamp counter while ensuring no out-of-order execution.
213      * Patch it while the kernel text is still writable.
214      *
215      * Note: tsc_read is not patched for intel processors whose family
216      * is >6 and for amd whose family >f (in case they don't support rdtscp
```

```

217 * instruction, unlikely). By default tsc_read will use cpuid for
218 * serialization in such cases. The following code needs to be
219 * revisited if intel processors of family >= f retains the
220 * instruction serialization nature of mfence instruction.
221 * Note: tsc_read is not patched for x86 processors which do
222 * not support "mfence". By default tsc_read will use cpuid for
223 * serialization in such cases.
224 *
225 * The Xen hypervisor does not correctly report whether rdtscp is
226 * supported or not, so we must assume that it is not.
227 */
228 if ((get_hwenv() & HW_XEN_HVM) == 0 &&
229     is_x86_feature(x86_featureset, X86FSET_TSCP))
230     patch_tsc_read(X86_HAVE_TSCP);
231 else if (cpuid_getvendor(CPU) == X86_VENDOR_AMD &&
232         cpuid_getfamily(CPU) <= 0xf &&
233         is_x86_feature(x86_featureset, X86FSET_SSE2))
234     patch_tsc_read(X86_TSC_MFENCE);
235 else if (cpuid_getvendor(CPU) == X86_VENDOR_Intel &&
236         cpuid_getfamily(CPU) <= 6 &&
237         is_x86_feature(x86_featureset, X86FSET_SSE2))
238     patch_tsc_read(X86_TSC_LFENCE);
239
240 #endif /* !__xpv */
241
242 #if defined(__i386) && !defined(__xpv)
243 /*
244  * Some i386 processors do not implement the rdtsc instruction,
245  * or at least they do not implement it correctly. Patch them to
246  * return 0.
247  */
248 if (!is_x86_feature(x86_featureset, X86FSET_TSC))
249     patch_tsc_read(X86_NO_TSC);
250 #endif /* __i386 && !__xpv */
251
252 #if defined(__amd64) && !defined(__xpv)
253     patch_memops(cpuid_getvendor(CPU));
254 #endif /* __amd64 && !__xpv */
255
256 #if !defined(__xpv)
257 /* XXPV what, if anything, should be dorked with here under xen? */
258
259 /*
260  * While we're thinking about the TSC, let's set up %cr4 so that
261  * userland can issue rdtsc, and initialize the TSC_AUX value
262  * (the cpuid) for the rdtscp instruction on appropriately
263  * capable hardware.
264  */
265 if (is_x86_feature(x86_featureset, X86FSET_TSCP))
266     setcr4(getcr4() & ~CR4_TSD);
267
268 if (is_x86_feature(x86_featureset, X86FSET_TSCP))
269     (void) wrmsr(MSR_AMD_TSCAUX, 0);
270
271 /*
272  * Let's get the other %cr4 stuff while we're here. Note, we defer
273  * enabling CR4_SMAP until startup_end(); however, that's importantly
274  * before we start other CPUs. That ensures that it will be synced out
275  * to other CPUs.
276  */
277 if (is_x86_feature(x86_featureset, X86FSET_DE))
278     setcr4(getcr4() | CR4_DE);
279
280 if (is_x86_feature(x86_featureset, X86FSET_SMEP))
281     setcr4(getcr4() | CR4_SMEP);
282 #endif /* __xpv */

```

```

284 /*
285  * initialize t0
286  */
287 t0.t_stk = (caddr_t)rp - MINFRAME;
288 t0.t_stkbase = t0stk;
289 t0.t_pri = maxclsyspri - 3;
290 t0.t_schedflag = TS_LOAD | TS_DONT_SWAP;
291 t0.t_procp = &p0;
292 t0.t_plockp = &p0lock.pl_lock;
293 t0.t_lwp = &lwp0;
294 t0.t_forw = &t0;
295 t0.t_back = &t0;
296 t0.t_next = &t0;
297 t0.t_prev = &t0;
298 t0.t_cpu = cpu[0];
299 t0.t_disp_queue = &cpu0_disp;
300 t0.t_bind_cpu = PBIND_NONE;
301 t0.t_bind_pset = PS_NONE;
302 t0.t_bindflag = (uchar_t)default_binding_mode;
303 t0.t_cpupart = &cp_default;
304 t0.t_clfuncs = &sys_classfuncs.thread;
305 t0.t_copyops = NULL;
306 THREAD_ONPROC(&t0, CPU);
307
308 lwp0.lwp_thread = &t0;
309 lwp0.lwp_regs = (void *)rp;
310 lwp0.lwp_procp = &p0;
311 t0.t_tid = p0.p_lwpcnt = p0.p_lwprcnt = p0.p_lwpid = 1;
312
313 p0.p_exec = NULL;
314 p0.p_stat = SRUN;
315 p0.p_flag = SSSYS;
316 p0.p_tlist = &t0;
317 p0.p_stksize = 2*PAGESIZE;
318 p0.p_stkpageszc = 0;
319 p0.p_as = &kas;
320 p0.p_lockp = &p0lock;
321 p0.p_brkpageszc = 0;
322 p0.p_tl_lgrp_id = LGRP_NONE;
323 p0.p_tr_lgrp_id = LGRP_NONE;
324 psecflags_default(&p0.p_secflags);
325
326 #endif /* !codereview */
327 sigorset(&p0.p_ignore, &ignoredefault);
328
329 CPU->cpu_thread = &t0;
330 bzero(&cpu0_disp, sizeof (disp_t));
331 CPU->cpu_disp = &cpu0_disp;
332 CPU->cpu_disp->disp_cpu = CPU;
333 CPU->cpu_dispthread = &t0;
334 CPU->cpu_idle_thread = &t0;
335 CPU->cpu_flags = CPU_READY | CPU_RUNNING | CPU_EXISTS | CPU_ENABLE;
336 CPU->cpu_dispatch_pri = t0.t_pri;
337
338 CPU->cpu_id = 0;
339
340 CPU->cpu_pri = 12; /* initial PIL for the boot CPU */
341
342 /*
343  * The kernel doesn't use LDTs unless a process explicitly requests one.
344  */
345 p0.p_ldt_desc = null_sdesc;
346
347 /*
348  * Initialize thread/cpu microstate accounting

```

```

349  */
350  init_mstate(&t0, LMS_SYSTEM);
351  init_cpu_mstate(CPU, CMS_SYSTEM);

353  /*
354  * Initialize lists of available and active CPUs.
355  */
356  cpu_list_init(CPU);

358  pg_cpu_bootstrap(CPU);

360  /*
361  * Now that we have taken over the GDT, IDT and have initialized
362  * active CPU list it's time to inform kmdb if present.
363  */
364  if (boothowto & RB_DEBUG)
365      kdi_idt_sync();

367  /*
368  * Explicitly set console to text mode (0x3) if this is a boot
369  * post Fast Reboot, and the console is set to CONS_SCREEN_TEXT.
370  */
371  if (post_fastreboot && boot_console_type(NULL) == CONS_SCREEN_TEXT)
372      set_console_mode(0x3);

374  /*
375  * If requested (boot -d) drop into kmdb.
376  *
377  * This must be done after cpu_list_init() on the 64-bit kernel
378  * since taking a trap requires that we re-compute gsbase based
379  * on the cpu list.
380  */
381  if (boothowto & RB_DEBUGENTER)
382      kmdb_enter();

384  cpu_vm_data_init(CPU);

386  rp->r_fp = 0; /* terminate kernel stack traces! */

388  prom_init("kernel", (void *)NULL);

390  /* User-set option overrides firmware value. */
391  if (bootprop_getval(PLAT_DR_OPTIONS_NAME, &prop_value) == 0) {
392      plat_dr_options = (uint64_t)prop_value;
393  }
394  #if defined(__xpv)
395  /* No support of DR operations on xpv */
396  plat_dr_options = 0;
397  #else /* __xpv */
398  /* Flag PLAT_DR_FEATURE_ENABLED should only be set by DR driver. */
399  plat_dr_options &= ~PLAT_DR_FEATURE_ENABLED;
400  #ifnndef __amd64
401  /* Only enable CPU/memory DR on 64 bits kernel. */
402  plat_dr_options &= ~PLAT_DR_FEATURE_MEMORY;
403  plat_dr_options &= ~PLAT_DR_FEATURE_CPU;
404  #endif /* __amd64 */
405  #endif /* __xpv */

407  /*
408  * Get value of "plat_dr_physmax" boot option.
409  * It overrides values calculated from MSCT or SRAT table.
410  */
411  if (bootprop_getval(PLAT_DR_PHYSMAX_NAME, &prop_value) == 0) {
412      plat_dr_physmax = ((uint64_t)prop_value) >> PAGESHIFT;
413  }

```

```

415  /* Get value of boot_ncpus. */
416  if (bootprop_getval(BOOT_NCPUS_NAME, &prop_value) != 0) {
417      boot_ncpus = NCPU;
418  } else {
419      boot_ncpus = (int)prop_value;
420      if (boot_ncpus <= 0 || boot_ncpus > NCPU)
421          boot_ncpus = NCPU;
422  }

424  /*
425  * Set max_ncpus and boot_max_ncpus to boot_ncpus if platform doesn't
426  * support CPU DR operations.
427  */
428  if (plat_dr_support_cpu() == 0) {
429      max_ncpus = boot_max_ncpus = boot_ncpus;
430  } else {
431      if (bootprop_getval(PLAT_MAX_NCPUS_NAME, &prop_value) != 0) {
432          max_ncpus = NCPU;
433      } else {
434          max_ncpus = (int)prop_value;
435          if (max_ncpus <= 0 || max_ncpus > NCPU) {
436              max_ncpus = NCPU;
437          }
438          if (boot_ncpus > max_ncpus) {
439              boot_ncpus = max_ncpus;
440          }
441      }

443      if (bootprop_getval(BOOT_MAX_NCPUS_NAME, &prop_value) != 0) {
444          boot_max_ncpus = boot_ncpus;
445      } else {
446          boot_max_ncpus = (int)prop_value;
447          if (boot_max_ncpus <= 0 || boot_max_ncpus > NCPU) {
448              boot_max_ncpus = boot_ncpus;
449          } else if (boot_max_ncpus > max_ncpus) {
450              boot_max_ncpus = max_ncpus;
451          }
452      }
453  }

455  /*
456  * Initialize the lgrp framework
457  */
458  lgrp_init(LGRP_INIT_STAGE1);

460  if (boothowto & RB_HALT) {
461      prom_printf("unix: kernel halted by -h flag\n");
462      prom_enter_mon();
463  }

465  ASSERT_STACK_ALIGNED();

467  /*
468  * Fill out cpu_ucode_info. Update microcode if necessary.
469  */
470  ucode_check(CPU);

472  if (workaround_errata(CPU) != 0)
473      panic("critical workaround(s) missing for boot cpu");
474  }

477  void
478  mach_modpath(char *path, const char *filename)
479  {
480      /*

```

```
481     * Construct the directory path from the filename.
482     */
484     int len;
485     char *p;
486     const char isastr[] = "/amd64";
487     size_t isalen = strlen(isastr);
489     len = strlen(SYSTEM_BOOT_PATH "/kernel");
490     (void) strcpy(path, SYSTEM_BOOT_PATH "/kernel ");
491     path += len + 1;
493     if ((p = strchr(filename, '/')) == NULL)
494         return;
496     while (p > filename && *(p - 1) == '/')
497         p--; /* remove trailing '/' characters */
498     if (p == filename)
499         p++; /* so "/" -is- the modpath in this case */
501     /*
502     * Remove optional isa-dependent directory name - the module
503     * subsystem will put this back again (!)
504     */
505     len = p - filename;
506     if (len > isalen &&
507         strncmp(&filename[len - isalen], isastr, isalen) == 0)
508         p -= isalen;
510     /*
511     * "/platform/mumblefrotz" + " " + MOD_DEFPATH
512     */
513     len += (p - filename) + 1 + strlen(MOD_DEFPATH) + 1;
514     (void) strncpy(path, filename, p - filename);
515 }
```

```

*****
98287 Thu Jun 30 22:03:51 2016
new/usr/src/uts/i86pc/vm/vm_machdep.c
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
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 (c) 1992, 2010, Oracle and/or its affiliates. All rights reserved.
23 */
24 /*
25 * Copyright (c) 2010, Intel Corporation.
26 * All rights reserved.
27 */
28 /* Copyright (c) 1984, 1986, 1987, 1988, 1989 AT&T */
29 /* All Rights Reserved */
30
31
32 /*
33 * Portions of this source code were derived from Berkeley 4.3 BSD
34 * under license from the Regents of the University of California.
35 */
36
37 /*
38 * UNIX machine dependent virtual memory support.
39 */
40
41 #include <sys/types.h>
42 #include <sys/param.h>
43 #include <sys/system.h>
44 #include <sys/user.h>
45 #include <sys/proc.h>
46 #include <sys/kmem.h>
47 #include <sys/vmem.h>
48 #include <sys/buf.h>
49 #include <sys/cpuvar.h>
50 #include <sys/lgrp.h>
51 #include <sys/disp.h>
52 #include <sys/vm.h>
53 #include <sys/mman.h>
54 #include <sys/vnode.h>
55 #include <sys/cred.h>
56 #include <sys/exec.h>
57 #include <sys/exechdr.h>
58 #include <sys/debug.h>

```

```

59 #include <sys/vmsystem.h>
60 #include <sys/swap.h>
61 #include <sys/dumphdr.h>
62 #include <sys/random.h>
63 #endif /* ! codereview */
64
65 #include <vm/hat.h>
66 #include <vm/as.h>
67 #include <vm/seg.h>
68 #include <vm/seg_kp.h>
69 #include <vm/seg_vn.h>
70 #include <vm/page.h>
71 #include <vm/seg_kmem.h>
72 #include <vm/seg_kpm.h>
73 #include <vm/vm_dep.h>
74
75 #include <sys/cpu.h>
76 #include <sys/vm_machparam.h>
77 #include <sys/memlist.h>
78 #include <sys/bootconf.h> /* XXX the memlist stuff belongs in memlist_plat.h */
79 #include <vm/hat_i86.h>
80 #include <sys/x86_archext.h>
81 #include <sys/elf_386.h>
82 #include <sys/cmn_err.h>
83 #include <sys/archsystem.h>
84 #include <sys/machsystem.h>
85 #include <sys/secflags.h>
86 #endif /* ! codereview */
87
88 #include <sys/vtrace.h>
89 #include <sys/ddidmareq.h>
90 #include <sys/promif.h>
91 #include <sys/memnode.h>
92 #include <sys/stack.h>
93 #include <util/qsorth.h>
94 #include <sys/taskq.h>
95
96 #ifdef __xpv
97
98 #include <sys/hypervisor.h>
99 #include <sys/xen_mmu.h>
100 #include <sys/balloon_impl.h>
101
102 /*
103 * domain 0 pages usable for DMA are kept pre-allocated and kept in
104 * distinct lists, ordered by increasing mfn.
105 */
106 static kmutex_t io_pool_lock;
107 static kmutex_t contig_list_lock;
108 static page_t *io_pool_4g; /* pool for 32 bit dma limited devices */
109 static page_t *io_pool_16m; /* pool for 24 bit dma limited legacy devices */
110 static long io_pool_cnt;
111 static long io_pool_cnt_max = 0;
112 #define DEFAULT_IO_POOL_MIN 128
113 static long io_pool_cnt_min = DEFAULT_IO_POOL_MIN;
114 static long io_pool_cnt_lowater = 0;
115 static long io_pool_shrink_attempts; /* how many times did we try to shrink */
116 static long io_pool_shrinks; /* how many times did we really shrink */
117 static long io_pool_grows; /* how many times did we grow */
118 static mfn_t start_mfn = 1;
119 static caddr_t io_pool_kva; /* use to alloc pages when needed */
120
121 static int create_contig_pfnlist(uint_t);
122
123 /*
124 * percentage of phys mem to hold in the i/o pool

```



```

125 */
126 #define DEFAULT_IO_POOL_PCT      2
127 static long io_pool_physmem_pct = DEFAULT_IO_POOL_PCT;
128 static void page_io_pool_sub(page_t **, page_t *, page_t *);
129 int ioalloc_dbg = 0;

131 #endif /* __xpv */

133 uint_t vac_colors = 1;

135 int largepagesupport = 0;
136 extern uint_t page_create_new;
137 extern uint_t page_create_exists;
138 extern uint_t page_create_putbacks;
139 /*
140  * Allow users to disable the kernel's use of SSE.
141  */
142 extern int use_sse_pagecopy, use_sse_pagezero;

144 /*
145  * combined memory ranges from mnode and memranges[] to manage single
146  * mnode/mtype dimension in the page lists.
147  */
148 typedef struct {
149     pfn_t   mn_r_pfnlo;
150     pfn_t   mn_r_pfnhi;
151     int     mn_r_mnode;
152     int     mn_r_memrange; /* index into memranges[] */
153     int     mn_r_next; /* next lower PA mnode range */
154     int     mn_r_exists;
155     /* maintain page list stats */
156     pgcnt_t mn_r_mt_clpgcnt; /* cache list cnt */
157     pgcnt_t mn_r_mt_flgcnt[MMU_PAGE_SIZES]; /* free list cnt per szc */
158     pgcnt_t mn_r_mt_totcnt; /* sum of cache and free lists */
159 #ifndef DEBUG
160     struct mn_r_mts { /* mnode/mtype szc stats */
161         pgcnt_t mn_r_mts_pgcnt;
162         int     mn_r_mts_colors;
163         pgcnt_t *mn_r_mts_pgcnt;
164     }
165 #endif
166 } mnode_range_t;

168 #define MEMRANGEHI(mtype) \
169     ((mtype > 0) ? memranges[mtype - 1] - 1 : physmax)
170 #define MEMRANGELO(mtype) (memranges[mtype])

172 #define MTYPE_FREEMEM(mt) (mnode_ranges[mt].mn_r_mt_totcnt)

174 /*
175  * As the PC architecture evolved memory up was clumped into several
176  * ranges for various historical I/O devices to do DMA.
177  * < 16Meg - ISA bus
178  * < 2Gig - ???
179  * < 4Gig - PCI bus or drivers that don't understand PAE mode
180  *
181  * These are listed in reverse order, so that we can skip over unused
182  * ranges on machines with small memories.
183  *
184  * For now under the Hypervisor, we'll only ever have one memrange.
185  */
186 #define PFN_4GIG      0x100000
187 #define PFN_16MEG    0x1000
188 /* Indices into the memory range (arch_memranges) array. */
189 #define MRI_4G       0
190 #define MRI_2G       1

```

```

191 #define MRI_16M      2
192 #define MRI_0       3
193 static pfn_t arch_memranges[NUM_MEM_RANGES] = {
194     PFN_4GIG, /* pfn range for 4G and above */
195     0x80000, /* pfn range for 2G-4G */
196     PFN_16MEG, /* pfn range for 16M-2G */
197     0x00000, /* pfn range for 0-16M */
198 };
199 pfn_t *memranges = &arch_memranges[0];
200 int nranges = NUM_MEM_RANGES;

202 /*
203  * This combines mem_node_config and memranges into one data
204  * structure to be used for page list management.
205  */
206 mnode_range_t *mnode_ranges;
207 int mnode_rangecnt;
208 int mtype4g;
209 int mtype16m;
210 int mtype_top; /* index of highest pfn'ed mnode range */

212 /*
213  * 4g memory management variables for systems with more than 4g of memory:
214  *
215  * physical memory below 4g is required for 32bit dma devices and, currently,
216  * for kmem memory. On systems with more than 4g of memory, the pool of memory
217  * below 4g can be depleted without any paging activity given that there is
218  * likely to be sufficient memory above 4g.
219  *
220  * physmax4g is set true if the largest pfn is over 4g. The rest of the
221  * 4g memory management code is enabled only when physmax4g is true.
222  *
223  * maxmem4g is the count of the maximum number of pages on the page lists
224  * with physical addresses below 4g. It can be a lot less than 4g given that
225  * BIOS may reserve large chunks of space below 4g for hot plug pci devices,
226  * agp aperture etc.
227  *
228  * freemem4g maintains the count of the number of available pages on the
229  * page lists with physical addresses below 4g.
230  *
231  * DESFREE4G specifies the desired amount of below 4g memory. It defaults to
232  * 6% (desfree4gshift = 4) of maxmem4g.
233  *
234  * RESTRICT4G_ALLOC returns true if freemem4g falls below DESFREE4G
235  * and the amount of physical memory above 4g is greater than freemem4g.
236  * In this case, page_get_* routines will restrict below 4g allocations
237  * for requests that don't specifically require it.
238  */

240 #define DESFREE4G (maxmem4g >> desfree4gshift)

242 #define RESTRICT4G_ALLOC \
243     (physmax4g && (freemem4g < DESFREE4G) && ((freemem4g << 1) < freemem))

245 static pgcnt_t maxmem4g;
246 static pgcnt_t freemem4g;
247 static int physmax4g;
248 static int desfree4gshift = 4; /* maxmem4g shift to derive DESFREE4G */

250 /*
251  * 16m memory management:
252  *
253  * reserve some amount of physical memory below 16m for legacy devices.
254  *
255  * RESTRICT16M_ALLOC returns true if there are sufficient free pages above
256  * 16m or if the 16m pool drops below DESFREE16M.

```

```

257 *
258 * In this case, general page allocations via page_get_{free,cache}list
259 * routines will be restricted from allocating from the 16m pool. Allocations
260 * that require specific pfn ranges (page_get_anylist) and PG_PANIC allocations
261 * are not restricted.
262 */

264 #define FREEMEM16M      MTYPE_FREEMEM(mtypel6m)
265 #define DESFREE16M     desfree16m
266 #define RESTRICT16M_ALLOC(freemem, pgcnt, flags)          \
267     ((freemem != 0) && ((flags & PG_PANIC) == 0) &&      \
268     ((freemem >= (FREEMEM16M)) ||                      \
269     (FREEMEM16M < (DESFREE16M + pgcnt))))

271 static pgcnt_t  desfree16m = 0x380;

273 /*
274 * This can be patched via /etc/system to allow old non-PAE aware device
275 * drivers to use kmem_alloc'd memory on 32 bit systems with > 4Gig RAM.
276 */
277 int restricted_kmemalloc = 0;

279 #ifdef VM_STATS
280 struct {
281     ulong_t  pga_alloc;
282     ulong_t  pga_notfullrange;
283     ulong_t  pga_nulldmaattr;
284     ulong_t  pga_allocok;
285     ulong_t  pga_allocfailed;
286     ulong_t  pgma_alloc;
287     ulong_t  pgma_allocok;
288     ulong_t  pgma_allocfailed;
289     ulong_t  pgma_alloempty;
290 } pga_vmstats;
291 #endif

293 uint_t  mmu_page_sizes;

295 /* How many page sizes the users can see */
296 uint_t  mmu_exported_page_sizes;

298 /* page sizes that legacy applications can see */
299 uint_t  mmu_legacy_page_sizes;

301 /*
302 * Number of pages in 1 GB. Don't enable automatic large pages if we have
303 * fewer than this many pages.
304 */
305 pgcnt_t  shm_lpg_min_physmem = 1 << (30 - MMU_PAGESHIFT);
306 pgcnt_t  privm_lpg_min_physmem = 1 << (30 - MMU_PAGESHIFT);

308 /*
309 * Maximum and default segment size tunables for user private
310 * and shared anon memory, and user text and initialized data.
311 * These can be patched via /etc/system to allow large pages
312 * to be used for mapping application private and shared anon memory.
313 */
314 size_t  mcntl0_lpsize = MMU_PAGESIZE;
315 size_t  max_uheap_lpsize = MMU_PAGESIZE;
316 size_t  default_uheap_lpsize = MMU_PAGESIZE;
317 size_t  max_ustack_lpsize = MMU_PAGESIZE;
318 size_t  default_ustack_lpsize = MMU_PAGESIZE;
319 size_t  max_privmap_lpsize = MMU_PAGESIZE;
320 size_t  max_uidata_lpsize = MMU_PAGESIZE;
321 size_t  max_utext_lpsize = MMU_PAGESIZE;
322 size_t  max_shm_lpsize = MMU_PAGESIZE;

```

```

325 /*
326 * initialized by page_coloring_init().
327 */
328 uint_t  page_colors;
329 uint_t  page_colors_mask;
330 uint_t  page_coloring_shift;
331 int     cpu_page_colors;
332 static uint_t  l2_colors;

334 /*
335 * Page freelists and cachelists are dynamically allocated once mnoderangecnt
336 * and page_colors are calculated from the l2 cache n-way set size. Within a
337 * mnode range, the page freelist and cachelist are hashed into bins based on
338 * color. This makes it easier to search for a page within a specific memory
339 * range.
340 */
341 #define PAGE_COLORS_MIN 16

343 page_t  ****page_freelists;
344 page_t  ****page_cachelists;

347 /*
348 * Used by page layer to know about page sizes
349 */
350 hw_pagesize_t  hw_page_array[MAX_NUM_LEVEL + 1];

352 kmutex_t      *fpc_mutex[NPC_MUTEX];
353 kmutex_t      *cpc_mutex[NPC_MUTEX];

355 /* Lock to protect mnoderanges array for memory DR operations. */
356 static kmutex_t  mnoderange_lock;

358 /*
359 * Only let one thread at a time try to coalesce large pages, to
360 * prevent them from working against each other.
361 */
362 static kmutex_t  contig_lock;
363 #define CONTIG_LOCK()  mutex_enter(&contig_lock);
364 #define CONTIG_UNLOCK()  mutex_exit(&contig_lock);

366 #define PFN_16M      (mmu_btop((uint64_t)0x1000000))

368 /*
369 * Return the optimum page size for a given mapping
370 */
371 /*ARGSUSED*/
372 size_t
373 map_pgsz(int maptype, struct proc *p, caddr_t addr, size_t len, int memcntl)
374 {
375     level_t  l1 = 0;
376     size_t  pgsz = MMU_PAGESIZE;
377     size_t  max_lpsize;
378     uint_t  msz;

380     ASSERT(maptype != MAPPGSZ_VA);

382     if (maptype != MAPPGSZ_ISM && physmem < privm_lpg_min_physmem) {
383         return (MMU_PAGESIZE);
384     }

386     switch (maptype) {
387     case MAPPGSZ_HEAP:
388     case MAPPGSZ_STK:

```

```

389     max_lpsize = memcntl ? mcntl0_lpsize : (maptype ==
390     MAPPGSZ_HEAP ? max_uheap_lpsize : max_ustack_lpsize);
391     if (max_lpsize == MMU_PAGESIZE) {
392         return (MMU_PAGESIZE);
393     }
394     if (len == 0) {
395         len = (maptype == MAPPGSZ_HEAP) ? p->p_brkbase +
396             p->p_brksize - p->p_bssbase : p->p_stksize;
397     }
398     len = (maptype == MAPPGSZ_HEAP) ? MAX(len,
399         default_uheap_lpsize) : MAX(len, default_ustack_lpsize);

401     /*
402     * use the pages size that best fits len
403     */
404     for (l = mmu.umax_page_level; l > 0; --l) {
405         if (LEVEL_SIZE(l) > max_lpsize || len < LEVEL_SIZE(l)) {
406             continue;
407         } else {
408             pgsz = LEVEL_SIZE(l);
409         }
410         break;
411     }

413     mszc = (maptype == MAPPGSZ_HEAP ? p->p_brkpageszc :
414         p->p_stkpageszc);
415     if (addr == 0 && (pgsz < hw_page_array[mszc].hp_size)) {
416         pgsz = hw_page_array[mszc].hp_size;
417     }
418     return (pgsz);

420     case MAPPGSZ_ISM:
421         for (l = mmu.umax_page_level; l > 0; --l) {
422             if (len >= LEVEL_SIZE(l))
423                 return (LEVEL_SIZE(l));
424         }
425         return (LEVEL_SIZE(0));
426     }
427     return (pgsz);
428 }

430 static uint_t
431 map_szcvec(caddr_t addr, size_t size, uintptr_t off, size_t max_lpsize,
432 size_t min_physmem)
433 {
434     caddr_t eaddr = addr + size;
435     uint_t szcvec = 0;
436     caddr_t raddr;
437     caddr_t readr;
438     size_t pgsz;
439     int i;

441     if (physmem < min_physmem || max_lpsize <= MMU_PAGESIZE) {
442         return (0);
443     }

445     for (i = mmu.exported_page_sizes - 1; i > 0; i--) {
446         pgsz = page_get_pagesize(i);
447         if (pgsz > max_lpsize) {
448             continue;
449         }
450         raddr = (caddr_t)P2ROUNDUP((uintptr_t)addr, pgsz);
451         readr = (caddr_t)P2ALIGN((uintptr_t)eaddr, pgsz);
452         if (raddr < addr || raddr >= readr) {
453             continue;
454         }

```

```

455         if (P2PHASE((uintptr_t)addr ^ off, pgsz)) {
456             continue;
457         }
458         /*
459         * Set szcvec to the remaining page sizes.
460         */
461         szcvec = ((1 << (i + 1)) - 1) & ~1;
462         break;
463     }
464     return (szcvec);
465 }

467 /*
468 * Return a bit vector of large page size codes that
469 * can be used to map [addr, addr + len) region.
470 */
471 /*ARGSUSED*/
472 uint_t
473 map_pgszcvec(caddr_t addr, size_t size, uintptr_t off, int flags, int type,
474 int memcntl)
475 {
476     size_t max_lpsize = mcntl0_lpsize;

478     if (mmu.max_page_level == 0)
479         return (0);

481     if (flags & MAP_TEXT) {
482         if (!memcntl)
483             max_lpsize = max_utext_lpsize;
484         return (map_szcvec(addr, size, off, max_lpsize,
485             shm_lpg_min_physmem));
487     } else if (flags & MAP_INITDATA) {
488         if (!memcntl)
489             max_lpsize = max_uidata_lpsize;
490         return (map_szcvec(addr, size, off, max_lpsize,
491             privm_lpg_min_physmem));
493     } else if (type == MAPPGSZC_SHM) {
494         if (!memcntl)
495             max_lpsize = max_shm_lpsize;
496         return (map_szcvec(addr, size, off, max_lpsize,
497             shm_lpg_min_physmem));
499     } else if (type == MAPPGSZC_HEAP) {
500         if (!memcntl)
501             max_lpsize = max_uheap_lpsize;
502         return (map_szcvec(addr, size, off, max_lpsize,
503             privm_lpg_min_physmem));
505     } else if (type == MAPPGSZC_STACK) {
506         if (!memcntl)
507             max_lpsize = max_ustack_lpsize;
508         return (map_szcvec(addr, size, off, max_lpsize,
509             privm_lpg_min_physmem));
511     } else {
512         if (!memcntl)
513             max_lpsize = max_privmap_lpsize;
514         return (map_szcvec(addr, size, off, max_lpsize,
515             privm_lpg_min_physmem));
516     }
517 }

519 /*
520 * Handle a pagefault.

```

```

521  */
522  faultcode_t
523  pagefault(
524      caddr_t addr,
525      enum fault_type type,
526      enum seg_rw rw,
527      int iskernel)
528  {
529      struct as *as;
530      struct hat *hat;
531      struct proc *p;
532      kthread_t *t;
533      faultcode_t res;
534      caddr_t base;
535      size_t len;
536      int err;
537      int mapped_red;
538      uintptr_t ea;

540      ASSERT_STACK_ALIGNED();

542      if (INVALID_VADDR(addr))
543          return (FC_NOMAP);

545      mapped_red = segkp_map_red();

547      if (iskernel) {
548          as = &kas;
549          hat = as->a_hat;
550      } else {
551          t = curthread;
552          p = ttoproc(t);
553          as = p->p_as;
554          hat = as->a_hat;
555      }

557      /*
558       * Dispatch pagefault.
559       */
560      res = as_fault(hat, as, addr, 1, type, rw);

562      /*
563       * If this isn't a potential unmapped hole in the user's
564       * UNIX data or stack segments, just return status info.
565       */
566      if (res != FC_NOMAP || iskernel)
567          goto out;

569      /*
570       * Check to see if we happened to faulted on a currently unmapped
571       * part of the UNIX data or stack segments. If so, create a zfod
572       * mapping there and then try calling the fault routine again.
573       */
574      base = p->p_brkbase;
575      len = p->p_brksize;

577      if (addr < base || addr >= base + len) { /* data seg? */
578          base = (caddr_t)p->p_usrstack - p->p_stksize;
579          len = p->p_stksize;
580          if (addr < base || addr >= p->p_usrstack) { /* stack seg? */
581              /* not in either UNIX data or stack segments */
582              res = FC_NOMAP;
583              goto out;
584          }
585      }

```

```

587      /*
588       * the rest of this function implements a 3.X 4.X 5.X compatibility
589       * This code is probably not needed anymore
590       */
591      if (p->p_model == DATAMODEL_ILP32) {

593          /* expand the gap to the page boundaries on each side */
594          ea = P2ROUNDUP((uintptr_t)base + len, MMU_PAGESIZE);
595          base = (caddr_t)P2ALIGN((uintptr_t)base, MMU_PAGESIZE);
596          len = ea - (uintptr_t)base;

598          as_rangelock(as);
599          if (as_gap(as, MMU_PAGESIZE, &base, &len, AH_CONTAIN, addr) ==
600              0) {
601              err = as_map(as, base, len, segvn_create, zfod_argsp);
602              as_rangeunlock(as);
603              if (err) {
604                  res = FC_MAKE_ERR(err);
605                  goto out;
606              }
607          } else {
608              /*
609               * This page is already mapped by another thread after
610               * we returned from as_fault() above. We just fall
611               * through as_fault() below.
612               */
613              as_rangeunlock(as);
614          }

616          res = as_fault(hat, as, addr, 1, F_INVALID, rw);
617      }

619  out:
620      if (mapped_red)
621          segkp_unmap_red();

623      return (res);
624  }

626  void
627  map_addr(caddr_t *addrp, size_t len, offset_t off, int vacalign, uint_t flags)
628  {
629      struct proc *p = curproc;
630      caddr_t userlimit = (flags & _MAP_LOW32) ?
631          (caddr_t)_userlimit32 : p->p_as->a_userlimit;

633      map_addr_proc(addrp, len, off, vacalign, userlimit, curproc, flags);
634  }

636  /*ARGSUSED*/
637  int
638  map_addr_vacalign_check(caddr_t addr, u_offset_t off)
639  {
640      return (0);
641  }

643  /*
644   * The maximum amount a randomized mapping will be slewed. We should perhaps
645   * arrange things so these tunables can be separate for mmap, mmapobj, and
646   * ld.so
647   */
648  size_t aslr_max_map_skew = 256 * 1024 * 1024; /* 256MB */

650  /*
651  #endif /* ! codereview */
652  * map_addr_proc() is the routine called when the system is to

```

```

653 * choose an address for the user. We will pick an address
654 * range which is the highest available below userlimit.
655 *
656 * Every mapping will have a redzone of a single page on either side of
657 * the request. This is done to leave one page unmapped between segments.
658 * This is not required, but it's useful for the user because if their
659 * program strays across a segment boundary, it will catch a fault
660 * immediately making debugging a little easier. Currently the redzone
661 * is mandatory.
662 *
663 * addrp is a value/result parameter.
664 * On input it is a hint from the user to be used in a completely
665 * machine dependent fashion. We decide to completely ignore this hint.
666 * If MAP_ALIGN was specified, addrp contains the minimal alignment, which
667 * must be some "power of two" multiple of pagesize.
668 *
669 * On output it is NULL if no address can be found in the current
670 * processes address space or else an address that is currently
671 * not mapped for len bytes with a page of red zone on either side.
672 *
673 * vacalign is not needed on x86 (it's for viturally addressed caches)
674 */
675 /*ARGSUSED*/
676 void
677 map_addr_proc(
678     caddr_t *addrp,
679     size_t len,
680     offset_t off,
681     int vacalign,
682     caddr_t userlimit,
683     struct proc *p,
684     uint_t flags)
685 {
686     struct as *as = p->p_as;
687     caddr_t addr;
688     caddr_t base;
689     size_t slen;
690     size_t align_amount;
691
692     ASSERT32(userlimit == as->a_userlimit);
693
694     base = p->p_brkbase;
695 #if defined(__amd64)
696     /*
697      * XX64 Yes, this needs more work.
698      */
699     if (p->p_model == DATAMODEL_NATIVE) {
700         if (userlimit < as->a_userlimit) {
701             /*
702              * This happens when a program wants to map
703              * something in a range that's accessible to a
704              * program in a smaller address space. For example,
705              * a 64-bit program calling mmap32(2) to guarantee
706              * that the returned address is below 4Gbytes.
707              */
708             ASSERT((uintptr_t)userlimit < ADDRESS_C(0xffffffff));
709
710             if (userlimit > base)
711                 slen = userlimit - base;
712             else {
713                 *addrp = NULL;
714                 return;
715             }
716         } else {
717             /*
718              * XX64 This layout is probably wrong .. but in

```

```

719         * the event we make the amd64 address space look
720         * like sparcv9 i.e. with the stack -above- the
721         * heap, this bit of code might even be correct.
722         */
723         slen = p->p_usrstack - base -
724             ((p->p_stk_ctl + PAGEOFFSET) & PAGEMASK);
725     }
726 } else
727 #endif
728     slen = userlimit - base;
729
730 /* Make len be a multiple of PAGE_SIZE */
731 len = (len + PAGEOFFSET) & PAGEMASK;
732
733 /*
734  * figure out what the alignment should be
735  *
736  * XX64 -- is there an ELF_AMD64_MAXPGSZ or is it the same????
737  */
738 if (len <= ELF_386_MAXPGSZ) {
739     /*
740      * Align virtual addresses to ensure that ELF shared libraries
741      * are mapped with the appropriate alignment constraints by
742      * the run-time linker.
743      */
744     align_amount = ELF_386_MAXPGSZ;
745 } else {
746     /*
747      * For 32-bit processes, only those which have specified
748      * MAP_ALIGN and an addr will be aligned on a larger page size.
749      * Not doing so can potentially waste up to 1G of process
750      * address space.
751      */
752     int lvl = (p->p_model == DATAMODEL_ILP32) ? 1 :
753             mmu.umax_page_level;
754
755     while (lvl && len < LEVEL_SIZE(lvl))
756         --lvl;
757
758     align_amount = LEVEL_SIZE(lvl);
759 }
760 if ((flags & MAP_ALIGN) && ((uintptr_t)*addrp > align_amount))
761     align_amount = (uintptr_t)*addrp;
762
763 ASSERT(ISP2(align_amount));
764 ASSERT(align_amount == 0 || align_amount >= PAGE_SIZE);
765
766 off = off & (align_amount - 1);
767
768 #endif /* ! codereview */
769 /*
770  * Look for a large enough hole starting below userlimit.
771  * After finding it, use the upper part.
772  */
773 if (as_gap_aligned(as, len, &base, &slen, AH_HI, NULL, align_amount,
774     PAGE_SIZE, off) == 0) {
775     caddr_t as_addr;
776
777     /*
778      * addr is the highest possible address to use since we have
779      * a PAGE_SIZE redzone at the beginning and end.
780      */
781     addr = base + slen - (PAGE_SIZE + len);
782     as_addr = addr;
783     /*
784      * Round address DOWN to the alignment amount and

```

```

785     * add the offset in.
786     * If addr is greater than as_addr, len would not be large
787     * enough to include the redzone, so we must adjust down
788     * by the alignment amount.
789     */
790     addr = (caddr_t)((uintptr_t)addr & ~(align_amount - 1));
791     addr += (uintptr_t)off;
792     if (addr > as_addr) {
793         addr -= align_amount;
794     }
795
796     /*
797     * If randomization is requested, slew the allocation
798     * backwards, within the same gap, by a random amount.
799     */
800     if (flags & _MAP_RANDOMIZE) {
801         uint32_t slew;
802
803         (void) random_get_pseudo_bytes((uint8_t *)&slew,
804             sizeof (slew));
805
806         slew = slew % MIN(aslr_max_map_skew, (addr - base));
807         addr -= P2ALIGN(slew, align_amount);
808     }
809
810 #endif /* ! codereview */
811     ASSERT(addr > base);
812     ASSERT(addr + len < base + slen);
813     ASSERT(((uintptr_t)addr & (align_amount - 1)) ==
814         ((uintptr_t)(off)));
815     *addrp = addr;
816 } else {
817     *addrp = NULL; /* no more virtual space */
818 }
819 }
820
821 int valid_va_range_aligned_wraparound;
822
823 /*
824 * Determine whether [*basep, *basep + *lenp) contains a mappable range of
825 * addresses at least "minlen" long, where the base of the range is at "off"
826 * phase from an "align" boundary and there is space for a "redzone"-sized
827 * redzone on either side of the range. On success, 1 is returned and *basep
828 * and *lenp are adjusted to describe the acceptable range (including
829 * the redzone). On failure, 0 is returned.
830 */
831 /* ARGSUSED3 */
832 int
833 valid_va_range_aligned(caddr_t *basep, size_t *lenp, size_t minlen, int dir,
834     size_t align, size_t redzone, size_t off)
835 {
836     uintptr_t hi, lo;
837     size_t tot_len;
838
839     ASSERT(align == 0 ? off == 0 : off < align);
840     ASSERT(ISP2(align));
841     ASSERT(align == 0 || align >= PAGE_SIZE);
842
843     lo = (uintptr_t)*basep;
844     hi = lo + *lenp;
845     tot_len = minlen + 2 * redzone; /* need at least this much space */
846
847     /*
848     * If hi rolled over the top, try cutting back.
849     */
850     if (hi < lo) {

```

```

851         *lenp = 0UL - lo - 1UL;
852         /* See if this really happens. If so, then we figure out why */
853         valid_va_range_aligned_wraparound++;
854         hi = lo + *lenp;
855     }
856     if (*lenp < tot_len) {
857         return (0);
858     }
859
860 #if defined(__amd64)
861     /*
862     * Deal with a possible hole in the address range between
863     * hole_start and hole_end that should never be mapped.
864     */
865     if (lo < hole_start) {
866         if (hi > hole_start) {
867             if (hi < hole_end) {
868                 hi = hole_start;
869             } else {
870                 /* lo < hole_start && hi >= hole_end */
871                 if (dir == AH_LO) {
872                     /*
873                     * prefer lowest range
874                     */
875                     if (hole_start - lo >= tot_len)
876                         hi = hole_start;
877                     else if (hi - hole_end >= tot_len)
878                         lo = hole_end;
879                     else
880                         return (0);
881                 } else {
882                     /*
883                     * prefer highest range
884                     */
885                     if (hi - hole_end >= tot_len)
886                         lo = hole_end;
887                     else if (hole_start - lo >= tot_len)
888                         hi = hole_start;
889                     else
890                         return (0);
891                 }
892             }
893         }
894     } else {
895         /* lo >= hole_start */
896         if (hi < hole_end)
897             return (0);
898         if (lo < hole_end)
899             lo = hole_end;
900     }
901 #endif
902
903     if (hi - lo < tot_len)
904         return (0);
905
906     if (align > 1) {
907         uintptr_t tlo = lo + redzone;
908         uintptr_t thi = hi - redzone;
909         tlo = (uintptr_t)P2PHASEUP(tlo, align, off);
910         if (tlo < lo + redzone) {
911             return (0);
912         }
913         if (thi < tlo || thi - tlo < minlen) {
914             return (0);
915         }
916     }

```

```

918     *basep = (caddr_t)lo;
919     *lenp = hi - lo;
920     return (1);
921 }

923 /*
924  * Determine whether [*basep, *basep + *lenp) contains a mappable range of
925  * addresses at least "minlen" long. On success, 1 is returned and *basep
926  * and *lenp are adjusted to describe the acceptable range. On failure, 0
927  * is returned.
928  */
929 int
930 valid_va_range(caddr_t *basep, size_t *lenp, size_t minlen, int dir)
931 {
932     return (valid_va_range_aligned(basep, lenp, minlen, dir, 0, 0, 0));
933 }

935 /*
936  * Default to forbidding the first 64k of address space. This protects most
937  * reasonably sized structures from dereferences through NULL:
938  * ((foo_t *)0)->bar
939  */
940 uintptr_t forbidden_null_mapping_sz = 0x10000;

942 /*
943 #endif /* ! codereview */
944 * Determine whether [addr, addr+len] are valid user addresses.
945 */
946 /*ARGSUSED*/
947 int
948 valid_usr_range(caddr_t addr, size_t len, uint_t prot, struct as *as,
949     caddr_t userlimit)
950 {
951     caddr_t eaddr = addr + len;

953     if (eaddr <= addr || addr >= userlimit || eaddr > userlimit)
954         return (RANGE_BADADDR);

956     if ((addr <= (caddr_t)forbidden_null_mapping_sz) &&
957         secflag_enabled(as->a_proc, PROC_SEC_FORBIDNULLMAP))
958         return (RANGE_BADADDR);

960 #endif /* ! codereview */
961 #if defined(__amd64)
962     /*
963      * Check for the VA hole
964      */
965     if (eaddr > (caddr_t)hole_start && addr < (caddr_t)hole_end)
966         return (RANGE_BADADDR);
967 #endif

969     return (RANGE_OKAY);
970 }

972 /*
973  * Return 1 if the page frame is onboard memory, else 0.
974  */
975 int
976 pf_is_memory(pfn_t pf)
977 {
978     if (pfn_is_foreign(pf))
979         return (0);
980     return (address_in_memlist(phys_install, pfn_to_pa(pf), 1));
981 }

```

```

983 /*
984  * return the memrange containing pfn
985  */
986 int
987 memrange_num(pfn_t pfn)
988 {
989     int n;

991     for (n = 0; n < nranges - 1; ++n) {
992         if (pfn >= memranges[n])
993             break;
994     }
995     return (n);
996 }

998 /*
999  * return the mnoderange containing pfn
1000  */
1001 /*ARGSUSED*/
1002 int
1003 pfn_2_mtype(pfn_t pfn)
1004 {
1005     #if defined(__xpv)
1006         return (0);
1007     #else
1008         int n;

1010         /* Always start from highest pfn and work our way down */
1011         for (n = mtypetop; n != -1; n = mnoderanges[n].mnr_next) {
1012             if (pfn >= mnoderanges[n].mnr_pfnlo) {
1013                 break;
1014             }
1015         }
1016         return (n);
1017     #endif
1018 }

1020 #if !defined(__xpv)
1021 /*
1022  * is_contigpage_free:
1023  * returns a page list of contiguous pages. It minimally has to return
1024  * minctg pages. Caller determines minctg based on the scatter-gather
1025  * list length.
1026  *
1027  * pfnp is set to the next page frame to search on return.
1028  */
1029 static page_t *
1030 is_contigpage_free(
1031     pfn_t *pfnp,
1032     pgcnt_t *pgcnt,
1033     pgcnt_t minctg,
1034     uint64_t pfnseg,
1035     int iolock)
1036 {
1037     int i = 0;
1038     pfn_t pfn = *pfnp;
1039     page_t *pp;
1040     page_t *plist = NULL;

1042     /*
1043      * fail if pfn + minctg crosses a segment boundary.
1044      * Adjust for next starting pfn to begin at segment boundary.
1045      */

1047     if (((*pfnp + minctg - 1) & pfnseg) < (*pfnp & pfnseg)) {
1048         *pfnp = roundup(*pfnp, pfnseg + 1);

```

```

1049         return (NULL);
1050     }
1052     do {
1053 retry:
1054         pp = page_numtopp_nolock(pfn + i);
1055         if ((pp == NULL) || IS_DUMP_PAGE(pp) ||
1056             (page_trylock(pp, SE_EXCL) == 0)) {
1057             (*pfnp)++;
1058             break;
1059         }
1060         if (page_pptonum(pp) != pfn + i) {
1061             page_unlock(pp);
1062             goto retry;
1063         }
1065         if (!PP_ISFREE(pp)) {
1066             page_unlock(pp);
1067             (*pfnp)++;
1068             break;
1069         }
1071         if (!PP_ISAGED(pp)) {
1072             page_list_sub(pp, PG_CACHE_LIST);
1073             page_hashout(pp, (kmutex_t *)NULL);
1074         } else {
1075             page_list_sub(pp, PG_FREE_LIST);
1076         }
1078         if (iolock)
1079             page_io_lock(pp);
1080         page_list_concat(&plist, &pp);
1082         /*
1083          * exit loop when pgcnt satisfied or segment boundary reached.
1084          */
1086     } while ((++i < *pgcnt) && ((pfn + i) & pfnsegt));
1088     *pfnp += i;          /* set to next pfn to search */
1090     if (i >= minctg) {
1091         *pgcnt -= i;
1092         return (plist);
1093     }
1095     /*
1096     * failure: minctg not satisfied.
1097     *
1098     * if next request crosses segment boundary, set next pfn
1099     * to search from the segment boundary.
1100     */
1101     if (((*pfnp + minctg - 1) & pfnsegt) < (*pfnp & pfnsegt))
1102         *pfnp = roundup(*pfnp, pfnsegt + 1);
1104     /* clean up any pages already allocated */
1106     while (plist) {
1107         pp = plist;
1108         page_sub(&plist, pp);
1109         page_list_add(pp, PG_FREE_LIST | PG_LIST_TAIL);
1110         if (iolock)
1111             page_io_unlock(pp);
1112         page_unlock(pp);
1113     }

```

```

1115         return (NULL);
1116     }
1117 #endif /* !_xpv */
1119 /*
1120  * verify that pages being returned from allocator have correct DMA attribute
1121  */
1122 #ifndef DEBUG
1123 #define check_dma(a, b, c) (void)(0)
1124 #else
1125 static void
1126 check_dma(ddi_dma_attr_t *dma_attr, page_t *pp, int cnt)
1127 {
1128     if (dma_attr == NULL)
1129         return;
1131     while (cnt-- > 0) {
1132         if (pa_to_ma(pfn_to_pa(pp->p_pagenum)) <
1133             dma_attr->dma_attr_addr_lo)
1134             panic("PFN (pp=%p) below dma_attr_addr_lo", (void *)pp);
1135         if (pa_to_ma(pfn_to_pa(pp->p_pagenum)) >=
1136             dma_attr->dma_attr_addr_hi)
1137             panic("PFN (pp=%p) above dma_attr_addr_hi", (void *)pp);
1138         pp = pp->p_next;
1139     }
1140 }
1141 #endif
1143 #if defined(_xpv)
1144 static page_t *
1145 page_get_contigpage(pgcnt_t *pgcnt, ddi_dma_attr_t *matr, int iolock)
1146 {
1147     pfn_t         pfn;
1148     int           sgllen;
1149     uint64_t      pfnsegt;
1150     pgcnt_t       minctg;
1151     page_t        *plist = NULL, *p;
1152     uint64_t       lo, hi;
1153     pgcnt_t       pfnalign = 0;
1154     static pfn_t  startpfn;
1155     static pgcnt_t lastctgcnt;
1156     uintptr_t      align;
1158     CONTIG_LOCK();
1160     if (matr) {
1161         lo = mmu_btop((matr->dma_attr_addr_lo + MMU_PAGEOFFSET));
1162         hi = mmu_btop(matr->dma_attr_addr_hi);
1163         if (hi >= physmax)
1164             hi = physmax - 1;
1165         sgllen = matr->dma_attr_sgllen;
1166         pfnsegt = mmu_btop(matr->dma_attr_seg);
1168         align = maxbit(matr->dma_attr_align, matr->dma_attr_minxfer);
1169         if (align > MMU_PAGESIZE)
1170             pfnalign = mmu_btop(align);
1172         /*
1173          * in order to satisfy the request, must minimally
1174          * acquire minctg contiguous pages
1175          */
1176         minctg = howmany(*pgcnt, sgllen);
1178         ASSERT(hi >= lo);
1180         /*

```



```

1181         * start from where last searched if the minctg >= lastctgcnt
1182         */
1183         if (minctg < lastctgcnt || startpfn < lo || startpfn > hi)
1184             startpfn = lo;
1185     } else {
1186         hi = physmax - 1;
1187         lo = 0;
1188         sgllen = 1;
1189         pfnseg = mmu.highest_pfn;
1190         minctg = *pgcnt;
1191
1192         if (minctg < lastctgcnt)
1193             startpfn = lo;
1194     }
1195     lastctgcnt = minctg;
1196
1197     ASSERT(pfnseg + 1 >= (uint64_t)minctg);
1198
1199     /* conserve 16m memory - start search above 16m when possible */
1200     if (hi > PFN_16M && startpfn < PFN_16M)
1201         startpfn = PFN_16M;
1202
1203     pfn = startpfn;
1204     if (pfnalign)
1205         pfn = P2ROUNDUP(pfn, pfnalign);
1206
1207     while (pfn + minctg - 1 <= hi) {
1208
1209         plist = is_contigpage_free(&pfn, pgcnt, minctg, pfnseg, iolock);
1210         if (plist) {
1211             page_list_concat(&pplist, &plist);
1212             sgllen--;
1213             /*
1214              * return when contig pages no longer needed
1215              */
1216             if (!*pgcnt || ((*pgcnt <= sgllen) && !pfnalign)) {
1217                 startpfn = pfn;
1218                 CONTIG_UNLOCK();
1219                 check_dma(matr, pplist, *pgcnt);
1220                 return (pplist);
1221             }
1222             minctg = howmany(*pgcnt, sgllen);
1223         }
1224         if (pfnalign)
1225             pfn = P2ROUNDUP(pfn, pfnalign);
1226     }
1227
1228     /* cannot find contig pages in specified range */
1229     if (startpfn == lo) {
1230         CONTIG_UNLOCK();
1231         return (NULL);
1232     }
1233
1234     /* did not start with lo previously */
1235     pfn = lo;
1236     if (pfnalign)
1237         pfn = P2ROUNDUP(pfn, pfnalign);
1238
1239     /* allow search to go above startpfn */
1240     while (pfn < startpfn) {
1241
1242         plist = is_contigpage_free(&pfn, pgcnt, minctg, pfnseg, iolock);
1243         if (plist != NULL) {
1244             page_list_concat(&pplist, &plist);
1245             sgllen--;

```

```

1248         /*
1249          * return when contig pages no longer needed
1250          */
1251         if (!*pgcnt || ((*pgcnt <= sgllen) && !pfnalign)) {
1252             startpfn = pfn;
1253             CONTIG_UNLOCK();
1254             check_dma(matr, pplist, *pgcnt);
1255             return (pplist);
1256         }
1257         minctg = howmany(*pgcnt, sgllen);
1258     }
1259     if (pfnalign)
1260         pfn = P2ROUNDUP(pfn, pfnalign);
1261     }
1262     CONTIG_UNLOCK();
1263     return (NULL);
1264 }
1265 #endif /* !__xpv */
1266
1267 /*
1268 * mnode_range_cnt() calculates the number of memory ranges for mnode and
1269 * memranges[]. Used to determine the size of page lists and mnoderanges.
1270 */
1271 int
1272 mnode_range_cnt(int mnode)
1273 {
1274     #if defined(__xpv)
1275         ASSERT(mnode == 0);
1276         return (1);
1277     #else /* __xpv */
1278         int mri;
1279         int mnrCnt = 0;
1280
1281         if (mem_node_config[mnode].exists != 0) {
1282             mri = nranges - 1;
1283
1284             /* find the memranges index below contained in mnode range */
1285
1286             while (MEMRANGEHI(mri) < mem_node_config[mnode].physbase)
1287                 mri--;
1288
1289             /*
1290              * increment mnode range counter when memranges or mnode
1291              * boundary is reached.
1292              */
1293             while (mri >= 0 &&
1294                 mem_node_config[mnode].physmax >= MEMRANGELO(mri)) {
1295                 mnrCnt++;
1296                 if (mem_node_config[mnode].physmax > MEMRANGEHI(mri))
1297                     mri--;
1298                 else
1299                     break;
1300             }
1301         }
1302         ASSERT(mnrCnt <= MAX_MNODE_MRANGES);
1303         return (mnrCnt);
1304     #endif /* __xpv */
1305 }
1306
1307 /*
1308 * mnode_range_setup() initializes mnoderanges.
1309 */
1310 void
1311 mnode_range_setup(mnoderange_t *mnoderanges)
1312 {

```

```

1313 mnode_range_t *mp = mnoderanges;
1314 int mnode, mri;
1315 int mindex = 0; /* current index into mnoderanges array */
1316 int i, j;
1317 pfn_t hipfn;
1318 int last, hi;

1320 for (mnode = 0; mnode < max_mem_nodes; mnode++) {
1321     if (mem_node_config[mnode].exists == 0)
1322         continue;

1324     mri = nranges - 1;

1326     while (MEMRANGEHI(mri) < mem_node_config[mnode].physbase)
1327         mri--;

1329     while (mri >= 0 && mem_node_config[mnode].physmax >=
1330             MEMRANGELO(mri)) {
1331         mnoderanges->mnr_pfnlo = MAX(MEMRANGELO(mri),
1332             mem_node_config[mnode].physbase);
1333         mnoderanges->mnr_pfnhi = MIN(MEMRANGEHI(mri),
1334             mem_node_config[mnode].physmax);
1335         mnoderanges->mnr_mnode = mnode;
1336         mnoderanges->mnr_memrange = mri;
1337         mnoderanges->mnr_exists = 1;
1338         mnoderanges++;
1339         mindex++;
1340         if (mem_node_config[mnode].physmax > MEMRANGEHI(mri))
1341             mri--;
1342         else
1343             break;
1344     }
1345 }

1347 /*
1348  * For now do a simple sort of the mnoderanges array to fill in
1349  * the mnr_next fields. Since mindex is expected to be relatively
1350  * small, using a simple O(N^2) algorithm.
1351  */
1352 for (i = 0; i < mindex; i++) {
1353     if (mp[i].mnr_pfnlo == 0) /* find lowest */
1354         break;
1355 }
1356 ASSERT(i < mindex);
1357 last = i;
1358 mtypel6m = last;
1359 mp[last].mnr_next = -1;
1360 for (i = 0; i < mindex - 1; i++) {
1361     hipfn = (pfn_t)(-1);
1362     hi = -1;
1363     /* find next highest mnode range */
1364     for (j = 0; j < mindex; j++) {
1365         if (mp[j].mnr_pfnlo > mp[last].mnr_pfnlo &&
1366             mp[j].mnr_pfnlo < hipfn) {
1367             hipfn = mp[j].mnr_pfnlo;
1368             hi = j;
1369         }
1370     }
1371     mp[hi].mnr_next = last;
1372     last = hi;
1373 }
1374 mtypetop = last;
1375 }

1377 #ifndef __xpv
1378 /*

```

```

1379  * Update mnoderanges for memory hot-add DR operations.
1380  */
1381 static void
1382 mnode_range_add(int mnode)
1383 {
1384     int *prev;
1385     int n, mri;
1386     pfn_t start, end;
1387     extern void membar_sync(void);

1389     ASSERT(0 <= mnode && mnode < max_mem_nodes);
1390     ASSERT(mem_node_config[mnode].exists);
1391     start = mem_node_config[mnode].physbase;
1392     end = mem_node_config[mnode].physmax;
1393     ASSERT(start <= end);
1394     mutex_enter(&mnoderange_lock);

1396 #ifdef DEBUG
1397     /* Check whether it interleaves with other memory nodes. */
1398     for (n = mtypetop; n != -1; n = mnoderanges[n].mnr_next) {
1399         ASSERT(mnoderanges[n].mnr_exists);
1400         if (mnoderanges[n].mnr_mnode == mnode)
1401             continue;
1402         ASSERT(start > mnoderanges[n].mnr_pfnhi ||
1403             end < mnoderanges[n].mnr_pfnlo);
1404     }
1405 #endif /* DEBUG */

1407     mri = nranges - 1;
1408     while (MEMRANGEHI(mri) < mem_node_config[mnode].physbase)
1409         mri--;
1410     while (mri >= 0 && mem_node_config[mnode].physmax >= MEMRANGELO(mri)) {
1411         /* Check whether mtype already exists. */
1412         for (n = mtypetop; n != -1; n = mnoderanges[n].mnr_next) {
1413             if (mnoderanges[n].mnr_mnode == mnode &&
1414                 mnoderanges[n].mnr_memrange == mri) {
1415                 mnoderanges[n].mnr_pfnlo = MAX(MEMRANGELO(mri),
1416                     start);
1417                 mnoderanges[n].mnr_pfnhi = MIN(MEMRANGEHI(mri),
1418                     end);
1419                 break;
1420             }
1421         }

1423         /* Add a new entry if it doesn't exist yet. */
1424         if (n == -1) {
1425             /* Try to find an unused entry in mnoderanges array. */
1426             for (n = 0; n < mnoderangecnt; n++) {
1427                 if (mnoderanges[n].mnr_exists == 0)
1428                     break;
1429             }
1430             ASSERT(n < mnoderangecnt);
1431             mnoderanges[n].mnr_pfnlo = MAX(MEMRANGELO(mri), start);
1432             mnoderanges[n].mnr_pfnhi = MIN(MEMRANGEHI(mri), end);
1433             mnoderanges[n].mnr_mnode = mnode;
1434             mnoderanges[n].mnr_memrange = mri;
1435             mnoderanges[n].mnr_exists = 1;
1436             /* Page 0 should always be present. */
1437             for (prev = &mtypetop;
1438                 mnoderanges[*prev].mnr_pfnlo > start;
1439                 prev = &mnoderanges[*prev].mnr_next) {
1440                 ASSERT(mnoderanges[*prev].mnr_next >= 0);
1441                 ASSERT(mnoderanges[*prev].mnr_pfnlo > end);
1442             }
1443             mnoderanges[n].mnr_next = *prev;
1444             membar_sync();

```

```

1445         *prev = n;
1446     }
1448     if (mem_node_config[mnode].physmax > MEMRANGEHI(mri))
1449         mri--;
1450     else
1451         break;
1452 }
1454     mutex_exit(&mnode_range_lock);
1455 }
1457 /*
1458  * Update mnode ranges for memory hot-removal DR operations.
1459  */
1460 static void
1461 mnode_range_del(int mnode)
1462 {
1463     _NOTE(ARGUNUSED(mnode));
1464     ASSERT(0 <= mnode && mnode < max_mem_nodes);
1465     /* TODO: support deletion operation. */
1466     ASSERT(0);
1467 }
1469 void
1470 plat_slice_add(pfn_t start, pfn_t end)
1471 {
1472     mem_node_add_slice(start, end);
1473     if (plat_dr_enabled()) {
1474         mnode_range_add(PFN_2_MEM_NODE(start));
1475     }
1476 }
1478 void
1479 plat_slice_del(pfn_t start, pfn_t end)
1480 {
1481     ASSERT(PFN_2_MEM_NODE(start) == PFN_2_MEM_NODE(end));
1482     ASSERT(plat_dr_enabled());
1483     mnode_range_del(PFN_2_MEM_NODE(start));
1484     mem_node_del_slice(start, end);
1485 }
1486 #endif /* __xpv */
1488 /*ARGSUSED*/
1489 int
1490 mtype_init(vnode_t *vp, caddr_t vaddr, uint_t *flags, size_t pgsz)
1491 {
1492     int mtype = mtypetop;
1494 #if !defined(__xpv)
1495 #if defined(__i386)
1496     /*
1497      * set the mtype range
1498      * - kmem requests need to be below 4g if restricted_kmemalloc is set.
1499      * - for non kmem requests, set range to above 4g if memory below 4g
1500      * runs low.
1501      */
1502     if (restricted_kmemalloc && VN_ISKAS(vp) &&
1503         (caddr_t)vaddr >= kernelheap &&
1504         (caddr_t)vaddr < ekernelheap) {
1505         ASSERT(physmax4g);
1506         mtype = mtype4g;
1507         if (RESTRICT16M_ALLOC(freemem4g - btop(pgsz),
1508             btop(pgsz), *flags)) {
1509             *flags |= PGI_MT_RANGE16M;
1510         } else {

```

```

1511         VM_STAT_ADD(vmm_vmstats.unrestrict16mcnt);
1512         VM_STAT_COND_ADD((*flags & PG_PANIC),
1513             vmm_vmstats.pgpanicalloc);
1514         *flags |= PGI_MT_RANGE0;
1515     }
1516     return (mtype);
1517 }
1518 #endif /* __i386 */
1520     if (RESTRICT4G_ALLOC) {
1521         VM_STAT_ADD(vmm_vmstats.restrict4gcnt);
1522         /* here only for > 4g systems */
1523         *flags |= PGI_MT_RANGE4G;
1524     } else if (RESTRICT16M_ALLOC(freemem, btop(pgsz), *flags)) {
1525         *flags |= PGI_MT_RANGE16M;
1526     } else {
1527         VM_STAT_ADD(vmm_vmstats.unrestrict16mcnt);
1528         VM_STAT_COND_ADD((*flags & PG_PANIC), vmm_vmstats.pgpanicalloc);
1529         *flags |= PGI_MT_RANGE0;
1530     }
1531 #endif /* !__xpv */
1532     return (mtype);
1533 }
1536 /* mtype init for page_get_replacement_page */
1537 /*ARGSUSED*/
1538 int
1539 mtype_pgr_init(int *flags, page_t *pp, int mnode, pgcnt_t pgcnt)
1540 {
1541     int mtype = mtypetop;
1542 #if !defined(__xpv)
1543     if (RESTRICT16M_ALLOC(freemem, pgcnt, *flags)) {
1544         *flags |= PGI_MT_RANGE16M;
1545     } else {
1546         VM_STAT_ADD(vmm_vmstats.unrestrict16mcnt);
1547         *flags |= PGI_MT_RANGE0;
1548     }
1549 #endif
1550     return (mtype);
1551 }
1553 /*
1554  * Determine if the mnode range specified in mtype contains memory belonging
1555  * to memory node mnode. If flags & PGI_MT_RANGE is set then mtype contains
1556  * the range from high pfn to 0, 16m or 4g.
1557  *
1558  * Return first mnode range type index found otherwise return -1 if none found.
1559  */
1560 int
1561 mtype_func(int mnode, int mtype, uint_t flags)
1562 {
1563     if (flags & PGI_MT_RANGE) {
1564         int mnr_lim = MRI_0;
1566         if (flags & PGI_MT_NEXT) {
1567             mtype = mnode_ranges[mtype].mnr_next;
1568         }
1569         if (flags & PGI_MT_RANGE4G)
1570             mnr_lim = MRI_4G; /* exclude 0-4g range */
1571         else if (flags & PGI_MT_RANGE16M)
1572             mnr_lim = MRI_16M; /* exclude 0-16m range */
1573         while (mtype != -1 &&
1574             mnode_ranges[mtype].mnr_memrange <= mnr_lim) {
1575             if (mnode_ranges[mtype].mnr_mnode == mnode)
1576                 return (mtype);

```

```

1577         mtype = mnoderanges[mtype].mnr_next;
1578     }
1579     } else if (mnoderanges[mtype].mnr_mnode == mnode) {
1580         return (mtype);
1581     }
1582     return (-1);
1583 }

1585 /*
1586  * Update the page list max counts with the pfn range specified by the
1587  * input parameters.
1588  */
1589 void
1590 mtype_modify_max(pfn_t startpfn, long cnt)
1591 {
1592     int             mtype;
1593     pgcnt_t        inc;
1594     spgcnt_t       scnt = (spgcnt_t)(cnt);
1595     pgcnt_t        acnt = ABS(scnt);
1596     pfn_t          endpfn = startpfn + acnt;
1597     pfn_t          pfn, lo;

1599     if (!physmax4g)
1600         return;

1602     mtype = mtypetop;
1603     for (pfn = endpfn; pfn > startpfn; ) {
1604         ASSERT(mtype != -1);
1605         lo = mnoderanges[mtype].mnr_pfnlo;
1606         if (pfn > lo) {
1607             if (startpfn >= lo) {
1608                 inc = pfn - startpfn;
1609             } else {
1610                 inc = pfn - lo;
1611             }
1612             if (mnoderanges[mtype].mnr_memrange != MRI_4G) {
1613                 if (scnt > 0)
1614                     maxmem4g += inc;
1615                 else
1616                     maxmem4g -= inc;
1617             }
1618             pfn -= inc;
1619         }
1620         mtype = mnoderanges[mtype].mnr_next;
1621     }
1622 }

1624 int
1625 mtype_2_mrange(int mtype)
1626 {
1627     return (mnoderanges[mtype].mnr_memrange);
1628 }

1630 void
1631 mnodetype_2_pfn(int mnode, int mtype, pfn_t *pfnlo, pfn_t *pfnhi)
1632 {
1633     _NOTE(ARGUNUSED(mnode));
1634     ASSERT(mnoderanges[mtype].mnr_mnode == mnode);
1635     *pfnlo = mnoderanges[mtype].mnr_pfnlo;
1636     *pfnhi = mnoderanges[mtype].mnr_pfnhi;
1637 }

1639 size_t
1640 plcnt_sz(size_t ctrs_sz)
1641 {
1642 #ifdef DEBUG

```

```

1643     int             szc, colors;

1645     ctrs_sz += mnoderangecnt * sizeof (struct mnr_mts) * mmu_page_sizes;
1646     for (szc = 0; szc < mmu_page_sizes; szc++) {
1647         colors = page_get_pagecolors(szc);
1648         ctrs_sz += mnoderangecnt * sizeof (pgcnt_t) * colors;
1649     }
1650 #endif
1651     return (ctrs_sz);
1652 }

1654 caddr_t
1655 plcnt_init(caddr_t addr)
1656 {
1657 #ifdef DEBUG
1658     int             mt, szc, colors;

1660     for (mt = 0; mt < mnoderangecnt; mt++) {
1661         mnoderanges[mt].mnr_mts = (struct mnr_mts *)addr;
1662         addr += (sizeof (struct mnr_mts) * mmu_page_sizes);
1663         for (szc = 0; szc < mmu_page_sizes; szc++) {
1664             colors = page_get_pagecolors(szc);
1665             mnoderanges[mt].mnr_mts[szc].mnr_mts_colors = colors;
1666             mnoderanges[mt].mnr_mts[szc].mnr_mts_color =
1667                 (pgcnt_t *)addr;
1668             addr += (sizeof (pgcnt_t) * colors);
1669         }
1670     }
1671 #endif
1672     return (addr);
1673 }

1675 void
1676 plcnt_inc_dec(page_t *pp, int mtype, int szc, long cnt, int flags)
1677 {
1678     _NOTE(ARGUNUSED(pp));
1679 #ifdef DEBUG
1680     int             bin = PP_2_BIN(pp);

1682     atomic_add_long(&mnoderanges[mtype].mnr_mts[szc].mnr_mts_pgcnt, cnt);
1683     atomic_add_long(&mnoderanges[mtype].mnr_mts[szc].mnr_mts_color[bin],
1684                    cnt);
1685 #endif
1686     ASSERT(mtype == PP_2_MTYPE(pp));
1687     if (physmax4g && mnoderanges[mtype].mnr_memrange != MRI_4G)
1688         atomic_add_long(&freemem4g, cnt);
1689     if (flags & PG_CACHE_LIST)
1690         atomic_add_long(&mnoderanges[mtype].mnr_mt_clpgcnt, cnt);
1691     else
1692         atomic_add_long(&mnoderanges[mtype].mnr_mt_flpgcnt[szc], cnt);
1693     atomic_add_long(&mnoderanges[mtype].mnr_mt_totcnt, cnt);
1694 }

1696 /*
1697  * Returns the free page count for mnode
1698  */
1699 int
1700 mnode_pgcnt(int mnode)
1701 {
1702     int             mtype = mtypetop;
1703     int             flags = PGI_MT_RANGE0;
1704     pgcnt_t         pgcnt = 0;

1706     mtype = mtype_func(mnode, mtype, flags);

1708     while (mtype != -1) {

```

```

1709         pgcnt += MTYPE_FREEMEM(mtype);
1710         mtype = mtype_func(mnode, mtype, flags | PGI_MT_NEXT);
1711     }
1712     return (pgcnt);
1713 }

1715 /*
1716  * Initialize page coloring variables based on the l2 cache parameters.
1717  * Calculate and return memory needed for page coloring data structures.
1718  */
1719 size_t
1720 page_coloring_init(uint_t l2_sz, int l2_linesz, int l2_assoc)
1721 {
1722     NOTE(ARGUNUSED(l2_linesz));
1723     size_t colorsz = 0;
1724     int i;
1725     int colors;

1727 #if defined(__xpv)
1728     /*
1729     * Hypervisor domains currently don't have any concept of NUMA.
1730     * Hence we'll act like there is only 1 memrange.
1731     */
1732     i = memrange_num(1);
1733 #else /* !__xpv */
1734     /*
1735     * Reduce the memory ranges lists if we don't have large amounts
1736     * of memory. This avoids searching known empty free lists.
1737     * To support memory DR operations, we need to keep memory ranges
1738     * for possible memory hot-add operations.
1739     */
1740     if (plat_dr_physmax > physmax)
1741         i = memrange_num(plat_dr_physmax);
1742     else
1743         i = memrange_num(physmax);
1744 #if defined(__i386)
1745     if (i > MRI_4G)
1746         restricted_kmemalloc = 0;
1747 #endif
1748     /* physmax greater than 4g */
1749     if (i == MRI_4G)
1750         physmax4g = 1;
1751 #endif /* !__xpv */
1752     memranges += i;
1753     nranges -= i;

1755     ASSERT(mmu_page_sizes <= MMU_PAGE_SIZES);

1757     ASSERT(ISP2(l2_linesz));
1758     ASSERT(l2_sz > MMU_PAGESIZE);

1760     /* l2_assoc is 0 for fully associative l2 cache */
1761     if (l2_assoc)
1762         l2_colors = MAX(1, l2_sz / (l2_assoc * MMU_PAGESIZE));
1763     else
1764         l2_colors = 1;

1766     ASSERT(ISP2(l2_colors));

1768     /* for scalability, configure at least PAGE_COLORS_MIN color bins */
1769     page_colors = MAX(l2_colors, PAGE_COLORS_MIN);

1771     /*
1772     * cpu_page_colors is non-zero when a page color may be spread across
1773     * multiple bins.
1774     */

```

```

1775     if (l2_colors < page_colors)
1776         cpu_page_colors = l2_colors;

1778     ASSERT(ISP2(page_colors));

1780     page_colors_mask = page_colors - 1;

1782     ASSERT(ISP2(CPUSETSIZE()));
1783     page_coloring_shift = lowbit(CPUSETSIZE());

1785     /* initialize number of colors per page size */
1786     for (i = 0; i <= mmu.max_page_level; i++) {
1787         hw_page_array[i].hp_size = LEVEL_SIZE(i);
1788         hw_page_array[i].hp_shift = LEVEL_SHIFT(i);
1789         hw_page_array[i].hp_pgcnt = LEVEL_SIZE(i) >> LEVEL_SHIFT(0);
1790         hw_page_array[i].hp_colors = (page_colors_mask >>
1791             (hw_page_array[i].hp_shift - hw_page_array[0].hp_shift))
1792             + 1;
1793         colorequivszc[i] = 0;
1794     }

1796     /*
1797     * The value of cpu_page_colors determines if additional color bins
1798     * need to be checked for a particular color in the page_get routines.
1799     */
1800     if (cpu_page_colors != 0) {

1802         int a = lowbit(page_colors) - lowbit(cpu_page_colors);
1803         ASSERT(a > 0);
1804         ASSERT(a < 16);

1806         for (i = 0; i <= mmu.max_page_level; i++) {
1807             if ((colors = hw_page_array[i].hp_colors) <= 1) {
1808                 colorequivszc[i] = 0;
1809                 continue;
1810             }
1811             while ((colors >> a) == 0)
1812                 a--;
1813             ASSERT(a >= 0);

1815             /* higher 4 bits encodes color equiv mask */
1816             colorequivszc[i] = (a << 4);
1817         }
1818     }

1820     /* factor in colorequiv to check additional 'equivalent' bins. */
1821     if (colorequiv > 1) {

1823         int a = lowbit(colorequiv) - 1;
1824         if (a > 15)
1825             a = 15;

1827         for (i = 0; i <= mmu.max_page_level; i++) {
1828             if ((colors = hw_page_array[i].hp_colors) <= 1) {
1829                 continue;
1830             }
1831             while ((colors >> a) == 0)
1832                 a--;
1833             if ((a << 4) > colorequivszc[i]) {
1834                 colorequivszc[i] = (a << 4);
1835             }
1836         }
1837     }

1839     /* size for mnoderanges */
1840     for (mnoderangecnt = 0, i = 0; i < max_mem_nodes; i++)

```

```

1841     mnoderangecnt += mnoderange_cnt(i);
1842     if (plat_dr_support_memory()) {
1843         /*
1844          * Reserve enough space for memory DR operations.
1845          * Two extra mnoderanges for possible fragmentations,
1846          * one for the 2G boundary and the other for the 4G boundary.
1847          * We don't expect a memory board crossing the 16M boundary
1848          * for memory hot-add operations on x86 platforms.
1849          */
1850         mnoderangecnt += 2 + max_mem_nodes - lgrp_plat_node_cnt;
1851     }
1852     colorsz = mnoderangecnt * sizeof (mnoderange_t);
1853
1854     /* size for fpc_mutex and cpc_mutex */
1855     colorsz += (2 * max_mem_nodes * sizeof (kmutex_t) * NPC_MUTEX);
1856
1857     /* size of page_freelists */
1858     colorsz += mnoderangecnt * sizeof (page_t ***);
1859     colorsz += mnoderangecnt * mmu_page_sizes * sizeof (page_t **);
1860
1861     for (i = 0; i < mmu_page_sizes; i++) {
1862         colors = page_get_pagecolors(i);
1863         colorsz += mnoderangecnt * colors * sizeof (page_t *);
1864     }
1865
1866     /* size of page_cachelists */
1867     colorsz += mnoderangecnt * sizeof (page_t **);
1868     colorsz += mnoderangecnt * page_colors * sizeof (page_t *);
1869
1870     return (colorsz);
1871 }
1872
1873 /*
1874  * Called once at startup to configure page_coloring data structures and
1875  * does the 1st page_free()/page_freelist_add().
1876  */
1877 void
1878 page_coloring_setup(caddr_t pcmemaddr)
1879 {
1880     int    i;
1881     int    j;
1882     int    k;
1883     caddr_t addr;
1884     int    colors;
1885
1886     /*
1887      * do page coloring setup
1888      */
1889     addr = pcmemaddr;
1890
1891     mnoderanges = (mnoderange_t *)addr;
1892     addr += (mnoderangecnt * sizeof (mnoderange_t));
1893
1894     mnoderange_setup(mnoderanges);
1895
1896     if (physmax4g)
1897         mtype4g = pfn_2_mtype(0xfffff);
1898
1899     for (k = 0; k < NPC_MUTEX; k++) {
1900         fpc_mutex[k] = (kmutex_t *)addr;
1901         addr += (max_mem_nodes * sizeof (kmutex_t));
1902     }
1903     for (k = 0; k < NPC_MUTEX; k++) {
1904         cpc_mutex[k] = (kmutex_t *)addr;
1905         addr += (max_mem_nodes * sizeof (kmutex_t));
1906     }

```

```

1907     page_freelists = (page_t ****)addr;
1908     addr += (mnoderangecnt * sizeof (page_t ***));
1909
1910     page_cachelists = (page_t ***)addr;
1911     addr += (mnoderangecnt * sizeof (page_t **));
1912
1913     for (i = 0; i < mnoderangecnt; i++) {
1914         page_freelists[i] = (page_t ****)addr;
1915         addr += (mmu_page_sizes * sizeof (page_t ***));
1916
1917         for (j = 0; j < mmu_page_sizes; j++) {
1918             colors = page_get_pagecolors(j);
1919             page_freelists[i][j] = (page_t **)addr;
1920             addr += (colors * sizeof (page_t *));
1921         }
1922         page_cachelists[i] = (page_t **)addr;
1923         addr += (page_colors * sizeof (page_t *));
1924     }
1925 }
1926
1927 #if defined(__xpv)
1928 /*
1929  * Give back 10% of the io_pool pages to the free list.
1930  * Don't shrink the pool below some absolute minimum.
1931  */
1932 static void
1933 page_io_pool_shrink()
1934 {
1935     int    retcnt;
1936     page_t *pp, *pp_first, *pp_last, **curpool;
1937     mfn_t mfn;
1938     int    bothpools = 0;
1939
1940     mutex_enter(&io_pool_lock);
1941     io_pool_shrink_attempts++; /* should be a kstat? */
1942     retcnt = io_pool_cnt / 10;
1943     if (io_pool_cnt - retcnt < io_pool_cnt_min)
1944         retcnt = io_pool_cnt - io_pool_cnt_min;
1945     if (retcnt <= 0)
1946         goto done;
1947     io_pool_shrinks++; /* should be a kstat? */
1948     curpool = &io_pool_4g;
1949 domore:
1950     /*
1951      * Loop through taking pages from the end of the list
1952      * (highest mfns) till amount to return reached.
1953      */
1954     for (pp = *curpool; pp && retcnt > 0; ) {
1955         pp_first = pp_last = pp->p_prev;
1956         if (pp_first == *curpool)
1957             break;
1958         retcnt--;
1959         io_pool_cnt--;
1960         page_io_pool_sub(curpool, pp_first, pp_last);
1961         if ((mfn = pfn_to_mfn(pp->p_pagenum)) < start_mfn)
1962             start_mfn = mfn;
1963         page_free(pp_first, 1);
1964         pp = *curpool;
1965     }
1966     if (retcnt != 0 && !bothpools) {
1967         /*
1968          * If not enough found in less constrained pool try the
1969          * more constrained one.
1970          */
1971         curpool = &io_pool_16m;
1972         bothpools = 1;

```

```

1973         goto domore;
1974     }
1975 done:
1976     mutex_exit(&io_pool_lock);
1977 }

1979 #endif /* __xpv */

1981 uint_t
1982 page_create_update_flags_x86(uint_t flags)
1983 {
1984 #if defined(__xpv)
1985     /*
1986      * Check this is an urgent allocation and free pages are depleted.
1987      */
1988     if (!(flags & PG_WAIT) && freemem < desfree)
1989         page_io_pool_shrink();
1990 #else /* !__xpv */
1991     /*
1992      * page_create_get_something may call this because 4g memory may be
1993      * depleted. Set flags to allow for relocation of base page below
1994      * 4g if necessary.
1995      */
1996     if (physmax4g)
1997         flags |= (PGI_PGCPSZC0 | PGI_PGCPHIPRI);
1998 #endif /* __xpv */
1999     return (flags);
2000 }

2002 /*ARGSUSED*/
2003 int
2004 bp_color(struct buf *bp)
2005 {
2006     return (0);
2007 }

2009 #if defined(__xpv)

2011 /*
2012  * Take pages out of an io_pool
2013  */
2014 static void
2015 page_io_pool_sub(page_t **poolp, page_t *pp_first, page_t *pp_last)
2016 {
2017     if (*poolp == pp_first) {
2018         *poolp = pp_last->p_next;
2019         if (*poolp == pp_first)
2020             *poolp = NULL;
2021     }
2022     pp_first->p_prev->p_next = pp_last->p_next;
2023     pp_last->p_next->p_prev = pp_first->p_prev;
2024     pp_first->p_prev = pp_last;
2025     pp_last->p_next = pp_first;
2026 }

2028 /*
2029  * Put a page on the io_pool list. The list is ordered by increasing MFN.
2030  */
2031 static void
2032 page_io_pool_add(page_t **poolp, page_t *pp)
2033 {
2034     page_t *look;
2035     mfn_t mfn = mfn_list[pp->p_pagenum];

2037     if (*poolp == NULL) {
2038         *poolp = pp;

```

```

2039         pp->p_next = pp;
2040         pp->p_prev = pp;
2041         return;
2042     }

2044     /*
2045      * Since we try to take pages from the high end of the pool
2046      * chances are good that the pages to be put on the list will
2047      * go at or near the end of the list. so start at the end and
2048      * work backwards.
2049      */
2050     look = (*poolp)->p_prev;
2051     while (mfn < mfn_list[look->p_pagenum]) {
2052         look = look->p_prev;
2053         if (look == (*poolp)->p_prev)
2054             break; /* backed all the way to front of list */
2055     }

2057     /* insert after look */
2058     pp->p_prev = look;
2059     pp->p_next = look->p_next;
2060     pp->p_next->p_prev = pp;
2061     look->p_next = pp;
2062     if (mfn < mfn_list[(**poolp)->p_pagenum]) {
2063         /*
2064          * we inserted a new first list element
2065          * adjust pool pointer to newly inserted element
2066          */
2067         **poolp = pp;
2068     }
2069 }

2071 /*
2072  * Add a page to the io_pool. Setting the force flag will force the page
2073  * into the io_pool no matter what.
2074  */
2075 static void
2076 add_page_to_pool(page_t *pp, int force)
2077 {
2078     page_t *highest;
2079     page_t *freep = NULL;

2081     mutex_enter(&io_pool_lock);
2082     /*
2083      * Always keep the scarce low memory pages
2084      */
2085     if (mfn_list[pp->p_pagenum] < PFN_16MEG) {
2086         ++io_pool_cnt;
2087         page_io_pool_add(&io_pool_16m, pp);
2088         goto done;
2089     }
2090     if (io_pool_cnt < io_pool_cnt_max || force || io_pool_4g == NULL) {
2091         ++io_pool_cnt;
2092         page_io_pool_add(&io_pool_4g, pp);
2093     } else {
2094         highest = io_pool_4g->p_prev;
2095         if (mfn_list[pp->p_pagenum] < mfn_list[highest->p_pagenum]) {
2096             page_io_pool_sub(&io_pool_4g, highest, highest);
2097             page_io_pool_add(&io_pool_4g, pp);
2098             freep = highest;
2099         } else {
2100             freep = pp;
2101         }
2102     }
2103 done:
2104     mutex_exit(&io_pool_lock);

```

```

2105     if (freep)
2106         page_free(freep, 1);
2107 }

2110 int contig_pfn_cnt;    /* no of pfns in the contig pfn list */
2111 int contig_pfn_max;    /* capacity of the contig pfn list */
2112 int next_alloc_pfn;    /* next position in list to start a contig search */
2113 int contig_pfnlist_updates; /* pfn list update count */
2114 int contig_pfnlist_builds; /* how many times have we (re)built list */
2115 int contig_pfnlist_buildfailed; /* how many times has list build failed */
2116 int create_contig_pending; /* nonzero means taskq creating contig list */
2117 pfn_t *contig_pfn_list = NULL; /* list of contig pfns in ascending mfn order */

2119 /*
2120  * Function to use in sorting a list of pfns by their underlying mfns.
2121  */
2122 static int
2123 mfn_compare(const void *pfnp1, const void *pfnp2)
2124 {
2125     mfn_t mfn1 = mfn_list[(pfn_t *)pfnp1];
2126     mfn_t mfn2 = mfn_list[(pfn_t *)pfnp2];

2128     if (mfn1 > mfn2)
2129         return (1);
2130     if (mfn1 < mfn2)
2131         return (-1);
2132     return (0);
2133 }

2135 /*
2136  * Compact the contig_pfn_list by tossing all the non-contiguous
2137  * elements from the list.
2138  */
2139 static void
2140 compact_contig_pfn_list(void)
2141 {
2142     pfn_t pfn, lapfn, prev_lapfn;
2143     mfn_t mfn;
2144     int i, newcnt = 0;

2146     prev_lapfn = 0;
2147     for (i = 0; i < contig_pfn_cnt - 1; i++) {
2148         pfn = contig_pfn_list[i];
2149         lapfn = contig_pfn_list[i + 1];
2150         mfn = mfn_list[pfn];
2151         /*
2152          * See if next pfn is for a contig mfn
2153          */
2154         if (mfn_list[lapfn] != mfn + 1)
2155             continue;
2156         /*
2157          * pfn and lookahead are both put in list
2158          * unless pfn is the previous lookahead.
2159          */
2160         if (pfn != prev_lapfn)
2161             contig_pfn_list[newcnt++] = pfn;
2162         contig_pfn_list[newcnt++] = lapfn;
2163         prev_lapfn = lapfn;
2164     }
2165     for (i = newcnt; i < contig_pfn_cnt; i++)
2166         contig_pfn_list[i] = 0;
2167     contig_pfn_cnt = newcnt;
2168 }

2170 /*ARGSUSED*/

```

```

2171 static void
2172 call_create_contiglist(void *arg)
2173 {
2174     (void) create_contig_pfnlist(PG_WAIT);
2175 }

2177 /*
2178  * Create list of freelist pfns that have underlying
2179  * contiguous mfns. The list is kept in ascending mfn order.
2180  * returns 1 if list created else 0.
2181  */
2182 static int
2183 create_contig_pfnlist(uint_t flags)
2184 {
2185     pfn_t pfn;
2186     page_t *pp;
2187     int ret = 1;

2189     mutex_enter(&contig_list_lock);
2190     if (contig_pfn_list != NULL)
2191         goto out;
2192     contig_pfn_max = freemem + (freemem / 10);
2193     contig_pfn_list = kmem_zalloc(contig_pfn_max * sizeof (pfn_t),
2194     (flags & PG_WAIT) ? KM_SLEEP : KM_NOSLEEP);
2195     if (contig_pfn_list == NULL) {
2196         /*
2197          * If we could not create the contig list (because
2198          * we could not sleep for memory). Dispatch a taskq that can
2199          * sleep to get the memory.
2200          */
2201         if (!create_contig_pending) {
2202             if (taskq_dispatch(system_taskq, call_create_contiglist,
2203             NULL, TQ_NOSLEEP) != NULL)
2204                 create_contig_pending = 1;
2205         }
2206         contig_pfnlist_buildfailed++; /* count list build failures */
2207         ret = 0;
2208         goto out;
2209     }
2210     create_contig_pending = 0;
2211     ASSERT(contig_pfn_cnt == 0);
2212     for (pfn = 0; pfn < mfn_count; pfn++) {
2213         pp = page_numtopp_nolock(pfn);
2214         if (pp == NULL || !PP_ISFREE(pp))
2215             continue;
2216         contig_pfn_list[contig_pfn_cnt] = pfn;
2217         if (++contig_pfn_cnt == contig_pfn_max)
2218             break;
2219     }
2220     /*
2221      * Sanity check the new list.
2222      */
2223     if (contig_pfn_cnt < 2) { /* no contig pfns */
2224         contig_pfn_cnt = 0;
2225         contig_pfnlist_buildfailed++;
2226         kmem_free(contig_pfn_list, contig_pfn_max * sizeof (pfn_t));
2227         contig_pfn_list = NULL;
2228         contig_pfn_max = 0;
2229         ret = 0;
2230         goto out;
2231     }
2232     qsort(contig_pfn_list, contig_pfn_cnt, sizeof (pfn_t), mfn_compare);
2233     compact_contig_pfn_list();
2234     /*
2235      * Make sure next search of the newly created contiguous pfn
2236      * list starts at the beginning of the list.

```



```

2237     */
2238     next_alloc_pfn = 0;
2239     contig_pfnlist_builds++;          /* count list builds */
2240 out:
2241     mutex_exit(&contig_list_lock);
2242     return (ret);
2243 }

2246 /*
2247  * Toss the current contig pfnlist.  Someone is about to do a massive
2248  * update to pfn->mfns mappings.  So we have them destroy the list and lock
2249  * it till they are done with their update.
2250  */
2251 void
2252 clear_and_lock_contig_pfnlist()
2253 {
2254     pfn_t *listp = NULL;
2255     size_t listsize;

2257     mutex_enter(&contig_list_lock);
2258     if (contig_pfn_list != NULL) {
2259         listp = contig_pfn_list;
2260         listsize = contig_pfn_max * sizeof (pfn_t);
2261         contig_pfn_list = NULL;
2262         contig_pfn_max = contig_pfn_cnt = 0;
2263     }
2264     if (listp != NULL)
2265         kmem_free(listp, listsize);
2266 }

2268 /*
2269  * Unlock the contig_pfn_list.  The next attempted use of it will cause
2270  * it to be re-created.
2271  */
2272 void
2273 unlock_contig_pfnlist()
2274 {
2275     mutex_exit(&contig_list_lock);
2276 }

2278 /*
2279  * Update the contiguous pfn list in response to a pfn <-> mfn reassignment
2280  */
2281 void
2282 update_contig_pfnlist(pfn_t pfn, mfn_t oldmfn, mfn_t newmfn)
2283 {
2284     int probe_hi, probe_lo, probe_pos, insert_after, insert_point;
2285     pfn_t probe_pfn;
2286     mfn_t probe_mfn;
2287     int drop_lock = 0;

2289     if (mutex_owner(&contig_list_lock) != curthread) {
2290         drop_lock = 1;
2291         mutex_enter(&contig_list_lock);
2292     }
2293     if (contig_pfn_list == NULL)
2294         goto done;
2295     contig_pfnlist_updates++;
2296     /*
2297      * Find the pfn in the current list.  Use a binary chop to locate it.
2298      */
2299     probe_hi = contig_pfn_cnt - 1;
2300     probe_lo = 0;
2301     probe_pos = (probe_hi + probe_lo) / 2;
2302     while ((probe_pfn = contig_pfn_list[probe_pos]) != pfn) {

```

```

2303         if (probe_pos == probe_lo) { /* pfn not in list */
2304             probe_pos = -1;
2305             break;
2306         }
2307         if (pfn_to_mfn(probe_pfn) <= oldmfn)
2308             probe_lo = probe_pos;
2309         else
2310             probe_hi = probe_pos;
2311         probe_pos = (probe_hi + probe_lo) / 2;
2312     }
2313     if (probe_pos >= 0) {
2314         /*
2315          * Remove pfn from list and ensure next alloc
2316          * position stays in bounds.
2317          */
2318         if (--contig_pfn_cnt <= next_alloc_pfn)
2319             next_alloc_pfn = 0;
2320         if (contig_pfn_cnt < 2) { /* no contig pfns */
2321             contig_pfn_cnt = 0;
2322             kmem_free(contig_pfn_list,
2323                 contig_pfn_max * sizeof (pfn_t));
2324             contig_pfn_list = NULL;
2325             contig_pfn_max = 0;
2326             goto done;
2327         }
2328         ovbcopy(&contig_pfn_list[probe_pos + 1],
2329             &contig_pfn_list[probe_pos],
2330             (contig_pfn_cnt - probe_pos) * sizeof (pfn_t));
2331     }
2332     if (newmfn == MFN_INVALID)
2333         goto done;
2334     /*
2335      * Check if new mfn has adjacent mfns in the list
2336      */
2337     probe_hi = contig_pfn_cnt - 1;
2338     probe_lo = 0;
2339     insert_after = -2;
2340     do {
2341         probe_pos = (probe_hi + probe_lo) / 2;
2342         probe_mfn = pfn_to_mfn(contig_pfn_list[probe_pos]);
2343         if (newmfn == probe_mfn + 1)
2344             insert_after = probe_pos;
2345         else if (newmfn == probe_mfn - 1)
2346             insert_after = probe_pos - 1;
2347         if (probe_pos == probe_lo)
2348             break;
2349         if (probe_mfn <= newmfn)
2350             probe_lo = probe_pos;
2351         else
2352             probe_hi = probe_pos;
2353     } while (insert_after == -2);
2354     /*
2355      * If there is space in the list and there are adjacent mfns
2356      * insert the pfn in to its proper place in the list.
2357      */
2358     if (insert_after != -2 && contig_pfn_cnt + 1 <= contig_pfn_max) {
2359         insert_point = insert_after + 1;
2360         ovbcopy(&contig_pfn_list[insert_point],
2361             &contig_pfn_list[insert_point + 1],
2362             (contig_pfn_cnt - insert_point) * sizeof (pfn_t));
2363         contig_pfn_list[insert_point] = pfn;
2364         contig_pfn_cnt++;
2365     }
2366 done:
2367     if (drop_lock)
2368         mutex_exit(&contig_list_lock);

```

```

2369 }

2371 /*
2372  * Called to (re-)populate the io_pool from the free page lists.
2373  */
2374 long
2375 populate_io_pool(void)
2376 {
2377     pfn_t pfn;
2378     mfn_t mfn, max_mfn;
2379     page_t *pp;

2381     /*
2382      * Figure out the bounds of the pool on first invocation.
2383      * We use a percentage of memory for the io pool size.
2384      * We allow that to shrink, but not to less than a fixed minimum
2385      */
2386     if (io_pool_cnt_max == 0) {
2387         io_pool_cnt_max = physmem / (100 / io_pool_physmem_pct);
2388         io_pool_cnt_lowater = io_pool_cnt_max;
2389         /*
2390          * This is the first time in populate_io_pool, grab a va to use
2391          * when we need to allocate pages.
2392          */
2393         io_pool_kva = vmem_alloc(heap_arena, PAGESIZE, VM_SLEEP);
2394     }
2395     /*
2396      * If we are out of pages in the pool, then grow the size of the pool
2397      */
2398     if (io_pool_cnt == 0) {
2399         /*
2400          * Grow the max size of the io pool by 5%, but never more than
2401          * 25% of physical memory.
2402          */
2403         if (io_pool_cnt_max < physmem / 4)
2404             io_pool_cnt_max += io_pool_cnt_max / 20;
2405     }
2406     io_pool_grows++;          /* should be a kstat? */

2408     /*
2409      * Get highest mfn on this platform, but limit to the 32 bit DMA max.
2410      */
2411     (void) mfn_to_pfn(start_mfn);
2412     max_mfn = MIN(cached_max_mfn, PFN_4GIG);
2413     for (mfn = start_mfn; mfn < max_mfn; start_mfn = ++mfn) {
2414         pfn = mfn_to_pfn(mfn);
2415         if (pfn & PFN_IS_FOREIGN_MFN)
2416             continue;
2417         /*
2418          * try to allocate it from free pages
2419          */
2420         pp = page_numtopp_alloc(pfn);
2421         if (pp == NULL)
2422             continue;
2423         PP_CLRFREE(pp);
2424         add_page_to_pool(pp, 1);
2425         if (io_pool_cnt >= io_pool_cnt_max)
2426             break;
2427     }

2429     return (io_pool_cnt);
2430 }

2432 /*
2433  * Destroy a page that was being used for DMA I/O. It may or
2434  * may not actually go back to the io_pool.

```

```

2435  */
2436 void
2437 page_destroy_io(page_t *pp)
2438 {
2439     mfn_t mfn = mfn_list[pp->p_pagenum];

2441     /*
2442      * When the page was alloc'd a reservation was made, release it now
2443      */
2444     page_unresv(1);
2445     /*
2446      * Unload translations, if any, then hash out the
2447      * page to erase its identity.
2448      */
2449     (void) hat_pageunload(pp, HAT_FORCE_PGUNLOAD);
2450     page_hashout(pp, NULL);

2452     /*
2453      * If the page came from the free lists, just put it back to them.
2454      * DomU pages always go on the free lists as well.
2455      */
2456     if (!DOMAIN_IS_INITDOMAIN(xen_info) || mfn >= PFN_4GIG) {
2457         page_free(pp, 1);
2458         return;
2459     }

2461     add_page_to_pool(pp, 0);
2462 }

2465 long contig_searches;          /* count of times contig pages requested */
2466 long contig_search_restarts;  /* count of contig ranges tried */
2467 long contig_search_failed;    /* count of contig alloc failures */

2469 /*
2470  * Free partial page list
2471  */
2472 static void
2473 free_partial_list(page_t **pplist)
2474 {
2475     page_t *pp;

2477     while (*pplist != NULL) {
2478         pp = *pplist;
2479         page_io_pool_sub(pplist, pp, pp);
2480         page_free(pp, 1);
2481     }
2482 }

2484 /*
2485  * Look thru the contiguous pfns that are not part of the io_pool for
2486  * contiguous free pages. Return a list of the found pages or NULL.
2487  */
2488 page_t *
2489 find_contig_free(uint_t npages, uint_t flags, uint64_t pfnseg,
2490                 pgcnt_t pfnalign)
2491 {
2492     page_t *pp, *plist = NULL;
2493     mfn_t mfn, prev_mfn, start_mfn;
2494     pfn_t pfn;
2495     int pages_needed, pages_requested;
2496     int search_start;

2498     /*
2499      * create the contig pfn list if not already done
2500      */

```

```

2501 retry:
2502     mutex_enter(&contig_list_lock);
2503     if (contig_pfn_list == NULL) {
2504         mutex_exit(&contig_list_lock);
2505         if (!create_contig_pfnlist(flags)) {
2506             return (NULL);
2507         }
2508         goto retry;
2509     }
2510     contig_searches++;
2511     /*
2512     * Search contiguous pfn list for physically contiguous pages not in
2513     * the io_pool. Start the search where the last search left off.
2514     */
2515     pages_requested = pages_needed = npages;
2516     search_start = next_alloc_pfn;
2517     start_mfn = prev_mfn = 0;
2518     while (pages_needed) {
2519         pfn = contig_pfn_list[next_alloc_pfn];
2520         mfn = pfn_to_mfn(pfn);
2521         /*
2522         * Check if mfn is first one or contig to previous one and
2523         * if page corresponding to mfn is free and that mfn
2524         * range is not crossing a segment boundary.
2525         */
2526         if ((prev_mfn == 0 || mfn == prev_mfn + 1) &&
2527             (pp = page_numtopp_alloc(pfn)) != NULL &&
2528             !((mfn & pfnseg) < (start_mfn & pfnseg))) {
2529             PP_CLRFREE(pp);
2530             page_io_pool_add(&plist, pp);
2531             pages_needed--;
2532             if (prev_mfn == 0) {
2533                 if (pfnalalign &&
2534                     mfn != P2ROUNDUP(mfn, pfnalign)) {
2535                     /*
2536                     * not properly aligned
2537                     */
2538                     contig_search_restarts++;
2539                     free_partial_list(&plist);
2540                     pages_needed = pages_requested;
2541                     start_mfn = prev_mfn = 0;
2542                     goto skip;
2543                 }
2544                 start_mfn = mfn;
2545             }
2546             prev_mfn = mfn;
2547         } else {
2548             contig_search_restarts++;
2549             free_partial_list(&plist);
2550             pages_needed = pages_requested;
2551             start_mfn = prev_mfn = 0;
2552         }
2553     skip:
2554     if (++next_alloc_pfn == contig_pfn_cnt)
2555         next_alloc_pfn = 0;
2556     if (next_alloc_pfn == search_start)
2557         break; /* all pfns searched */
2558     }
2559     mutex_exit(&contig_list_lock);
2560     if (pages_needed) {
2561         contig_search_failed++;
2562         /*
2563         * Failed to find enough contig pages.
2564         * free partial page list
2565         */
2566         free_partial_list(&plist);

```

```

2567     }
2568     return (plist);
2569 }
2571 /*
2572 * Search the reserved io pool pages for a page range with the
2573 * desired characteristics.
2574 */
2575 page_t *
2576 page_io_pool_alloc(dden_attr_t *mattr, int contig, pgcnt_t minctg)
2577 {
2578     page_t *pp_first, *pp_last;
2579     page_t *pp, **poolp;
2580     pgcnt_t nwanted, pfnalign;
2581     uint64_t pfnseg;
2582     mfn_t mfn, tmfn, hi_mfn, lo_mfn;
2583     int align, attempt = 0;
2585     if (minctg == 1)
2586         contig = 0;
2587     lo_mfn = mmu_btop(mattr->dma_attr_addr_lo);
2588     hi_mfn = mmu_btop(mattr->dma_attr_addr_hi);
2589     pfnseg = mmu_btop(mattr->dma_attr_seg);
2590     align = maxbit(mattr->dma_attr_align, mattr->dma_attr_minxfer);
2591     if (align > MMU_PAGESIZE)
2592         pfnalign = mmu_btop(align);
2593     else
2594         pfnalign = 0;
2596     try_again:
2597     /*
2598     * See if we want pages for a legacy device
2599     */
2600     if (hi_mfn < PFN_16MEG)
2601         poolp = &io_pool_16m;
2602     else
2603         poolp = &io_pool_4g;
2604     try_smaller:
2605     /*
2606     * Take pages from I/O pool. We'll use pages from the highest
2607     * MFN range possible.
2608     */
2609     pp_first = pp_last = NULL;
2610     mutex_enter(&io_pool_lock);
2611     nwanted = minctg;
2612     for (pp = *poolp; pp && nwanted > 0; ) {
2613         pp = pp->p_prev;
2615         /*
2616         * skip pages above allowable range
2617         */
2618         mfn = mfn_list[pp->p_pagenum];
2619         if (hi_mfn < mfn)
2620             goto skip;
2622         /*
2623         * stop at pages below allowable range
2624         */
2625         if (lo_mfn > mfn)
2626             break;
2627     restart:
2628     if (pp_last == NULL) {
2629         /*
2630         * Check alignment
2631         */
2632         tmfn = mfn - (minctg - 1);

```

```

2633         if (pfalign && tmfn != P2ROUNDUP(tmfn, pfalign))
2634             goto skip; /* not properly aligned */
2635         /*
2636          * Check segment
2637          */
2638         if ((mfns & pfns) < (tmfn & pfns))
2639             goto skip; /* crosses seg boundary */
2640         /*
2641          * Start building page list
2642          */
2643         pp_first = pp_last = pp;
2644         nwanted--;
2645     } else {
2646         /*
2647          * check physical contiguity if required
2648          */
2649         if (contig &&
2650             mfn_list[pp_first->p_pagenum] != mfns + 1) {
2651             /*
2652              * not a contiguous page, restart list.
2653              */
2654             pp_last = NULL;
2655             nwanted = minctg;
2656             goto restart;
2657         } else { /* add page to list */
2658             pp_first = pp;
2659             nwanted--;
2660         }
2661     }
2662 skip:
2663     if (pp == *poolp)
2664         break;
2665 }
2666
2667 /*
2668  * If we didn't find memory. Try the more constrained pool, then
2669  * sweep free pages into the DMA pool and try again.
2670  */
2671 if (nwanted != 0) {
2672     mutex_exit(&io_pool_lock);
2673     /*
2674      * If we were looking in the less constrained pool and
2675      * didn't find pages, try the more constrained pool.
2676      */
2677     if (poolp == &io_pool_4g) {
2678         poolp = &io_pool_16m;
2679         goto try_smaller;
2680     }
2681     kmem_reap();
2682     if (++attempt < 4) {
2683         /*
2684          * Grab some more io_pool pages
2685          */
2686         (void) populate_io_pool();
2687         goto try_again; /* go around and retry */
2688     }
2689     return (NULL);
2690 }
2691 /*
2692  * Found the pages, now snip them from the list
2693  */
2694 page_io_pool_sub(poolp, pp_first, pp_last);
2695 io_pool_cnt -= minctg;
2696 /*
2697  * reset low water mark
2698  */

```

```

2699         if (io_pool_cnt < io_pool_cnt_lowater)
2700             io_pool_cnt_lowater = io_pool_cnt;
2701         mutex_exit(&io_pool_lock);
2702         return (pp_first);
2703     }
2704
2705 page_t *
2706 page_swap_with_hypervisor(struct vnode *vp, u_offset_t off, caddr_t vaddr,
2707     ddi_dma_attr_t *mattr, uint_t flags, pgcnt_t minctg)
2708 {
2709     uint_t kflags;
2710     int order, extra, extpages, i, contig, nbits, extents;
2711     page_t *pp, *expp, *pp_first, **pplist = NULL;
2712     mfn_t *mfntlist = NULL;
2713
2714     contig = flags & PG_PHYSCONTIG;
2715     if (minctg == 1)
2716         contig = 0;
2717     flags &= ~PG_PHYSCONTIG;
2718     kflags = flags & PG_WAIT ? KM_SLEEP : KM_NOSLEEP;
2719     /*
2720      * Hypervisor will allocate extents, if we want contig
2721      * pages extent must be >= minctg
2722      */
2723     if (contig) {
2724         order = highbit(minctg) - 1;
2725         if (minctg & ((1 << order) - 1))
2726             order++;
2727         extpages = 1 << order;
2728     } else {
2729         order = 0;
2730         extpages = minctg;
2731     }
2732     if (extpages > minctg) {
2733         extra = extpages - minctg;
2734         if (!page_resv(extra, kflags))
2735             return (NULL);
2736     }
2737     pp_first = NULL;
2738     pplist = kmem_alloc(extpages * sizeof (page_t *), kflags);
2739     if (pplist == NULL)
2740         goto balloon_fail;
2741     mfntlist = kmem_alloc(extpages * sizeof (mfn_t), kflags);
2742     if (mfntlist == NULL)
2743         goto balloon_fail;
2744     pp = page_create_va(vp, off, minctg * PAGE_SIZE, flags, &kvseg, vaddr);
2745     if (pp == NULL)
2746         goto balloon_fail;
2747     pp_first = pp;
2748     if (extpages > minctg) {
2749         /*
2750          * fill out the rest of extent pages to swap
2751          * with the hypervisor
2752          */
2753         for (i = 0; i < extra; i++) {
2754             expp = page_create_va(vp,
2755                 (u_offset_t)(uintptr_t)io_pool_kva,
2756                 PAGE_SIZE, flags, &kvseg, io_pool_kva);
2757             if (expp == NULL)
2758                 goto balloon_fail;
2759             (void) hat_pageunload(expp, HAT_FORCE_PGUNLOAD);
2760             page_io_unlock(expp);
2761             page_hashout(expp, NULL);
2762             page_io_lock(expp);
2763             /*
2764              * add page to end of list

```

```

2765         */
2766         expp->p_prev = pp_first->p_prev;
2767         expp->p_next = pp_first;
2768         expp->p_prev->p_next = expp;
2769         pp_first->p_prev = expp;
2770     }
2771 }
2772
2773 for (i = 0; i < extpages; i++) {
2774     pplist[i] = pp;
2775     pp = pp->p_next;
2776 }
2777 nbits = highbit(mattr->dma_attr_addr_hi);
2778 extents = contig ? 1 : minctg;
2779 if (balloon_replace_pages(extents, pplist, nbits, order,
2780     mfnlist) != extents) {
2781     if (ioalloc_dbg)
2782         cmn_err(CE_NOTE, "request to hypervisor
2783             " for %d pages, maxaddr %" PRIx64 " failed",
2784             extpages, mattr->dma_attr_addr_hi);
2785     goto balloon_fail;
2786 }
2787
2788 kmem_free(pplist, extpages * sizeof(page_t *));
2789 kmem_free(mfnlist, extpages * sizeof(mfn_t));
2790 /*
2791  * Return any excess pages to free list
2792  */
2793 if (extpages > minctg) {
2794     for (i = 0; i < extra; i++) {
2795         pp = pp_first->p_prev;
2796         page_sub(&pp_first, pp);
2797         page_io_unlock(pp);
2798         page_unresv(1);
2799         page_free(pp, 1);
2800     }
2801 }
2802 return (pp_first);
2803 balloon_fail:
2804 /*
2805  * Return pages to free list and return failure
2806  */
2807 while (pp_first != NULL) {
2808     pp = pp_first;
2809     page_sub(&pp_first, pp);
2810     page_io_unlock(pp);
2811     if (pp->p_vnode != NULL)
2812         page_hashout(pp, NULL);
2813     page_free(pp, 1);
2814 }
2815 if (pplist)
2816     kmem_free(pplist, extpages * sizeof(page_t *));
2817 if (mfnlist)
2818     kmem_free(mfnlist, extpages * sizeof(mfn_t));
2819 page_unresv(extpages - minctg);
2820 return (NULL);
2821 }
2822
2823 static void
2824 return_partial_alloc(page_t *plist)
2825 {
2826     page_t *pp;
2827
2828     while (plist != NULL) {
2829         pp = plist;
2830         page_sub(&plist, pp);

```

```

2831         page_io_unlock(pp);
2832         page_destroy_io(pp);
2833     }
2834 }
2835
2836 static page_t *
2837 page_get_contigpages(
2838     struct vnode *vp,
2839     u_offset_t off,
2840     int *npagesp,
2841     uint_t flags,
2842     caddr_t vaddr,
2843     ddi_dma_attr_t *mattr)
2844 {
2845     mfn_t max_mfn = HYPERVISOR_memory_op(XENMEM_maximum_ram_page, NULL);
2846     page_t *plist; /* list to return */
2847     page_t *pp, *mcpl;
2848     int contig, anyaddr, npages, getone = 0;
2849     mfn_t lo_mfn;
2850     mfn_t hi_mfn;
2851     pgcnt_t pfnalign = 0;
2852     int align, sgllen;
2853     uint64_t pfnseg;
2854     pgcnt_t minctg;
2855
2856     npages = *npagesp;
2857     ASSERT(mattr != NULL);
2858     lo_mfn = mmu_btop(mattr->dma_attr_addr_lo);
2859     hi_mfn = mmu_btop(mattr->dma_attr_addr_hi);
2860     sgllen = mattr->dma_attr_sgllen;
2861     pfnseg = mmu_btop(mattr->dma_attr_seg);
2862     align = maxbit(mattr->dma_attr_align, mattr->dma_attr_minxfer);
2863     if (align > MMU_PAGESIZE)
2864         pfnalign = mmu_btop(align);
2865
2866     contig = flags & PG_PHYSCONTIG;
2867     if (npages == -1) {
2868         npages = 1;
2869         pfnalign = 0;
2870     }
2871     /*
2872      * Clear the contig flag if only one page is needed.
2873      */
2874     if (npages == 1) {
2875         getone = 1;
2876         contig = 0;
2877     }
2878
2879     /*
2880      * Check if any page in the system is fine.
2881      */
2882     anyaddr = lo_mfn == 0 && hi_mfn >= max_mfn;
2883     if (!contig && anyaddr && !pfnalign) {
2884         flags &= ~PG_PHYSCONTIG;
2885         plist = page_create_va(vp, off, npages * MMU_PAGESIZE,
2886             flags, &kvseg, vaddr);
2887         if (plist != NULL) {
2888             *npagesp = 0;
2889             return (plist);
2890         }
2891     }
2892     plist = NULL;
2893     minctg = howmany(npages, sgllen);
2894     while (npages > sgllen || getone) {
2895         if (minctg > npages)
2896             minctg = npages;

```

```

2897     mcpl = NULL;
2898     /*
2899      * We could want contig pages with no address range limits.
2900      */
2901     if (anyaddr && contig) {
2902         /*
2903          * Look for free contig pages to satisfy the request.
2904          */
2905         mcpl = find_contig_free(minctg, flags, pfnseg,
2906                               pfnalign);
2907     }
2908     /*
2909      * Try the reserved io pools next
2910      */
2911     if (mcpl == NULL)
2912         mcpl = page_io_pool_alloc(mattr, contig, minctg);
2913     if (mcpl != NULL) {
2914         pp = mcpl;
2915         do {
2916             if (!page_hashin(pp, vp, off, NULL)) {
2917                 panic("page_get_contigpages:"
2918                      " hashin failed"
2919                      " pp %p, vp %p, off %llx",
2920                      (void *)pp, (void *)vp, off);
2921             }
2922             off += MMU_PAGESIZE;
2923             PP_CLRFREE(pp);
2924             PP_CLRAGED(pp);
2925             page_set_props(pp, P_REF);
2926             page_io_lock(pp);
2927             pp = pp->p_next;
2928         } while (pp != mcpl);
2929     } else {
2930         /*
2931          * Hypervisor exchange doesn't handle segment or
2932          * alignment constraints
2933          */
2934         if (mattr->dma_attr_seg < mattr->dma_attr_addr_hi ||
2935             pfnalign)
2936             goto fail;
2937         /*
2938          * Try exchanging pages with the hypervisor
2939          */
2940         mcpl = page_swap_with_hypervisor(vp, off, vaddr, mattr,
2941                                         flags, minctg);
2942         if (mcpl == NULL)
2943             goto fail;
2944         off += minctg * MMU_PAGESIZE;
2945     }
2946     check_dma(mattr, mcpl, minctg);
2947     /*
2948      * Here with a minctg run of contiguous pages, add them to the
2949      * list we will return for this request.
2950      */
2951     page_list_concat(&plist, &mcpl);
2952     npages -= minctg;
2953     *npagesp = npages;
2954     sgllen--;
2955     if (getone)
2956         break;
2957 }
2958 return (plist);
2959 fail:
2960 return_partial_alloc(plist);
2961 return (NULL);
2962 }

```

```

2964 /*
2965  * Allocator for domain 0 I/O pages. We match the required
2966  * DMA attributes and contiguity constraints.
2967  */
2968 /*ARGSUSED*/
2969 page_t *
2970 page_create_io(
2971     struct vnode      *vp,
2972     u_offset_t        off,
2973     uint_t            bytes,
2974     uint_t            flags,
2975     struct as         *as,
2976     caddr_t           vaddr,
2977     ddi_dma_attr_t    *mattr)
2978 {
2979     page_t *plist = NULL, *pp;
2980     int npages = 0, contig, anyaddr, pages_req;
2981     mfn_t lo_mfn;
2982     mfn_t hi_mfn;
2983     pgcnt_t pfnalign = 0;
2984     int align;
2985     int is_domu = 0;
2986     int dummy, bytes_got;
2987     mfn_t max_mfn = HYPERVISOR_memory_op(XENMEM_maximum_ram_page, NULL);
2988
2989     ASSERT(mattr != NULL);
2990     lo_mfn = mmu_btop(mattr->dma_attr_addr_lo);
2991     hi_mfn = mmu_btop(mattr->dma_attr_addr_hi);
2992     align = maxbit(mattr->dma_attr_align, mattr->dma_attr_minxfer);
2993     if (align > MMU_PAGESIZE)
2994         pfnalign = mmu_btop(align);
2995
2996     /*
2997      * Clear the contig flag if only one page is needed or the scatter
2998      * gather list length is >= npages.
2999      */
3000     pages_req = npages = mmu_btopr(bytes);
3001     contig = (flags & PG_PHYSCONTIG);
3002     bytes = P2ROUNDUP(bytes, MMU_PAGESIZE);
3003     if (bytes == MMU_PAGESIZE || mattr->dma_attr_sgllen >= npages)
3004         contig = 0;
3005
3006     /*
3007      * Check if any old page in the system is fine.
3008      * DomU should always go down this path.
3009      */
3010     is_domu = !DOMAIN_IS_INITDOMAIN(xen_info);
3011     anyaddr = lo_mfn == 0 && hi_mfn >= max_mfn && !pfnalign;
3012     if ((!contig && anyaddr) || is_domu) {
3013         flags &= ~PG_PHYSCONTIG;
3014         plist = page_create_va(vp, off, bytes, flags, &kvseg, vaddr);
3015         if (plist != NULL)
3016             return (plist);
3017         else if (is_domu)
3018             return (NULL); /* no memory available */
3019     }
3020     /*
3021      * DomU should never reach here
3022      */
3023     if (contig) {
3024         plist = page_get_contigpages(vp, off, &npages, flags, vaddr,
3025                                     mattr);
3026         if (plist == NULL)
3027             goto fail;
3028         bytes_got = (pages_req - npages) << MMU_PAGESHIFT;

```

```

3029         vaddr += bytes_got;
3030         off += bytes_got;
3031         /*
3032          * We now have all the contiguous pages we need, but
3033          * we may still need additional non-contiguous pages.
3034          */
3035     }
3036     /*
3037     * now loop collecting the requested number of pages, these do
3038     * not have to be contiguous pages but we will use the contig
3039     * page alloc code to get the pages since it will honor any
3040     * other constraints the pages may have.
3041     */
3042     while (npages--) {
3043         dummy = -1;
3044         pp = page_get_contigpages(vp, off, &dummy, flags, vaddr, mattr);
3045         if (pp == NULL)
3046             goto fail;
3047         page_add(&plist, pp);
3048         vaddr += MMU_PAGESIZE;
3049         off += MMU_PAGESIZE;
3050     }
3051     return (plist);
3052 fail:
3053     /*
3054     * Failed to get enough pages, return ones we did get
3055     */
3056     return_partial_alloc(plist);
3057     return (NULL);
3058 }

3060 /*
3061  * Lock and return the page with the highest mfn that we can find. last_mfn
3062  * holds the last one found, so the next search can start from there. We
3063  * also keep a counter so that we don't loop forever if the machine has no
3064  * free pages.
3065  *
3066  * This is called from the balloon thread to find pages to give away. new_high
3067  * is used when new mfn's have been added to the system - we will reset our
3068  * search if the new mfn's are higher than our current search position.
3069  */
3070 page_t *
3071 page_get_high_mfn(mfn_t new_high)
3072 {
3073     static mfn_t last_mfn = 0;
3074     pfn_t pfn;
3075     page_t *pp;
3076     ulong_t loop_count = 0;

3078     if (new_high > last_mfn)
3079         last_mfn = new_high;

3081     for (; loop_count < mfn_count; loop_count++, last_mfn--) {
3082         if (last_mfn == 0) {
3083             last_mfn = cached_max_mfn;
3084         }

3086         pfn = mfn_to_pfn(last_mfn);
3087         if (pfn & PFN_IS_FOREIGN_MFN)
3088             continue;

3090         /* See if the page is free. If so, lock it. */
3091         pp = page_numtopp_alloc(pfn);
3092         if (pp == NULL)
3093             continue;
3094         PP_CLRFREE(pp);

```

```

3096         ASSERT(PAGE_EXCL(pp));
3097         ASSERT(pp->p_vnode == NULL);
3098         ASSERT(!that_page_is_mapped(pp));
3099         last_mfn--;
3100         return (pp);
3101     }
3102     return (NULL);
3103 }

3105 #else /* !_xpv */

3107 /*
3108  * get a page from any list with the given mnode
3109  */
3110 static page_t *
3111 page_get_mnode_anylist(ulong_t origbin, uchar_t szc, uint_t flags,
3112     int mnode, int mtype, ddi_dma_attr_t *dma_attr)
3113 {
3114     kmutex_t          *pcm;
3115     int                i;
3116     page_t            *pp;
3117     page_t            *first_pp;
3118     uint64_t          pgaddr;
3119     ulong_t           bin;
3120     int                mtypestart;
3121     int                plw_initialized;
3122     page_list_walker_t plw;

3124     VM_STAT_ADD(pga_vmstats.pgma_alloc);

3126     ASSERT((flags & PG_MATCH_COLOR) == 0);
3127     ASSERT(szc == 0);
3128     ASSERT(dma_attr != NULL);

3130     MTYPE_START(mnode, mtype, flags);
3131     if (mtype < 0) {
3132         VM_STAT_ADD(pga_vmstats.pgma_alloccempty);
3133         return (NULL);
3134     }

3136     mtypestart = mtype;

3138     bin = origbin;

3140     /*
3141     * check up to page_colors + 1 bins - origbin may be checked twice
3142     * because of BIN_STEP skip
3143     */
3144     do {
3145         plw_initialized = 0;

3147         for (plw.plw_count = 0;
3148             plw.plw_count < page_colors; plw.plw_count++) {

3150             if (PAGE_FREELISTS(mnode, szc, bin, mtype) == NULL)
3151                 goto nextfreebin;

3153             pcm = PC_BIN_MUTEX(mnode, bin, PG_FREE_LIST);
3154             mutex_enter(pcm);
3155             pp = PAGE_FREELISTS(mnode, szc, bin, mtype);
3156             first_pp = pp;
3157             while (pp != NULL) {
3158                 if (IS_DUMP_PAGE(pp) || page_trylock(pp,
3159                     SE_EXCL) == 0) {
3160                     pp = pp->p_next;

```

```

3161         if (pp == first_pp) {
3162             pp = NULL;
3163         }
3164         continue;
3165     }
3167     ASSERT(PP_ISFREE(pp));
3168     ASSERT(PP_ISAGED(pp));
3169     ASSERT(pp->p_vnode == NULL);
3170     ASSERT(pp->p_hash == NULL);
3171     ASSERT(pp->p_offset == (u_offset_t)-1);
3172     ASSERT(pp->p_szc == szc);
3173     ASSERT(PFN_2_MEM_NODE(pp->p_pagenum) == mnode);
3174     /* check if page within DMA attributes */
3175     pgaddr = pa_to_ma(pfn_to_pa(pp->p_pagenum));
3176     if ((pgaddr >= dma_attr->dma_attr_addr_lo) &&
3177         (pgaddr + MMU_PAGESIZE - 1 <=
3178          dma_attr->dma_attr_addr_hi)) {
3179         break;
3180     }
3182     /* continue looking */
3183     page_unlock(pp);
3184     pp = pp->p_next;
3185     if (pp == first_pp)
3186         pp = NULL;
3188 }
3189 if (pp != NULL) {
3190     ASSERT(mtype == PP_2_MTYPE(pp));
3191     ASSERT(pp->p_szc == 0);
3193     /* found a page with specified DMA attributes */
3194     page_sub(&PAGE_FREELISTS(mnode, szc, bin,
3195                mtype), pp);
3196     page_ctr_sub(mnode, mtype, pp, PG_FREE_LIST);
3198     if ((PP_ISFREE(pp) == 0) ||
3199         (PP_ISAGED(pp) == 0)) {
3200         cmn_err(CE_PANIC, "page %p is not free",
3201                (void *)pp);
3202     }
3204     mutex_exit(pcm);
3205     check_dma(dma_attr, pp, 1);
3206     VM_STAT_ADD(pga_vmstats.pgma_allocok);
3207     return (pp);
3208 }
3209 mutex_exit(pcm);
3210 nextfreebin:
3211 if (plw_initialized == 0) {
3212     page_list_walk_init(szc, 0, bin, 1, 0, &plw);
3213     ASSERT(plw.plw_ceq_dif == page_colors);
3214     plw_initialized = 1;
3215 }
3217 if (plw.plw_do_split) {
3218     pp = page_freelist_split(szc, bin, mnode,
3219                             mtype,
3220                             mmu_btop(dma_attr->dma_attr_addr_lo),
3221                             mmu_btop(dma_attr->dma_attr_addr_hi + 1),
3222                             &plw);
3223     if (pp != NULL) {
3224         check_dma(dma_attr, pp, 1);
3225         return (pp);
3226     }

```

```

3227     }
3229     bin = page_list_walk_next_bin(szc, bin, &plw);
3230 }
3232     MTYPE_NEXT(mnode, mtype, flags);
3233 } while (mtype >= 0);
3235 /* failed to find a page in the freelist; try it in the cachelist */
3237 /* reset mtype start for cachelist search */
3238 mtype = mtypestart;
3239 ASSERT(mtype >= 0);
3241 /* start with the bin of matching color */
3242 bin = origbin;
3244 do {
3245     for (i = 0; i <= page_colors; i++) {
3246         if (PAGE_CACHELISTS(mnode, bin, mtype) == NULL)
3247             goto nextcachebin;
3248         pcm = PC_BIN_MUTEX(mnode, bin, PG_CACHE_LIST);
3249         mutex_enter(pcm);
3250         pp = PAGE_CACHELISTS(mnode, bin, mtype);
3251         first_pp = pp;
3252         while (pp != NULL) {
3253             if (IS_DUMP_PAGE(pp) || page_trylock(pp,
3254                SE_EXCL) == 0) {
3255                 pp = pp->p_next;
3256                 if (pp == first_pp)
3257                     pp = NULL;
3258                 continue;
3259             }
3260             ASSERT(pp->p_vnode);
3261             ASSERT(PP_ISAGED(pp) == 0);
3262             ASSERT(pp->p_szc == 0);
3263             ASSERT(PFN_2_MEM_NODE(pp->p_pagenum) == mnode);
3265             /* check if page within DMA attributes */
3267             pgaddr = pa_to_ma(pfn_to_pa(pp->p_pagenum));
3268             if ((pgaddr >= dma_attr->dma_attr_addr_lo) &&
3269                 (pgaddr + MMU_PAGESIZE - 1 <=
3270                  dma_attr->dma_attr_addr_hi)) {
3271                 break;
3272             }
3274             /* continue looking */
3275             page_unlock(pp);
3276             pp = pp->p_next;
3277             if (pp == first_pp)
3278                 pp = NULL;
3279         }
3281     if (pp != NULL) {
3282         ASSERT(mtype == PP_2_MTYPE(pp));
3283         ASSERT(pp->p_szc == 0);
3285         /* found a page with specified DMA attributes */
3286         page_sub(&PAGE_CACHELISTS(mnode, bin,
3287                                mtype), pp);
3288         page_ctr_sub(mnode, mtype, pp, PG_CACHE_LIST);
3290         mutex_exit(pcm);
3291         ASSERT(pp->p_vnode);
3292         ASSERT(PP_ISAGED(pp) == 0);

```



```

3293         check_dma(dma_attr, pp, 1);
3294         VM_STAT_ADD(pga_vmstats.pgma_allocok);
3295         return (pp);
3296     }
3297     mutex_exit(pcm);
3298 nextcachebin:
3299     bin += (i == 0) ? BIN_STEP : 1;
3300     bin &= page_colors_mask;
3301 }
3302     MTYPE_NEXT(mnode, mtype, flags);
3303 } while (mtype >= 0);

3305     VM_STAT_ADD(pga_vmstats.pgma_allocfailed);
3306     return (NULL);
3307 }

3309 /*
3310  * This function is similar to page_get_freelist()/page_get_cachelist()
3311  * but it searches both the lists to find a page with the specified
3312  * color (or no color) and DMA attributes. The search is done in the
3313  * freelist first and then in the cache list within the highest memory
3314  * range (based on DMA attributes) before searching in the lower
3315  * memory ranges.
3316  *
3317  * Note: This function is called only by page_create_io().
3318  */
3319 /*ARGSUSED*/
3320 static page_t *
3321 page_get_anylist(struct vnode *vp, u_offset_t off, struct as *as, caddr_t vaddr,
3322     size_t size, uint_t flags, ddi_dma_attr_t *dma_attr, lgrp_t *lgrp)
3323 {
3324     uint_t      bin;
3325     int         mtype;
3326     page_t      *pp;
3327     int         n;
3328     int         m;
3329     int         szc;
3330     int         fullrange;
3331     int         mnode;
3332     int         local_failed_stat = 0;
3333     lgrp_mnode_cookie_t    lgrp_cookie;

3335     VM_STAT_ADD(pga_vmstats.pga_alloc);

3337     /* only base pagesize currently supported */
3338     if (size != MMU_PAGESIZE)
3339         return (NULL);

3341     /*
3342      * If we're passed a specific lgroup, we use it. Otherwise,
3343      * assume first-touch placement is desired.
3344      */
3345     if (!LGRP_EXISTS(lgrp))
3346         lgrp = lgrp_home_lgrp();

3348     /* LINTED */
3349     AS_2_BIN(as, seg, vp, vaddr, bin, 0);

3351     /*
3352      * Only hold one freelist or cachelist lock at a time, that way we
3353      * can start anywhere and not have to worry about lock
3354      * ordering.
3355      */
3356     if (dma_attr == NULL) {
3357         n = mtypel6m;
3358         m = mtypetop;

```

```

3359         fullrange = 1;
3360         VM_STAT_ADD(pga_vmstats.pga_nulldmaattr);
3361     } else {
3362         pfn_t pfnlo = mmu_btop(dma_attr->dma_attr_addr_lo);
3363         pfn_t pfnhi = mmu_btop(dma_attr->dma_attr_addr_hi);

3365         /*
3366          * We can guarantee alignment only for page boundary.
3367          */
3368         if (dma_attr->dma_attr_align > MMU_PAGESIZE)
3369             return (NULL);

3371         /* Sanity check the dma_attr */
3372         if (pfnlo > pfnhi)
3373             return (NULL);

3375         n = pfn_2_mtype(pfnlo);
3376         m = pfn_2_mtype(pfnhi);

3378         fullrange = ((pfnlo == mnoderanges[n].mnr_pfnlo) &&
3379             (pfnhi >= mnoderanges[m].mnr_pfnhi));
3380     }
3381     VM_STAT_COND_ADD(fullrange == 0, pga_vmstats.pga_notfullrange);

3383     szc = 0;

3385     /* cycling thru mtype handled by RANGE0 if n == mtypel6m */
3386     if (n == mtypel6m) {
3387         flags |= PGI_MT_RANGE0;
3388         n = m;
3389     }

3391     /*
3392      * Try local memory node first, but try remote if we can't
3393      * get a page of the right color.
3394      */
3395     LGRP_MNODE_COOKIE_INIT(lgrp_cookie, lgrp, LGRP_SRCH_HIER);
3396     while ((mnode = lgrp_memnode_choose(&lgrp_cookie)) >= 0) {
3397         /*
3398          * allocate pages from high pfn to low.
3399          */
3400         mtype = m;
3401         do {
3402             if (fullrange != 0) {
3403                 pp = page_get_mnode_freelist(mnode,
3404                     bin, mtype, szc, flags);
3405                 if (pp == NULL) {
3406                     pp = page_get_mnode_cachelist(
3407                         bin, flags, mnode, mtype);
3408                 }
3409             } else {
3410                 pp = page_get_mnode_anylist(bin, szc,
3411                     flags, mnode, mtype, dma_attr);
3412             }
3413             if (pp != NULL) {
3414                 VM_STAT_ADD(pga_vmstats.pga_allocok);
3415                 check_dma(dma_attr, pp, 1);
3416                 return (pp);
3417             }
3418         } while (mtype != n &&
3419             (mtype = mnoderanges[mtype].mnr_next) != -1);
3420     if (!local_failed_stat) {
3421         lgrp_stat_add(lgrp->lgrp_id, LGRP_NUM_ALLOC_FAIL, 1);
3422         local_failed_stat = 1;
3423     }
3424 }

```

```

3425     VM_STAT_ADD(pga_vmstats.pga_allocfailed);
3427     return (NULL);
3428 }

3430 /*
3431  * page_create_io()
3432  */
3433  * This function is a copy of page_create_va() with an additional
3434  * argument 'mattr' that specifies DMA memory requirements to
3435  * the page list functions. This function is used by the segkmem
3436  * allocator so it is only to create new pages (i.e PG_EXCL is
3437  * set).
3438  *
3439  * Note: This interface is currently used by x86 PSM only and is
3440  * not fully specified so the commitment level is only for
3441  * private interface specific to x86. This interface uses PSM
3442  * specific page_get_anylist() interface.
3443  */

3445 #define PAGE_HASH_SEARCH(index, pp, vp, off) { \
3446     for ((pp) = page_hash[(index)]; (pp); (pp) = (pp)->p_hash) { \
3447         if ((pp)->p_vnode == (vp) && (pp)->p_offset == (off)) \
3448             break; \
3449     } \
3450 }

3453 page_t *
3454 page_create_io(
3455     struct vnode      *vp,
3456     u_offset_t        off,
3457     uint_t            bytes,
3458     uint_t            flags,
3459     struct as         *as,
3460     caddr_t           vaddr,
3461     ddi_dma_attr_t   *mattr) /* DMA memory attributes if any */
3462 {
3463     page_t            *plist = NULL;
3464     uint_t            plist_len = 0;
3465     pgcnt_t           npages;
3466     page_t            *npp = NULL;
3467     uint_t            pages_req;
3468     page_t            *pp;
3469     kmutex_t          *phm = NULL;
3470     uint_t            index;

3472     TRACE_4(TR_FAC_VM, TR_PAGE_CREATE_START,
3473           "page_create_start:vp %p off %llx bytes %u flags %x",
3474           vp, off, bytes, flags);

3476     ASSERT((flags & ~(PG_EXCL | PG_WAIT | PG_PHYSCONTIG)) == 0);

3478     pages_req = npages = mmu_btopr(bytes);

3480     /*
3481      * Do the freemem and pcf accounting.
3482      */
3483     if (!page_create_wait(npages, flags)) {
3484         return (NULL);
3485     }

3487     TRACE_2(TR_FAC_VM, TR_PAGE_CREATE_SUCCESS,
3488           "page_create_success:vp %p off %llx", vp, off);

3490     /*

```

```

3491     * If satisfying this request has left us with too little
3492     * memory, start the wheels turning to get some back. The
3493     * first clause of the test prevents waking up the pageout
3494     * daemon in situations where it would decide that there's
3495     * nothing to do.
3496     */
3497     if (nscan < deSCAN && freemem < minfree) {
3498         TRACE_1(TR_FAC_VM, TR_PAGEOUT_CV_SIGNAL,
3499               "pageout_cv_signal:freemem %ld", freemem);
3500         cv_signal(&proc_pageout->p_cv);
3501     }

3503     if (flags & PG_PHYSCONTIG) {

3505         plist = page_get_contigpage(&npages, mattr, 1);
3506         if (plist == NULL) {
3507             page_create_putback(npages);
3508             return (NULL);
3509         }

3511         pp = plist;

3513     do {
3514         if (!page_hashin(pp, vp, off, NULL)) {
3515             panic("pg_creat_io: hashin failed %p %p %llx",
3516                   (void *)pp, (void *)vp, off);
3517         }
3518         VM_STAT_ADD(page_create_new);
3519         off += MMU_PAGESIZE;
3520         PP_CLRFREE(pp);
3521         PP_CLRAGED(pp);
3522         page_set_props(pp, P_REF);
3523         pp = pp->p_next;
3524     } while (pp != plist);

3526     if (!npages) {
3527         check_dma(mattr, plist, pages_req);
3528         return (plist);
3529     } else {
3530         vaddr += (pages_req - npages) << MMU_PAGESHIFT;
3531     }

3533     /*
3534      * fall-thru:
3535      *
3536      * page_get_contigpage returns when npages <= sglenn.
3537      * Grab the rest of the non-contig pages below from anylist.
3538      */
3539 }

3541     /*
3542      * Loop around collecting the requested number of pages.
3543      * Most of the time, we have to 'create' a new page. With
3544      * this in mind, pull the page off the free list before
3545      * getting the hash lock. This will minimize the hash
3546      * lock hold time, nesting, and the like. If it turns
3547      * out we don't need the page, we put it back at the end.
3548      */
3549     while (npages-- > 0) {
3550         phm = NULL;

3552         index = PAGE_HASH_FUNC(vp, off);
3553     top:
3554         ASSERT(phm == NULL);
3555         ASSERT(index == PAGE_HASH_FUNC(vp, off));
3556         ASSERT(MUTEX_NOT_HELD(page_vnode_mutex(vp)));

```

```

3558     if (npp == NULL) {
3559         /*
3560          * Try to get the page of any color either from
3561          * the freelist or from the cache list.
3562          */
3563         npp = page_get_anylist(vp, off, as, vaddr, MMU_PAGESIZE,
3564             flags & ~PG_MATCH_COLOR, mattr, NULL);
3565         if (npp == NULL) {
3566             if (mattr == NULL) {
3567                 /*
3568                  * Not looking for a special page;
3569                  * panic!
3570                  */
3571                 panic("no page found %d", (int)npages);
3572             }
3573             /*
3574              * No page found! This can happen
3575              * if we are looking for a page
3576              * within a specific memory range
3577              * for DMA purposes. If PG_WAIT is
3578              * specified then we wait for a
3579              * while and then try again. The
3580              * wait could be forever if we
3581              * don't get the page(s) we need.
3582              *
3583              * Note: XXX We really need a mechanism
3584              * to wait for pages in the desired
3585              * range. For now, we wait for any
3586              * pages and see if we can use it.
3587              */
3588
3589             if ((mattr != NULL) && (flags & PG_WAIT)) {
3590                 delay(10);
3591                 goto top;
3592             }
3593             goto fail; /* undo accounting stuff */
3594         }
3595
3596         if (PP_ISAGED(npp) == 0) {
3597             /*
3598              * Since this page came from the
3599              * cachelist, we must destroy the
3600              * old vnode association.
3601              */
3602             page_hashout(npp, (kmutex_t *)NULL);
3603         }
3604     }
3605
3606     /*
3607      * We own this page!
3608      */
3609     ASSERT(PAGE_EXCL(npp));
3610     ASSERT(npp->p_vnode == NULL);
3611     ASSERT(!that_page_is_mapped(npp));
3612     PP_CLRFREE(npp);
3613     PP_CLRAGED(npp);
3614
3615     /*
3616      * Here we have a page in our hot little mits and are
3617      * just waiting to stuff it on the appropriate lists.
3618      * Get the mutex and check to see if it really does
3619      * not exist.
3620      */
3621     phm = PAGE_HASH_MUTEX(index);
3622     mutex_enter(phm);

```

```

3623     PAGE_HASH_SEARCH(index, pp, vp, off);
3624     if (pp == NULL) {
3625         VM_STAT_ADD(page_create_new);
3626         pp = npp;
3627         npp = NULL;
3628         if (!page_hashin(pp, vp, off, phm)) {
3629             /*
3630              * Since we hold the page hash mutex and
3631              * just searched for this page, page_hashin
3632              * had better not fail. If it does, that
3633              * means somethread did not follow the
3634              * page hash mutex rules. Panic now and
3635              * get it over with. As usual, go down
3636              * holding all the locks.
3637              */
3638             ASSERT(MUTEX_HELD(phm));
3639             panic("page create: hashin fail %p %p %llx %p",
3640                 (void *)pp, (void *)vp, off, (void *)phm);
3641         }
3642         ASSERT(MUTEX_HELD(phm));
3643         mutex_exit(phm);
3644         phm = NULL;
3645
3646         /*
3647          * Hat layer locking need not be done to set
3648          * the following bits since the page is not hashed
3649          * and was on the free list (i.e., had no mappings).
3650          *
3651          * Set the reference bit to protect
3652          * against immediate pageout
3653          *
3654          * XXXmh modify freelist code to set reference
3655          * bit so we don't have to do it here.
3656          */
3657         page_set_props(pp, P_REF);
3658     } else {
3659         ASSERT(MUTEX_HELD(phm));
3660         mutex_exit(phm);
3661         phm = NULL;
3662         /*
3663          * NOTE: This should not happen for pages associated
3664          * with kernel vnode 'kvp'.
3665          */
3666         /* XX64 - to debug why this happens! */
3667         ASSERT(!VN_ISKAS(vp));
3668         if (VN_ISKAS(vp))
3669             cmn_err(CE_NOTE,
3670                 "page_create: page not expected "
3671                 "in hash list for kernel vnode - pp 0x%p",
3672                 (void *)pp);
3673         VM_STAT_ADD(page_create_exists);
3674         goto fail;
3675     }
3676
3677     /*
3678      * Got a page! It is locked. Acquire the i/o
3679      * lock since we are going to use the p_next and
3680      * p_prev fields to link the requested pages together.
3681      */
3682     page_io_lock(pp);
3683     page_add(&plist, pp);
3684     plist = plist->p_next;
3685     off += MMU_PAGESIZE;
3686     vaddr += MMU_PAGESIZE;
3687 }
3688

```

```

3690     check_dma(mattr, plist, pages_req);
3691     return (plist);

3693 fail:
3694     if (npp != NULL) {
3695         /*
3696          * Did not need this page after all.
3697          * Put it back on the free list.
3698          */
3699         VM_STAT_ADD(page_create_putbacks);
3700         PP_SETFREE(npp);
3701         PP_SETAGED(npp);
3702         npp->p_offset = (u_offset_t)-1;
3703         page_list_add(npp, PG_FREE_LIST | PG_LIST_TAIL);
3704         page_unlock(npp);
3705     }

3707     /*
3708     * Give up the pages we already got.
3709     */
3710     while (plist != NULL) {
3711         pp = plist;
3712         page_sub(&plist, pp);
3713         page_io_unlock(pp);
3714         plist_len++;
3715         /*LINTED: constant in conditional ctx*/
3716         VN_DISPOSE(pp, B_INVALID, 0, kcred);
3717     }

3719     /*
3720     * VN_DISPOSE does freemem accounting for the pages in plist
3721     * by calling page_free. So, we need to undo the pcf accounting
3722     * for only the remaining pages.
3723     */
3724     VM_STAT_ADD(page_create_putbacks);
3725     page_create_putback(pages_req - plist_len);

3727     return (NULL);
3728 }
3729 #endif /* !__xpv */

3732 /*
3733  * Copy the data from the physical page represented by "frompp" to
3734  * that represented by "topp". ppcopy uses CPU->cpu_caddr1 and
3735  * CPU->cpu_caddr2. It assumes that no one uses either map at interrupt
3736  * level and no one sleeps with an active mapping there.
3737  *
3738  * Note that the ref/mod bits in the page_t's are not affected by
3739  * this operation, hence it is up to the caller to update them appropriately.
3740  */
3741 int
3742 ppcopy(page_t *frompp, page_t *topp)
3743 {
3744     caddr_t      pp_addr1;
3745     caddr_t      pp_addr2;
3746     hat_mempte_t pte1;
3747     hat_mempte_t pte2;
3748     kmutex_t     *ppaddr_mutex;
3749     label_t      ljb;
3750     int          ret = 1;

3752     ASSERT_STACK_ALIGNED();
3753     ASSERT(PAGE_LOCKED(frompp));
3754     ASSERT(PAGE_LOCKED(toppp));

```

```

3756     if (kpm_enable) {
3757         pp_addr1 = hat_kpm_page2va(frompp, 0);
3758         pp_addr2 = hat_kpm_page2va(toppp, 0);
3759         kpreempt_disable();
3760     } else {
3761         /*
3762          * disable pre-emption so that CPU can't change
3763          */
3764         kpreempt_disable();

3766         pp_addr1 = CPU->cpu_caddr1;
3767         pp_addr2 = CPU->cpu_caddr2;
3768         pte1 = CPU->cpu_caddr1pte;
3769         pte2 = CPU->cpu_caddr2pte;

3771         ppaddr_mutex = &CPU->cpu_ppaddr_mutex;
3772         mutex_enter(ppaddr_mutex);

3774         hat_mempte_remap(page_pptonum(frompp), pp_addr1, pte1,
3775             PROT_READ | HAT_STORECACHING_OK, HAT_LOAD_NOCONSIST);
3776         hat_mempte_remap(page_pptonum(toppp), pp_addr2, pte2,
3777             PROT_READ | PROT_WRITE | HAT_STORECACHING_OK,
3778             HAT_LOAD_NOCONSIST);
3779     }

3781     if (on_fault(&ljb)) {
3782         ret = 0;
3783         goto faulted;
3784     }
3785     if (use_sse_pagecopy)
3786 #ifdef __xpv
3787         page_copy_no_xmm(pp_addr2, pp_addr1);
3788 #else
3789         hwblkpagecopy(pp_addr1, pp_addr2);
3790 #endif
3791     else
3792         bcopy(pp_addr1, pp_addr2, PAGE_SIZE);

3794     no_fault();
3795 faulted:
3796     if (!kpm_enable) {
3797 #ifdef __xpv
3798         /*
3799          * We can't leave unused mappings laying about under the
3800          * hypervisor, so blow them away.
3801          */
3802         if (HYPERVISOR_update_va_mapping((uintptr_t)pp_addr1, 0,
3803             UVMF_INVLPG | UVMF_LOCAL) < 0)
3804             panic("HYPERVISOR_update_va_mapping() failed");
3805         if (HYPERVISOR_update_va_mapping((uintptr_t)pp_addr2, 0,
3806             UVMF_INVLPG | UVMF_LOCAL) < 0)
3807             panic("HYPERVISOR_update_va_mapping() failed");
3808 #endif
3809         mutex_exit(ppaddr_mutex);
3810     }
3811     kpreempt_enable();
3812     return (ret);
3813 }

3815 void
3816 pagezero(page_t *pp, uint_t off, uint_t len)
3817 {
3818     ASSERT(PAGE_LOCKED(pp));
3819     pfnzero(page_pptonum(pp), off, len);
3820 }

```

```

3822 /*
3823  * Zero the physical page from off to off + len given by pfn
3824  * without changing the reference and modified bits of page.
3825  *
3826  * We use this using CPU private page address #2, see ppcopy() for more info.
3827  * pfnzero() must not be called at interrupt level.
3828  */
3829 void
3830 pfnzero(pfn_t pfn, uint_t off, uint_t len)
3831 {
3832     caddr_t      pp_addr2;
3833     hat_mempte_t pte2;
3834     kmutex_t      *ppaddr_mutex = NULL;

3836     ASSERT_STACK_ALIGNED();
3837     ASSERT(len <= MMU_PAGESIZE);
3838     ASSERT(off <= MMU_PAGESIZE);
3839     ASSERT(off + len <= MMU_PAGESIZE);

3841     if (kpm_enable && !pfn_is_foreign(pfn)) {
3842         pp_addr2 = hat_kpm_pfn2va(pfn);
3843         kpreempt_disable();
3844     } else {
3845         kpreempt_disable();

3847         pp_addr2 = CPU->cpu_caddr2;
3848         pte2 = CPU->cpu_caddr2pte;

3850         ppaddr_mutex = &CPU->cpu_ppaddr_mutex;
3851         mutex_enter(ppaddr_mutex);

3853         hat_mempte_remap(pfn, pp_addr2, pte2,
3854             PROT_READ | PROT_WRITE | HAT_STORECACHING_OK,
3855             HAT_LOAD_NOCONSIST);
3856     }

3858     if (use_sse_pagezero) {
3859 #ifdef __xpv
3860         uint_t rem;

3862         /*
3863          * zero a byte at a time until properly aligned for
3864          * block_zero_no_xmm().
3865          */
3866         while (!P2NPHASE(off, ((uint_t)BLOCKZEROALIGN)) && len-- > 0)
3867             pp_addr2[off++] = 0;

3869         /*
3870          * Now use faster block_zero_no_xmm() for any range
3871          * that is properly aligned and sized.
3872          */
3873         rem = P2PHASE(len, ((uint_t)BLOCKZEROALIGN));
3874         len -= rem;
3875         if (len != 0) {
3876             block_zero_no_xmm(pp_addr2 + off, len);
3877             off += len;
3878         }

3880         /*
3881          * zero remainder with byte stores.
3882          */
3883         while (rem-- > 0)
3884             pp_addr2[off++] = 0;
3885 #else
3886         hwblkclr(pp_addr2 + off, len);

```

```

3887 #endif
3888     } else {
3889         bzero(pp_addr2 + off, len);
3890     }

3892     if (!kpm_enable || pfn_is_foreign(pfn)) {
3893 #ifdef __xpv
3894         /*
3895          * On the hypervisor this page might get used for a page
3896          * table before any intervening change to this mapping,
3897          * so blow it away.
3898          */
3899         if (HYPERVISOR_update_va_mapping((uintptr_t)pp_addr2, 0,
3900             UVMF_INVLPG) < 0)
3901             panic("HYPERVISOR_update_va_mapping() failed");
3902 #endif
3903         mutex_exit(ppaddr_mutex);
3904     }

3906     kpreempt_enable();
3907 }

3909 /*
3910  * Platform-dependent page scrub call.
3911  */
3912 void
3913 pagescrub(page_t *pp, uint_t off, uint_t len)
3914 {
3915     /*
3916      * For now, we rely on the fact that pagezero() will
3917      * always clear UEs.
3918      */
3919     pagezero(pp, off, len);
3920 }

3922 /*
3923  * set up two private addresses for use on a given CPU for use in ppcopy()
3924  */
3925 void
3926 setup_vaddr_for_ppcopy(struct cpu *cpup)
3927 {
3928     void *addr;
3929     hat_mempte_t pte_pa;

3931     addr = vmem_alloc(heap_arena, mmu_ptob(1), VM_SLEEP);
3932     pte_pa = hat_mempte_setup(addr);
3933     cpup->cpu_caddr1 = addr;
3934     cpup->cpu_caddr1pte = pte_pa;

3936     addr = vmem_alloc(heap_arena, mmu_ptob(1), VM_SLEEP);
3937     pte_pa = hat_mempte_setup(addr);
3938     cpup->cpu_caddr2 = addr;
3939     cpup->cpu_caddr2pte = pte_pa;

3941     mutex_init(&cpup->cpu_ppaddr_mutex, NULL, MUTEX_DEFAULT, NULL);
3942 }

3944 /*
3945  * Undo setup_vaddr_for_ppcopy
3946  */
3947 void
3948 teardown_vaddr_for_ppcopy(struct cpu *cpup)
3949 {
3950     mutex_destroy(&cpup->cpu_ppaddr_mutex);

3952     hat_mempte_release(cpup->cpu_caddr2, cpup->cpu_caddr2pte);

```

```

3953     cpup->cpu_caddr2pte = 0;
3954     vmem_free(heap_arena, cpup->cpu_caddr2, mmu_ptob(1));
3955     cpup->cpu_caddr2 = 0;

3957     hat_mempte_release(cpup->cpu_caddr1, cpup->cpu_caddr1pte);
3958     cpup->cpu_caddr1pte = 0;
3959     vmem_free(heap_arena, cpup->cpu_caddr1, mmu_ptob(1));
3960     cpup->cpu_caddr1 = 0;
3961 }

3963 /*
3964  * Function for flushing D-cache when performing module relocations
3965  * to an alternate mapping. Unnecessary on Intel / AMD platforms.
3966  */
3967 void
3968 dcache_flushall()
3969 {}

    62 size_t
    63 exec_get_spslew(void)
    64 {
    65     return (0);
    66 }

3971 /*
3972  * Allocate a memory page. The argument 'seed' can be any pseudo-random
3973  * number to vary where the pages come from. This is quite a hacked up
3974  * method -- it works for now, but really needs to be fixed up a bit.
3975  *
3976  * We currently use page_create_va() on the kvp with fake offsets,
3977  * segments and virt address. This is pretty bogus, but was copied from the
3978  * old hat_i86.c code. A better approach would be to specify either mnode
3979  * random or mnode local and takes a page from whatever color has the MOST
3980  * available - this would have a minimal impact on page coloring.
3981  */
3982 page_t *
3983 page_get_physical(uintptr_t seed)
3984 {
3985     page_t *pp;
3986     u_offset_t offset;
3987     static struct seg tmpseg;
3988     static uintptr_t ctr = 0;

3990     /*
3991      * This code is gross, we really need a simpler page allocator.
3992      *
3993      * We need to assign an offset for the page to call page_create_va()
3994      * To avoid conflicts with other pages, we get creative with the offset.
3995      * For 32 bits, we need an offset > 4Gig
3996      * For 64 bits, need an offset somewhere in the VA hole.
3997      */
3998     offset = seed;
3999     if (offset > kernelbase)
4000         offset -= kernelbase;
4001     offset <<= MMU_PAGESHIFT;
4002 #if defined(__amd64)
4003     offset += mmu.hole_start; /* something in VA hole */
4004 #else
4005     offset += 1ULL << 40; /* something > 4 Gig */
4006 #endif

4008     if (page_resv(1, KM_NOSLEEP) == 0)
4009         return (NULL);

4011 #ifdef DEBUG
4012     pp = page_exists(&kvp, offset);

```

```

4013     if (pp != NULL)
4014         panic("page already exists %p", (void *)pp);
4015 #endif

4017     pp = page_create_va(&kvp, offset, MMU_PAGESIZE, PG_EXCL,
4018                       &tmpseg, (caddr_t)(ctr += MMU_PAGESIZE)); /* changing VA usage */
4019     if (pp != NULL) {
4020         page_io_unlock(pp);
4021         page_downgrade(pp);
4022     }
4023     return (pp);
4024 }

    unchanged_portion_omitted

```

new/usr/src/uts/intel/ia32/ml/modstubs.s

1

```
*****
47146 Thu Jun 30 22:03:52 2016
new/usr/src/uts/intel/ia32/ml/modstubs.s
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
_____unchanged_portion_omitted_____

362 #endif /* __i386 */

364 #define STUB(module, fcname, retfcn) \
365     STUB_COMMON(module, fcname, mod_hold_stub, retfcn, 0)

367 /*
368  * "weak stub", don't load on account of this call
369  */
370 #define WSTUB(module, fcname, retfcn) \
371     STUB_COMMON(module, fcname, retfcn, retfcn, MODS_WEAK)

373 /*
374  * "non-unloadable stub", don't bother 'holding' module if it's already loaded
375  * since the module cannot be unloaded.
376  *
377  * User *MUST* guarantee the module is not unloadable (no _fini routine).
378  */
379 #define NO_UNLOAD_STUB(module, fcname, retfcn) \
380     STUB_UNLOADABLE(module, fcname, retfcn, retfcn, MODS_NOUNLOAD)

382 /*
383  * "weak stub" for non-unloadable module, don't load on account of this call
384  */
385 #define NO_UNLOAD_WSTUB(module, fcname, retfcn) \
386     STUB_UNLOADABLE(module, fcname, retfcn, retfcn, MODS_NOUNLOAD|MODS_WEAK)

388 /*
389  * this is just a marker for the beginning area of text that contains stubs
390  */
391     ENTRY_NP(stubs_base)
392     nop

394 /*
395  * WARNING WARNING WARNING!!!!!!
396  *
397  * On the MODULE macro you MUST NOT use any spaces!!! They are
398  * significant to the preprocessor. With ansi c there is a way around this
399  * but for some reason (yet to be investigated) ansi didn't work for other
400  * reasons!
401  *
402  * When zero is used as the return function, the system will call
403  * panic if the stub can't be resolved.
404  */

406 /*
407  * Stubs for devfs. A non-unloadable module.
408  */

410 #ifndef DEVFS_MODULE
411     MODULE(devfs,fs);
412     NO_UNLOAD_STUB(devfs, devfs_clean,          nomod_minus_one);
413     NO_UNLOAD_STUB(devfs, devfs_lookupname,     nomod_minus_one);
414     NO_UNLOAD_STUB(devfs, devfs_walk,          nomod_minus_one);
415     NO_UNLOAD_STUB(devfs, devfs_devpolicy,     nomod_minus_one);
416     NO_UNLOAD_STUB(devfs, devfs_reset_perm,    nomod_minus_one);
417     NO_UNLOAD_STUB(devfs, devfs_remdrv_cleanup, nomod_minus_one);

```

new/usr/src/uts/intel/ia32/ml/modstubs.s

2

```
418     END_MODULE(devfs);
419 #endif

421 #ifndef DEV_MODULE
422     MODULE(dev,fs);
423     NO_UNLOAD_STUB(dev, sdev_modctl_readdir,    nomod_minus_one);
424     NO_UNLOAD_STUB(dev, sdev_modctl_readdir_free, nomod_minus_one);
425     NO_UNLOAD_STUB(dev, devname_filename_register, nomod_minus_one);
426     NO_UNLOAD_STUB(dev, sdev_modctl_devexists,  nomod_minus_one);
427     NO_UNLOAD_STUB(dev, devname_profile_update, nomod_minus_one);
428     NO_UNLOAD_STUB(dev, sdev_devstate_change,  nomod_minus_one);
429     NO_UNLOAD_STUB(dev, devvt_getvnodeops,     nomod_minus_one);
430     NO_UNLOAD_STUB(dev, devpts_getvnodeops,    nomod_zero);
431     END_MODULE(dev);
432 #endif

434 /*
435  * Stubs for specfs. A non-unloadable module.
436  */

438 #ifndef SPEC_MODULE
439     MODULE(specfs,fs);
440     NO_UNLOAD_STUB(specfs, common_specvp,      nomod_zero);
441     NO_UNLOAD_STUB(specfs, makeectty,          nomod_zero);
442     NO_UNLOAD_STUB(specfs, makespecvp,         nomod_zero);
443     NO_UNLOAD_STUB(specfs, smark,              nomod_zero);
444     NO_UNLOAD_STUB(specfs, spec_segmap,        nomod_einval);
445     NO_UNLOAD_STUB(specfs, specfind,           nomod_zero);
446     NO_UNLOAD_STUB(specfs, specvp,             nomod_zero);
447     NO_UNLOAD_STUB(specfs, devi_stillreferenced, nomod_zero);
448     NO_UNLOAD_STUB(specfs, spec_getvnodeops,   nomod_zero);
449     NO_UNLOAD_STUB(specfs, spec_char_map,      nomod_zero);
450     NO_UNLOAD_STUB(specfs, specvp_devfs,       nomod_zero);
451     NO_UNLOAD_STUB(specfs, spec_assoc_vp_with_devi, nomod_void);
452     NO_UNLOAD_STUB(specfs, spec_hold_devi_by_vp, nomod_zero);
453     NO_UNLOAD_STUB(specfs, spec_snode_walk,    nomod_void);
454     NO_UNLOAD_STUB(specfs, spec_devi_open_count, nomod_minus_one);
455     NO_UNLOAD_STUB(specfs, spec_is_clone,      nomod_zero);
456     NO_UNLOAD_STUB(specfs, spec_is_selfclone,  nomod_zero);
457     NO_UNLOAD_STUB(specfs, spec_fence_snode,   nomod_minus_one);
458     NO_UNLOAD_STUB(specfs, spec_unfence_snode, nomod_minus_one);
459     END_MODULE(specfs);
460 #endif

463 /*
464  * Stubs for sockfs. A non-unloadable module.
465  */
466 #ifndef SOCK_MODULE
467     MODULE(sockfs,fs);
468     NO_UNLOAD_STUB(sockfs, so_socket,          nomod_zero);
469     NO_UNLOAD_STUB(sockfs, so_socketpair,      nomod_zero);
470     NO_UNLOAD_STUB(sockfs, bind,               nomod_zero);
471     NO_UNLOAD_STUB(sockfs, listen,             nomod_zero);
472     NO_UNLOAD_STUB(sockfs, accept,             nomod_zero);
473     NO_UNLOAD_STUB(sockfs, connect,            nomod_zero);
474     NO_UNLOAD_STUB(sockfs, shutdown,           nomod_zero);
475     NO_UNLOAD_STUB(sockfs, recv,               nomod_zero);
476     NO_UNLOAD_STUB(sockfs, recvfrom,           nomod_zero);
477     NO_UNLOAD_STUB(sockfs, recvmsg,            nomod_zero);
478     NO_UNLOAD_STUB(sockfs, send,               nomod_zero);
479     NO_UNLOAD_STUB(sockfs, sendmsg,            nomod_zero);
480     NO_UNLOAD_STUB(sockfs, sendto,             nomod_zero);
481 #ifndef _SYS_CALL32_IMPL
482     NO_UNLOAD_STUB(sockfs, recv32,              nomod_zero);
483     NO_UNLOAD_STUB(sockfs, recvfrom32,         nomod_zero);

```

```

484     NO_UNLOAD_STUB(sockfs, send32,          nomod_zero);
485     NO_UNLOAD_STUB(sockfs, sendto32,       nomod_zero);
486 #endif /* _SYSCALL32_IMPL */
487     NO_UNLOAD_STUB(sockfs, getpeername,    nomod_zero);
488     NO_UNLOAD_STUB(sockfs, getsockname,    nomod_zero);
489     NO_UNLOAD_STUB(sockfs, getsockopt,     nomod_zero);
490     NO_UNLOAD_STUB(sockfs, setsockopt,     nomod_zero);
491     NO_UNLOAD_STUB(sockfs, sockconfig,     nomod_zero);
492     NO_UNLOAD_STUB(sockfs, sock_getmsg,    nomod_zero);
493     NO_UNLOAD_STUB(sockfs, sock_putmsg,    nomod_zero);
494     NO_UNLOAD_STUB(sockfs, sosendfile64,   nomod_zero);
495     NO_UNLOAD_STUB(sockfs, snf_segmap,     nomod_einval);
496     NO_UNLOAD_STUB(sockfs, sock_getfasync, nomod_zero);
497     NO_UNLOAD_STUB(sockfs, nl7c_sendfilev, nomod_zero);
498     NO_UNLOAD_STUB(sockfs, sotpi_sototpi,  nomod_zero);
499     NO_UNLOAD_STUB(sockfs, socket_sendmbk, nomod_zero);
500     NO_UNLOAD_STUB(sockfs, socket_setsockopt, nomod_zero);
501     END_MODULE(sockfs);
502 #endif

504 /*
505  * IPsec stubs.
506  */

508 #ifndef IPSECAH_MODULE
509     MODULE(ipsecah,drv);
510     WSTUB(ipsecah, ipsec_construct_inverse_acquire, nomod_zero);
511     WSTUB(ipsecah, sadb_acquire, nomod_zero);
512     WSTUB(ipsecah, ipsecah_algs_changed, nomod_zero);
513     WSTUB(ipsecah, sadb_alg_update, nomod_zero);
514     WSTUB(ipsecah, sadb_unlinkassoc, nomod_zero);
515     WSTUB(ipsecah, sadb_insertassoc, nomod_zero);
516     WSTUB(ipsecah, ipsecah_in_assocfailure, nomod_zero);
517     WSTUB(ipsecah, sadb_set_lpkt, nomod_zero);
518     WSTUB(ipsecah, ipsecah_icmp_error, nomod_zero);
519     END_MODULE(ipsecah);
520 #endif
521
522 #ifndef IPSECESP_MODULE
523     MODULE(ipsecesp,drv);
524     WSTUB(ipsecesp, ipsecesp_fill_defs, nomod_zero);
525     WSTUB(ipsecesp, ipsecesp_algs_changed, nomod_zero);
526     WSTUB(ipsecesp, ipsecesp_in_assocfailure, nomod_zero);
527     WSTUB(ipsecesp, ipsecesp_init_funcs, nomod_zero);
528     WSTUB(ipsecesp, ipsecesp_icmp_error, nomod_zero);
529     WSTUB(ipsecesp, ipsecesp_send_keepalive, nomod_zero);
530     END_MODULE(ipsecesp);
531 #endif
532
533 #ifndef KEYSOCK_MODULE
534     MODULE(keysock, drv);
535     WSTUB(keysock, keysock_plumb_ipsec, nomod_zero);
536     WSTUB(keysock, keysock_extended_reg, nomod_zero);
537     WSTUB(keysock, keysock_next_seq, nomod_zero);
538     END_MODULE(keysock);
539 #endif

541 #ifndef SPDSOCK_MODULE
542     MODULE(spdsock,drv);
543     WSTUB(spdsock, spdsock_update_pending_algs, nomod_zero);
544     END_MODULE(spdsock);
545 #endif

547 /*
548  * Stubs for nfs common code.
549  * XXX nfs_getvnodeops should go away with removal of kludge in vnode.c

```

```

550  */
551 #ifndef NFS_MODULE
552     MODULE(nfs,fs);
553     WSTUB(nfs, nfs_getvnodeops, nomod_zero);
554     WSTUB(nfs, nfs_perror, nomod_zero);
555     WSTUB(nfs, nfs_cmh_err, nomod_zero);
556     WSTUB(nfs, clcleanup_zone, nomod_zero);
557     WSTUB(nfs, clcleanup4_zone, nomod_zero);
558     END_MODULE(nfs);
559 #endif

562 /*
563  * Stubs for nfs_dlboot (diskless booting).
564  */
565 #ifndef NFS_DLBOOT_MODULE
566     MODULE(nfs_dlboot,misc);
567     STUB(nfs_dlboot, mount_root, nomod_minus_one);
568     STUB(nfs_dlboot, dhcpinit, nomod_minus_one);
569     END_MODULE(nfs_dlboot);
570 #endif

572 /*
573  * Stubs for nfs server-only code.
574  */
575 #ifndef NFSSRV_MODULE
576     MODULE(nfssrv,misc);
577     STUB(nfssrv, exportfs, nomod_minus_one);
578     STUB(nfssrv, nfs_getfh, nomod_minus_one);
579     STUB(nfssrv, nfs_l_flush, nomod_minus_one);
580     STUB(nfssrv, rfs4_check_delegated, nomod_zero);
581     STUB(nfssrv, mountd_args, nomod_minus_one);
582     NO_UNLOAD_STUB(nfssrv, rdma_start, nomod_zero);
583     NO_UNLOAD_STUB(nfssrv, nfs_svc, nomod_zero);
584     END_MODULE(nfssrv);
585 #endif

587 /*
588  * Stubs for kernel lock manager.
589  */
590 #ifndef KLM_MODULE
591     MODULE(klmmod,misc);
592     NO_UNLOAD_STUB(klmmod, lm_svc, nomod_zero);
593     NO_UNLOAD_STUB(klmmod, lm_shutdown, nomod_zero);
594     NO_UNLOAD_STUB(klmmod, lm_unexport, nomod_zero);
595     NO_UNLOAD_STUB(klmmod, lm_cprresume, nomod_zero);
596     NO_UNLOAD_STUB(klmmod, lm_cprsuspend, nomod_zero);
597     NO_UNLOAD_STUB(klmmod, lm_safelock, nomod_zero);
598     NO_UNLOAD_STUB(klmmod, lm_safemap, nomod_zero);
599     NO_UNLOAD_STUB(klmmod, lm_has_sleep, nomod_zero);
600     NO_UNLOAD_STUB(klmmod, lm_free_config, nomod_zero);
601     NO_UNLOAD_STUB(klmmod, lm_vp_active, nomod_zero);
602     NO_UNLOAD_STUB(klmmod, lm_get_sysid, nomod_zero);
603     NO_UNLOAD_STUB(klmmod, lm_rel_sysid, nomod_zero);
604     NO_UNLOAD_STUB(klmmod, lm_alloc_sysidt, nomod_minus_one);
605     NO_UNLOAD_STUB(klmmod, lm_free_sysidt, nomod_zero);
606     NO_UNLOAD_STUB(klmmod, lm_sysidt, nomod_minus_one);
607     END_MODULE(klmmod);
608 #endif

610 #ifndef KLMOPS_MODULE
611     MODULE(klmops,misc);
612     NO_UNLOAD_STUB(klmops, lm_frlock, nomod_zero);
613     NO_UNLOAD_STUB(klmops, lm4_frlock, nomod_zero);
614     NO_UNLOAD_STUB(klmops, lm_shrlock, nomod_zero);
615     NO_UNLOAD_STUB(klmops, lm4_shrlock, nomod_zero);

```



```

616     NO_UNLOAD_STUB(klmops, lm_nlm_dispatch, nomod_zero);
617     NO_UNLOAD_STUB(klmops, lm_nlm4_dispatch, nomod_zero);
618     NO_UNLOAD_STUB(klmops, lm_nlm_reclaim, nomod_zero);
619     NO_UNLOAD_STUB(klmops, lm_nlm4_reclaim, nomod_zero);
620     NO_UNLOAD_STUB(klmops, lm_register_lock_locally, nomod_zero);
621     END_MODULE(klmops);
622 #endif

624 /*
625  * Stubs for kernel TLI module
626  * XXX currently we never allow this to unload
627  */
628 #ifndef TLI_MODULE
629     MODULE(tlimod, misc);
630     NO_UNLOAD_STUB(tlimod, t_kopen, nomod_minus_one);
631     NO_UNLOAD_STUB(tlimod, t_kunbind, nomod_zero);
632     NO_UNLOAD_STUB(tlimod, t_kadvise, nomod_zero);
633     NO_UNLOAD_STUB(tlimod, t_krcvudata, nomod_zero);
634     NO_UNLOAD_STUB(tlimod, t_ksndudata, nomod_zero);
635     NO_UNLOAD_STUB(tlimod, t_kalloc, nomod_zero);
636     NO_UNLOAD_STUB(tlimod, t_kbind, nomod_zero);
637     NO_UNLOAD_STUB(tlimod, t_kclose, nomod_zero);
638     NO_UNLOAD_STUB(tlimod, t_kspoll, nomod_zero);
639     NO_UNLOAD_STUB(tlimod, t_kfree, nomod_zero);
640     END_MODULE(tlimod);
641 #endif

643 /*
644  * Stubs for kernel RPC module
645  * XXX currently we never allow this to unload
646  */
647 #ifndef RPC_MODULE
648     MODULE(rpcmod, strmod);
649     NO_UNLOAD_STUB(rpcmod, clnt_tli_kcreate, nomod_minus_one);
650     NO_UNLOAD_STUB(rpcmod, svc_tli_kcreate, nomod_minus_one);
651     NO_UNLOAD_STUB(rpcmod, bindresvport, nomod_minus_one);
652     NO_UNLOAD_STUB(rpcmod, rdma_register_mod, nomod_minus_one);
653     NO_UNLOAD_STUB(rpcmod, rdma_unregister_mod, nomod_minus_one);
654     NO_UNLOAD_STUB(rpcmod, svc_queuereq, nomod_minus_one);
655     NO_UNLOAD_STUB(rpcmod, clist_add, nomod_minus_one);
656     END_MODULE(rpcmod);
657 #endif

659 /*
660  * Stubs for des
661  */
662 #ifndef DES_MODULE
663     MODULE(des, misc);
664     STUB(des, cbc_crypt, nomod_zero);
665     STUB(des, ecb_crypt, nomod_zero);
666     STUB(des, _des_crypt, nomod_zero);
667     END_MODULE(des);
668 #endif

670 /*
671  * Stubs for procfs. A non-unloadable module.
672  */
673 #ifndef PROC_MODULE
674     MODULE(procfs, fs);
675     NO_UNLOAD_STUB(procfs, prfree, nomod_zero);
676     NO_UNLOAD_STUB(procfs, prexit, nomod_zero);
677     NO_UNLOAD_STUB(procfs, prlwpfree, nomod_zero);
678     NO_UNLOAD_STUB(procfs, prlwpexit, nomod_zero);
679     NO_UNLOAD_STUB(procfs, prinvalidate, nomod_zero);
680     NO_UNLOAD_STUB(procfs, prnsegs, nomod_zero);
681     NO_UNLOAD_STUB(procfs, prgetcred, nomod_zero);

```

```

682     NO_UNLOAD_STUB(procfs, prgetpriv, nomod_zero);
683     NO_UNLOAD_STUB(procfs, prgetprivsize, nomod_zero);
684     NO_UNLOAD_STUB(procfs, prgetsecflags, nomod_zero);
685 #endif /* ! codereview */
686     NO_UNLOAD_STUB(procfs, prgetstatus, nomod_zero);
687     NO_UNLOAD_STUB(procfs, prgetlwpstatus, nomod_zero);
688     NO_UNLOAD_STUB(procfs, prgetpsinfo, nomod_zero);
689     NO_UNLOAD_STUB(procfs, prgetlwpsinfo, nomod_zero);
690     NO_UNLOAD_STUB(procfs, oprgetstatus, nomod_zero);
691     NO_UNLOAD_STUB(procfs, oprgetpsinfo, nomod_zero);
692 #ifndef _SYSCALL32_IMPL
693     NO_UNLOAD_STUB(procfs, prgetstatus32, nomod_zero);
694     NO_UNLOAD_STUB(procfs, prgetlwpstatus32, nomod_zero);
695     NO_UNLOAD_STUB(procfs, prgetpsinfo32, nomod_zero);
696     NO_UNLOAD_STUB(procfs, prgetlwpsinfo32, nomod_zero);
697     NO_UNLOAD_STUB(procfs, oprgetstatus32, nomod_zero);
698     NO_UNLOAD_STUB(procfs, oprgetpsinfo32, nomod_zero);
699     NO_UNLOAD_STUB(procfs, psinfo_kto32, nomod_zero);
700     NO_UNLOAD_STUB(procfs, lwpsinfo_kto32, nomod_zero);
701 #endif /* _SYSCALL32_IMPL */
702     NO_UNLOAD_STUB(procfs, prnotify, nomod_zero);
703     NO_UNLOAD_STUB(procfs, prexecstart, nomod_zero);
704     NO_UNLOAD_STUB(procfs, prexecend, nomod_zero);
705     NO_UNLOAD_STUB(procfs, prrelvm, nomod_zero);
706     NO_UNLOAD_STUB(procfs, prbarrier, nomod_zero);
707     NO_UNLOAD_STUB(procfs, estimate_msacct, nomod_zero);
708     NO_UNLOAD_STUB(procfs, pr_getprot, nomod_zero);
709     NO_UNLOAD_STUB(procfs, pr_getprot_done, nomod_zero);
710     NO_UNLOAD_STUB(procfs, pr_getsegsize, nomod_zero);
711     NO_UNLOAD_STUB(procfs, pr_isobject, nomod_zero);
712     NO_UNLOAD_STUB(procfs, pr_isself, nomod_zero);
713     NO_UNLOAD_STUB(procfs, pr_allstopped, nomod_zero);
714     NO_UNLOAD_STUB(procfs, pr_free_watched_pages, nomod_zero);
715     END_MODULE(procfs);
716 #endif

718 /*
719  * Stubs for fifofs
720  */
721 #ifndef FIFO_MODULE
722     MODULE(fifofs, fs);
723     STUB(fifofs, fifovp, 0);
724     STUB(fifofs, fifo_getinfo, 0);
725     STUB(fifofs, fifo_vfastoff, 0);
726     END_MODULE(fifofs);
727 #endif

729 /*
730  * Stubs for ufs
731  */
732 * This is needed to support the old quotactl system call.
733 * When the old sysent stuff goes away, this will need to be revisited.
734 */
735 #ifndef UFS_MODULE
736     MODULE(ufs, fs);
737     STUB(ufs, quotactl, nomod_minus_one);
738     END_MODULE(ufs);
739 #endif

741 /*
742  * Stubs for zfs
743  */
744 #ifndef ZFS_MODULE
745     MODULE(zfs, fs);
746     STUB(zfs, dsl_prop_get, nomod_minus_one);
747     STUB(zfs, spa_boot_init, nomod_minus_one);

```

```

748     STUB(zfs, zfs_prop_to_name, nomod_zero);
749     END_MODULE(zfs);
750 #endif

752 /*
753  * Stubs for dcfs
754  */
755 #ifndef DCFS_MODULE
756     MODULE(dcfs,fs);
757     STUB(dcfs, decompvp, 0);
758     END_MODULE(dcfs);
759 #endif

761 /*
762  * Stubs for namefs
763  */
764 #ifndef NAMEFS_MODULE
765     MODULE(namefs,fs);
766     STUB(namefs, nm_unmountall, 0);
767     END_MODULE(namefs);
768 #endif

770 /*
771  * Stubs for sysdc
772  */
773 #ifndef SDC_MODULE
774     MODULE(SDC,sched);
775     NO_UNLOAD_STUB(SDC, sysdc_thread_enter, nomod_zero);
776     END_MODULE(SDC);
777 #endif

779 /*
780  * Stubs for ts_dptbl
781  */
782 #ifndef TS_DPTBL_MODULE
783     MODULE(TS_DPTBL,sched);
784     STUB(TS_DPTBL, ts_getdptbl, 0);
785     STUB(TS_DPTBL, ts_getkmdpris, 0);
786     STUB(TS_DPTBL, ts_getmaxumdpr, 0);
787     END_MODULE(TS_DPTBL);
788 #endif

790 /*
791  * Stubs for rt_dptbl
792  */
793 #ifndef RT_DPTBL_MODULE
794     MODULE(RT_DPTBL,sched);
795     STUB(RT_DPTBL, rt_getdptbl, 0);
796     END_MODULE(RT_DPTBL);
797 #endif

799 /*
800  * Stubs for ia_dptbl
801  */
802 #ifndef IA_DPTBL_MODULE
803     MODULE(IA_DPTBL,sched);
804     STUB(IA_DPTBL, ia_getdptbl, nomod_zero);
805     STUB(IA_DPTBL, ia_getkmdpris, nomod_zero);
806     STUB(IA_DPTBL, ia_getmaxumdpr, nomod_zero);
807     END_MODULE(IA_DPTBL);
808 #endif

810 /*
811  * Stubs for FSS scheduler
812  */
813 #ifndef FSS_MODULE

```

```

814     MODULE(FSS,sched);
815     WSTUB(FSS, fss_allocbuf, nomod_zero);
816     WSTUB(FSS, fss_freebuf, nomod_zero);
817     WSTUB(FSS, fss_changeproj, nomod_zero);
818     WSTUB(FSS, fss_changepset, nomod_zero);
819     END_MODULE(FSS);
820 #endif

822 /*
823  * Stubs for fx_dptbl
824  */
825 #ifndef FX_DPTBL_MODULE
826     MODULE(FX_DPTBL,sched);
827     STUB(FX_DPTBL, fx_getdptbl, 0);
828     STUB(FX_DPTBL, fx_getmaxumdpr, 0);
829     END_MODULE(FX_DPTBL);
830 #endif

832 /*
833  * Stubs for bootdev
834  */
835 #ifndef BOOTDEV_MODULE
836     MODULE(bootdev,misc);
837     STUB(bootdev, i_promname_to_devname, 0);
838     STUB(bootdev, i_convert_boot_device_name, 0);
839     END_MODULE(bootdev);
840 #endif

842 /*
843  * stubs for strplumb...
844  */
845 #ifndef STRPLUMB_MODULE
846     MODULE(strplumb,misc);
847     STUB(strplumb, strplumb, 0);
848     STUB(strplumb, strplumb_load, 0);
849     STUB(strplumb, strplumb_get_netdev_path, 0);
850     END_MODULE(strplumb);
851 #endif

853 /*
854  * Stubs for console configuration module
855  */
856 #ifndef CONSCONFIG_MODULE
857     MODULE(consconfig,misc);
858     STUB(consconfig, consconfig, 0);
859     STUB(consconfig, consconfig_get_usb_kb_path, 0);
860     STUB(consconfig, consconfig_get_usb_ms_path, 0);
861     STUB(consconfig, consconfig_get_plat_fbpath, 0);
862     STUB(consconfig, consconfig_console_is_ready, 0);
863     END_MODULE(consconfig);
864 #endif

866 /*
867  * Stubs for accounting.
868  */
869 #ifndef SYSACCT_MODULE
870     MODULE(sysacct,sys);
871     WSTUB(sysacct, acct, nomod_zero);
872     WSTUB(sysacct, acct_fs_in_use, nomod_zero);
873     END_MODULE(sysacct);
874 #endif

876 /*
877  * Stubs for semaphore routines. sem.c
878  */
879 #ifndef SEMSYS_MODULE

```

```

880     MODULE(semsys,sys);
881     WSTUB(semsys,semexit,          nomod_zero);
882     END_MODULE(semsys);
883 #endif

885 /*
886  * Stubs for shm routines. shm.c
887  */
888 #ifndef SHMSYS_MODULE
889     MODULE(shmsys,sys);
890     WSTUB(shmsys,shmexit,          nomod_zero);
891     WSTUB(shmsys,shmfork,          nomod_zero);
892     WSTUB(shmsys,shmgetid,         nomod_minus_one);
893     END_MODULE(shmsys);
894 #endif

896 /*
897  * Stubs for doors
898  */
899 #ifndef DOOR_MODULE
900     MODULE(doorfs,sys);
901     WSTUB(doorfs,door_slam,          nomod_zero);
902     WSTUB(doorfs,door_exit,          nomod_zero);
903     WSTUB(doorfs,door_revoke_all,    nomod_zero);
904     WSTUB(doorfs,door_fork,          nomod_zero);
905     NO_UNLOAD_STUB(doorfs,door_upcall, nomod_einval);
906     NO_UNLOAD_STUB(doorfs,door_ki_create, nomod_einval);
907     NO_UNLOAD_STUB(doorfs,door_ki_open, nomod_einval);
908     NO_UNLOAD_STUB(doorfs,door_ki_lookup, nomod_zero);
909     WSTUB(doorfs,door_ki_upcall,      nomod_einval);
910     WSTUB(doorfs,door_ki_upcall_limited, nomod_einval);
911     WSTUB(doorfs,door_ki_hold,        nomod_zero);
912     WSTUB(doorfs,door_ki_rele,        nomod_zero);
913     WSTUB(doorfs,door_ki_info,        nomod_einval);
914     END_MODULE(doorfs);
915 #endif

917 /*
918  * Stubs for MD5
919  */
920 #ifndef MD5_MODULE
921     MODULE(md5,misc);
922     WSTUB(md5,MD5Init,              nomod_zero);
923     WSTUB(md5,MD5Update,            nomod_zero);
924     WSTUB(md5,MD5Final,              nomod_zero);
925     END_MODULE(md5);
926 #endif

928 /*
929  * Stubs for idmap
930  */
931 #ifndef IDMAP_MODULE
932     MODULE(idmap,misc);
933     STUB(idmap,kidmap_batch_getgidbysid, nomod_zero);
934     STUB(idmap,kidmap_batch_getpidbysid, nomod_zero);
935     STUB(idmap,kidmap_batch_getsidbygid, nomod_zero);
936     STUB(idmap,kidmap_batch_getsidbyuid, nomod_zero);
937     STUB(idmap,kidmap_batch_getuidbysid, nomod_zero);
938     STUB(idmap,kidmap_get_create,       nomod_zero);
939     STUB(idmap,kidmap_get_destroy,       nomod_zero);
940     STUB(idmap,kidmap_get_mappings,      nomod_zero);
941     STUB(idmap,kidmap_getgidbysid,      nomod_zero);
942     STUB(idmap,kidmap_getpidbysid,      nomod_zero);
943     STUB(idmap,kidmap_getsidbygid,      nomod_zero);
944     STUB(idmap,kidmap_getsidbyuid,      nomod_zero);
945     STUB(idmap,kidmap_getuidbysid,      nomod_zero);

```

```

946     STUB(idmap, idmap_get_door,          nomod_einval);
947     STUB(idmap, idmap_unreg_dh,          nomod_einval);
948     STUB(idmap, idmap_reg_dh,           nomod_einval);
949     STUB(idmap, idmap_purge_cache,      nomod_einval);
950     END_MODULE(idmap);
951 #endif

953 /*
954  * Stubs for auditing.
955  */
956 #ifndef C2AUDIT_MODULE
957     MODULE(c2audit,sys);
958     NO_UNLOAD_STUB(c2audit, audit_init_module, nomod_zero);
959     NO_UNLOAD_STUB(c2audit, audit_start,        nomod_zero);
960     NO_UNLOAD_STUB(c2audit, audit_finish,       nomod_zero);
961     NO_UNLOAD_STUB(c2audit, audit,              nomod_zero);
962     NO_UNLOAD_STUB(c2audit, auditdoor,         nomod_zero);
963     NO_UNLOAD_STUB(c2audit, audit_closef,      nomod_zero);
964     NO_UNLOAD_STUB(c2audit, audit_core_start,  nomod_zero);
965     NO_UNLOAD_STUB(c2audit, audit_core_finish, nomod_zero);
966     NO_UNLOAD_STUB(c2audit, audit_strputmsg,   nomod_zero);
967     NO_UNLOAD_STUB(c2audit, audit_savepath,    nomod_zero);
968     NO_UNLOAD_STUB(c2audit, audit_anchorpath,  nomod_zero);
969     NO_UNLOAD_STUB(c2audit, audit_exit,        nomod_zero);
970     NO_UNLOAD_STUB(c2audit, audit_exec,        nomod_zero);
971     NO_UNLOAD_STUB(c2audit, audit_symlink,     nomod_zero);
972     NO_UNLOAD_STUB(c2audit, audit_symlink_create, nomod_zero);
973     NO_UNLOAD_STUB(c2audit, audit_vncreate_start, nomod_zero);
974     NO_UNLOAD_STUB(c2audit, audit_vncreate_finish, nomod_zero);
975     NO_UNLOAD_STUB(c2audit, audit_enterprom,  nomod_zero);
976     NO_UNLOAD_STUB(c2audit, audit_exitprom,   nomod_zero);
977     NO_UNLOAD_STUB(c2audit, audit_chdirec,    nomod_zero);
978     NO_UNLOAD_STUB(c2audit, audit_setf,       nomod_zero);
979     NO_UNLOAD_STUB(c2audit, audit_sock,       nomod_zero);
980     NO_UNLOAD_STUB(c2audit, audit_strgetmsg,   nomod_zero);
981     NO_UNLOAD_STUB(c2audit, audit_ipc,        nomod_zero);
982     NO_UNLOAD_STUB(c2audit, audit_ipget,      nomod_zero);
983     NO_UNLOAD_STUB(c2audit, audit_fdsend,     nomod_zero);
984     NO_UNLOAD_STUB(c2audit, audit_fdrecv,     nomod_zero);
985     NO_UNLOAD_STUB(c2audit, audit_priv,       nomod_zero);
986     NO_UNLOAD_STUB(c2audit, audit_setppriv,   nomod_zero);
987     NO_UNLOAD_STUB(c2audit, audit_psecflags,  nomod_zero);
988 #endif /* ! codereview */
989     NO_UNLOAD_STUB(c2audit, audit_devpolicy,   nomod_zero);
990     NO_UNLOAD_STUB(c2audit, audit_setfsat_path, nomod_zero);
991     NO_UNLOAD_STUB(c2audit, audit_cryptoadm,   nomod_zero);
992     NO_UNLOAD_STUB(c2audit, audit_kssl,        nomod_zero);
993     NO_UNLOAD_STUB(c2audit, audit_pf_policy,   nomod_zero);
994     NO_UNLOAD_STUB(c2audit, au_doormsg,       nomod_zero);
995     NO_UNLOAD_STUB(c2audit, au_uwrite,        nomod_zero);
996     NO_UNLOAD_STUB(c2audit, au_to_arg32,      nomod_zero);
997     NO_UNLOAD_STUB(c2audit, au_free_rec,      nomod_zero);
998     END_MODULE(c2audit);
999 #endif

1001 /*
1002  * Stubs for kernel rpc security service module
1003  */
1004 #ifndef RPCSEC_MODULE
1005     MODULE(rpcsec,misc);
1006     NO_UNLOAD_STUB(rpcsec, sec_clnt_revoke,    nomod_zero);
1007     NO_UNLOAD_STUB(rpcsec, authkern_create,    nomod_zero);
1008     NO_UNLOAD_STUB(rpcsec, sec_svc_msg,        nomod_zero);
1009     NO_UNLOAD_STUB(rpcsec, sec_svc_control,    nomod_zero);
1010     END_MODULE(rpcsec);
1011 #endif

```

```

1012
1013 /*
1014 * Stubs for rpc RPCSEC_GSS security service module
1015 */
1016 #ifndef RPCSEC_GSS_MODULE
1017     MODULE(rpcsec_gss,misc);
1018     NO_UNLOAD_STUB(rpcsec_gss, __svcrpcsec_gss,          nomod_zero);
1019     NO_UNLOAD_STUB(rpcsec_gss, rpc_gss_getcred,          nomod_zero);
1020     NO_UNLOAD_STUB(rpcsec_gss, rpc_gss_set_callback,     nomod_zero);
1021     NO_UNLOAD_STUB(rpcsec_gss, rpc_gss_secget,          nomod_zero);
1022     NO_UNLOAD_STUB(rpcsec_gss, rpc_gss_secfree,         nomod_zero);
1023     NO_UNLOAD_STUB(rpcsec_gss, rpc_gss_seccreate,       nomod_zero);
1024     NO_UNLOAD_STUB(rpcsec_gss, rpc_gss_set_defaults,    nomod_zero);
1025     NO_UNLOAD_STUB(rpcsec_gss, rpc_gss_revauth,         nomod_zero);
1026     NO_UNLOAD_STUB(rpcsec_gss, rpc_gss_secpurge,       nomod_zero);
1027     NO_UNLOAD_STUB(rpcsec_gss, rpc_gss_cleanup,         nomod_zero);
1028     NO_UNLOAD_STUB(rpcsec_gss, rpc_gss_get_versions,   nomod_zero);
1029     NO_UNLOAD_STUB(rpcsec_gss, rpc_gss_max_data_length, nomod_zero);
1030     NO_UNLOAD_STUB(rpcsec_gss, rpc_gss_svc_max_data_length, nomod_zero);
1031     NO_UNLOAD_STUB(rpcsec_gss, rpc_gss_get_service_type, nomod_zero);
1032     END_MODULE(rpcsec_gss);
1033 #endif

1035 /*
1036 * Stubs for PCI configurator module (misc/pcicfg).
1037 */
1038 #ifndef PCICFG_MODULE
1039     MODULE(pcicfg,misc);
1040     STUB(pcicfg, pcicfg_configure, 0);
1041     STUB(pcicfg, pcicfg_unconfigure, 0);
1042     END_MODULE(pcicfg);
1043 #endif

1045 /*
1046 * Stubs for pcieb nexus driver.
1047 */
1048 #ifndef PCIEB_MODULE
1049     MODULE(pcieb,drv);
1050     STUB(pcieb, pcieb_intel_error_workaround, 0);
1051     END_MODULE(pcieb);
1052 #endif

1054 #ifndef IWSCN_MODULE
1055     MODULE(iwscn,drv);
1056     STUB(iwscn, srpop, 0);
1057     END_MODULE(iwscn);
1058 #endif

1060 /*
1061 * Stubs for checkpoint-resume module
1062 */
1063 #ifndef CPR_MODULE
1064     MODULE(cpr,misc);
1065     STUB(cpr, cpr, 0);
1066     END_MODULE(cpr);
1067 #endif

1069 /*
1070 * Stubs for kernel probes (tnf module). Not unloadable.
1071 */
1072 #ifndef TNF_MODULE
1073     MODULE(tnf,drv);
1074     NO_UNLOAD_STUB(tnf, tnf_ref32_1,          nomod_zero);
1075     NO_UNLOAD_STUB(tnf, tnf_string_1,        nomod_zero);
1076     NO_UNLOAD_STUB(tnf, tnf_opaque_array_1,  nomod_zero);
1077     NO_UNLOAD_STUB(tnf, tnf_struct_tag_1,    nomod_zero);

```

```

1078     NO_UNLOAD_STUB(tnf, tnf_allocate,        nomod_zero);
1079     END_MODULE(tnf);
1080 #endif

1082 /*
1083 * Stubs for i86hvm bootstrapping
1084 */
1085 #ifndef HVM_BOOTSTRAP
1086     MODULE(hvm_bootstrap,misc);
1087     NO_UNLOAD_STUB(hvm_bootstrap, hvmboot_rootconf, nomod_zero);
1088     END_MODULE(hvm_bootstrap);
1089 #endif

1091 /*
1092 * Clustering: stubs for bootstrapping.
1093 */
1094 #ifndef CL_BOOTSTRAP
1095     MODULE(cl_bootstrap,misc);
1096     NO_UNLOAD_WSTUB(cl_bootstrap, clboot_modload, nomod_minus_one);
1097     NO_UNLOAD_WSTUB(cl_bootstrap, clboot_loadrootmodules, nomod_zero);
1098     NO_UNLOAD_WSTUB(cl_bootstrap, clboot_rootconf, nomod_zero);
1099     NO_UNLOAD_WSTUB(cl_bootstrap, clboot_mountroot, nomod_zero);
1100     NO_UNLOAD_WSTUB(cl_bootstrap, clconf_init, nomod_zero);
1101     NO_UNLOAD_WSTUB(cl_bootstrap, clconf_get_nodeid, nomod_zero);
1102     NO_UNLOAD_WSTUB(cl_bootstrap, clconf_maximum_nodeid, nomod_zero);
1103     NO_UNLOAD_WSTUB(cl_bootstrap, cluster, nomod_zero);
1104     END_MODULE(cl_bootstrap);
1105 #endif

1107 /*
1108 * Clustering: stubs for cluster infrastructure.
1109 */
1110 #ifndef CL_COMM_MODULE
1111     MODULE(cl_comm,misc);
1112     NO_UNLOAD_STUB(cl_comm, cladmin, nomod_minus_one);
1113     END_MODULE(cl_comm);
1114 #endif

1116 /*
1117 * Clustering: stubs for global file system operations.
1118 */
1119 #ifndef PXFS_MODULE
1120     MODULE(pxfs,fs);
1121     NO_UNLOAD_WSTUB(pxfs, clpxfs_aio_read, nomod_zero);
1122     NO_UNLOAD_WSTUB(pxfs, clpxfs_aio_write, nomod_zero);
1123     NO_UNLOAD_WSTUB(pxfs, cl_flk_state_transition_notify, nomod_zero);
1124     END_MODULE(pxfs);
1125 #endif

1127 /*
1128 * Stubs for kernel cryptographic framework module (misc/kcf).
1129 */
1130 #ifndef KCF_MODULE
1131     MODULE(kcf,misc);
1132     NO_UNLOAD_STUB(kcf, crypto_mech2id, nomod_minus_one);
1133     NO_UNLOAD_STUB(kcf, crypto_register_provider, nomod_minus_one);
1134     NO_UNLOAD_STUB(kcf, crypto_unregister_provider, nomod_minus_one);
1135     NO_UNLOAD_STUB(kcf, crypto_provider_notification, nomod_minus_one);
1136     NO_UNLOAD_STUB(kcf, crypto_op_notification, nomod_minus_one);
1137     NO_UNLOAD_STUB(kcf, crypto_kmflag, nomod_minus_one);
1138     NO_UNLOAD_STUB(kcf, crypto_digest, nomod_minus_one);
1139     NO_UNLOAD_STUB(kcf, crypto_digest_prov, nomod_minus_one);
1140     NO_UNLOAD_STUB(kcf, crypto_digest_init, nomod_minus_one);
1141     NO_UNLOAD_STUB(kcf, crypto_digest_init_prov, nomod_minus_one);
1142     NO_UNLOAD_STUB(kcf, crypto_digest_update, nomod_minus_one);
1143     NO_UNLOAD_STUB(kcf, crypto_digest_final, nomod_minus_one);

```

```

1144 NO_UNLOAD_STUB(kcf, crypto_digest_key_prov, nomod_minus_one);
1145 NO_UNLOAD_STUB(kcf, crypto_encrypt, nomod_minus_one);
1146 NO_UNLOAD_STUB(kcf, crypto_encrypt_prov, nomod_minus_one);
1147 NO_UNLOAD_STUB(kcf, crypto_encrypt_init, nomod_minus_one);
1148 NO_UNLOAD_STUB(kcf, crypto_encrypt_init_prov, nomod_minus_one);
1149 NO_UNLOAD_STUB(kcf, crypto_encrypt_update, nomod_minus_one);
1150 NO_UNLOAD_STUB(kcf, crypto_encrypt_final, nomod_minus_one);
1151 NO_UNLOAD_STUB(kcf, crypto_decrypt, nomod_minus_one);
1152 NO_UNLOAD_STUB(kcf, crypto_decrypt_prov, nomod_minus_one);
1153 NO_UNLOAD_STUB(kcf, crypto_decrypt_init, nomod_minus_one);
1154 NO_UNLOAD_STUB(kcf, crypto_decrypt_init_prov, nomod_minus_one);
1155 NO_UNLOAD_STUB(kcf, crypto_decrypt_update, nomod_minus_one);
1156 NO_UNLOAD_STUB(kcf, crypto_decrypt_final, nomod_minus_one);
1157 NO_UNLOAD_STUB(kcf, crypto_get_all_mech_info, nomod_minus_one);
1158 NO_UNLOAD_STUB(kcf, crypto_key_check, nomod_minus_one);
1159 NO_UNLOAD_STUB(kcf, crypto_key_check_prov, nomod_minus_one);
1160 NO_UNLOAD_STUB(kcf, crypto_key_derive, nomod_minus_one);
1161 NO_UNLOAD_STUB(kcf, crypto_key_generate, nomod_minus_one);
1162 NO_UNLOAD_STUB(kcf, crypto_key_generate_pair, nomod_minus_one);
1163 NO_UNLOAD_STUB(kcf, crypto_key_unwrap, nomod_minus_one);
1164 NO_UNLOAD_STUB(kcf, crypto_key_wrap, nomod_minus_one);
1165 NO_UNLOAD_STUB(kcf, crypto_mac, nomod_minus_one);
1166 NO_UNLOAD_STUB(kcf, crypto_mac_prov, nomod_minus_one);
1167 NO_UNLOAD_STUB(kcf, crypto_mac_verify, nomod_minus_one);
1168 NO_UNLOAD_STUB(kcf, crypto_mac_verify_prov, nomod_minus_one);
1169 NO_UNLOAD_STUB(kcf, crypto_mac_init, nomod_minus_one);
1170 NO_UNLOAD_STUB(kcf, crypto_mac_init_prov, nomod_minus_one);
1171 NO_UNLOAD_STUB(kcf, crypto_mac_update, nomod_minus_one);
1172 NO_UNLOAD_STUB(kcf, crypto_mac_final, nomod_minus_one);
1173 NO_UNLOAD_STUB(kcf, crypto_mac_decrypt, nomod_minus_one);
1174 NO_UNLOAD_STUB(kcf, crypto_mac_decrypt_prov, nomod_minus_one);
1175 NO_UNLOAD_STUB(kcf, crypto_mac_verify_decrypt, nomod_minus_one);
1176 NO_UNLOAD_STUB(kcf, crypto_mac_verify_decrypt_prov, nomod_minus_one);
1177 NO_UNLOAD_STUB(kcf, crypto_mac_decrypt_init, nomod_minus_one);
1178 NO_UNLOAD_STUB(kcf, crypto_mac_decrypt_init_prov, nomod_minus_one);
1179 NO_UNLOAD_STUB(kcf, crypto_mac_decrypt_update, nomod_minus_one);
1180 NO_UNLOAD_STUB(kcf, crypto_mac_decrypt_final, nomod_minus_one);
1181 NO_UNLOAD_STUB(kcf, crypto_object_copy, nomod_minus_one);
1182 NO_UNLOAD_STUB(kcf, crypto_object_create, nomod_minus_one);
1183 NO_UNLOAD_STUB(kcf, crypto_object_destroy, nomod_minus_one);
1184 NO_UNLOAD_STUB(kcf, crypto_object_find_final, nomod_minus_one);
1185 NO_UNLOAD_STUB(kcf, crypto_object_find_init, nomod_minus_one);
1186 NO_UNLOAD_STUB(kcf, crypto_object_find, nomod_minus_one);
1187 NO_UNLOAD_STUB(kcf, crypto_object_get_attribute_value, nomod_minus_one);
1188 NO_UNLOAD_STUB(kcf, crypto_object_get_size, nomod_minus_one);
1189 NO_UNLOAD_STUB(kcf, crypto_object_set_attribute_value, nomod_minus_one);
1190 NO_UNLOAD_STUB(kcf, crypto_session_close, nomod_minus_one);
1191 NO_UNLOAD_STUB(kcf, crypto_session_login, nomod_minus_one);
1192 NO_UNLOAD_STUB(kcf, crypto_session_logout, nomod_minus_one);
1193 NO_UNLOAD_STUB(kcf, crypto_session_open, nomod_minus_one);
1194 NO_UNLOAD_STUB(kcf, crypto_encrypt_mac, nomod_minus_one);
1195 NO_UNLOAD_STUB(kcf, crypto_encrypt_mac_prov, nomod_minus_one);
1196 NO_UNLOAD_STUB(kcf, crypto_encrypt_mac_init, nomod_minus_one);
1197 NO_UNLOAD_STUB(kcf, crypto_encrypt_mac_init_prov, nomod_minus_one);
1198 NO_UNLOAD_STUB(kcf, crypto_encrypt_mac_update, nomod_minus_one);
1199 NO_UNLOAD_STUB(kcf, crypto_encrypt_mac_final, nomod_minus_one);
1200 NO_UNLOAD_STUB(kcf, crypto_create_ctx_template, nomod_minus_one);
1201 NO_UNLOAD_STUB(kcf, crypto_destroy_ctx_template, nomod_minus_one);
1202 NO_UNLOAD_STUB(kcf, crypto_get_mech_list, nomod_minus_one);
1203 NO_UNLOAD_STUB(kcf, crypto_free_mech_list, nomod_minus_one);
1204 NO_UNLOAD_STUB(kcf, crypto_cancel_req, nomod_minus_one);
1205 NO_UNLOAD_STUB(kcf, crypto_cancel_ctx, nomod_minus_one);
1206 NO_UNLOAD_STUB(kcf, crypto_bufcall_alloc, nomod_minus_one);
1207 NO_UNLOAD_STUB(kcf, crypto_bufcall_free, nomod_minus_one);
1208 NO_UNLOAD_STUB(kcf, crypto_bufcall, nomod_minus_one);
1209 NO_UNLOAD_STUB(kcf, crypto_unbufcall, nomod_minus_one);

```

```

1210 NO_UNLOAD_STUB(kcf, crypto_notify_events, nomod_minus_one);
1211 NO_UNLOAD_STUB(kcf, crypto_unnotify_events, nomod_minus_one);
1212 NO_UNLOAD_STUB(kcf, crypto_get_provider, nomod_minus_one);
1213 NO_UNLOAD_STUB(kcf, crypto_get_provinfo, nomod_minus_one);
1214 NO_UNLOAD_STUB(kcf, crypto_release_provider, nomod_minus_one);
1215 NO_UNLOAD_STUB(kcf, crypto_sign, nomod_minus_one);
1216 NO_UNLOAD_STUB(kcf, crypto_sign_prov, nomod_minus_one);
1217 NO_UNLOAD_STUB(kcf, crypto_sign_init, nomod_minus_one);
1218 NO_UNLOAD_STUB(kcf, crypto_sign_init_prov, nomod_minus_one);
1219 NO_UNLOAD_STUB(kcf, crypto_sign_update, nomod_minus_one);
1220 NO_UNLOAD_STUB(kcf, crypto_sign_final, nomod_minus_one);
1221 NO_UNLOAD_STUB(kcf, crypto_sign_recover, nomod_minus_one);
1222 NO_UNLOAD_STUB(kcf, crypto_sign_recover_prov, nomod_minus_one);
1223 NO_UNLOAD_STUB(kcf, crypto_sign_recover_init_prov, nomod_minus_one);
1224 NO_UNLOAD_STUB(kcf, crypto_verify, nomod_minus_one);
1225 NO_UNLOAD_STUB(kcf, crypto_verify_prov, nomod_minus_one);
1226 NO_UNLOAD_STUB(kcf, crypto_verify_init, nomod_minus_one);
1227 NO_UNLOAD_STUB(kcf, crypto_verify_init_prov, nomod_minus_one);
1228 NO_UNLOAD_STUB(kcf, crypto_verify_update, nomod_minus_one);
1229 NO_UNLOAD_STUB(kcf, crypto_verify_final, nomod_minus_one);
1230 NO_UNLOAD_STUB(kcf, crypto_verify_recover, nomod_minus_one);
1231 NO_UNLOAD_STUB(kcf, crypto_verify_recover_prov, nomod_minus_one);
1232 NO_UNLOAD_STUB(kcf, crypto_verify_recover_init_prov, nomod_minus_one);
1233 NO_UNLOAD_STUB(kcf, random_add_entropy, nomod_minus_one);
1234 NO_UNLOAD_STUB(kcf, random_add_pseudo_entropy, nomod_minus_one);
1235 NO_UNLOAD_STUB(kcf, random_get_blocking_bytes, nomod_minus_one);
1236 NO_UNLOAD_STUB(kcf, random_get_bytes, nomod_minus_one);
1237 NO_UNLOAD_STUB(kcf, random_get_pseudo_bytes, nomod_minus_one);
1238 END_MODULE(kcf);
1239 #endif

1241 /*
1242  * Stubs for sha1. A non-unloadable module.
1243  */
1244 #ifndef SHA1_MODULE
1245 MODULE(sha1,crypto);
1246 NO_UNLOAD_STUB(sha1, SHA1Init, nomod_void);
1247 NO_UNLOAD_STUB(sha1, SHA1Update, nomod_void);
1248 NO_UNLOAD_STUB(sha1, SHA1Final, nomod_void);
1249 END_MODULE(sha1);
1250 #endif

1252 /*
1253  * The following stubs are used by the mac module.
1254  * Since dld already depends on mac, these
1255  * stubs are needed to avoid circular dependencies.
1256  */
1257 #ifndef DLD_MODULE
1258 MODULE(dld,drv);
1259 STUB(dld, dld_init_ops, nomod_void);
1260 STUB(dld, dld_fini_ops, nomod_void);
1261 STUB(dld, dld_devt_to_instance, nomod_minus_one);
1262 STUB(dld, dld_autopush, nomod_minus_one);
1263 STUB(dld, dld_ioc_register, nomod_einval);
1264 STUB(dld, dld_ioc_unregister, nomod_void);
1265 END_MODULE(dld);
1266 #endif

1268 /*
1269  * The following stubs are used by the mac module.
1270  * Since dls already depends on mac, these
1271  * stubs are needed to avoid circular dependencies.
1272  */
1273 #ifndef DLS_MODULE
1274 MODULE(dls,misc);
1275 STUB(dls, dls_devnet_mac, nomod_zero);

```

```

1276     STUB(dls, dls_devnet_hold_tmp, nomod_einval);
1277     STUB(dls, dls_devnet_rele_tmp, nomod_void);
1278     STUB(dls, dls_devnet_hold_link, nomod_einval);
1279     STUB(dls, dls_devnet_rele_link, nomod_void);
1280     STUB(dls, dls_devnet_prop_task_wait, nomod_void);
1281     STUB(dls, dls_mgmt_get_linkid, nomod_einval);
1282     STUB(dls, dls_devnet_macname2linkid, nomod_einval);
1283     STUB(dls, dls_mgmt_get_linkinfo, nomod_einval);
1284     END_MODULE(dls);
1285 #endif

1287 #ifndef SOFTMAC_MODULE
1288     MODULE(softmac,drv);
1289     STUB(softmac, softmac_hold_device, nomod_einval);
1290     STUB(softmac, softmac_rele_device, nomod_void);
1291     STUB(softmac, softmac_recreate, nomod_void);
1292     END_MODULE(softmac);
1293 #endif

1295 #ifndef IPTUN_MODULE
1296     MODULE(iptun,drv);
1297     STUB(iptun, iptun_create, nomod_einval);
1298     STUB(iptun, iptun_delete, nomod_einval);
1299     STUB(iptun, iptun_set_policy, nomod_void);
1300     END_MODULE(iptun);
1301 #endif

1303 /*
1304  * Stubs for dcopy, for Intel IOAT KAPIs
1305  */
1306 #ifndef DCOPY_MODULE
1307     MODULE(dcopy,misc);
1308     NO_UNLOAD_STUB(dcopy, dcopy_query, nomod_minus_one);
1309     NO_UNLOAD_STUB(dcopy, dcopy_query_channel, nomod_minus_one);
1310     NO_UNLOAD_STUB(dcopy, dcopy_alloc, nomod_minus_one);
1311     NO_UNLOAD_STUB(dcopy, dcopy_free, nomod_minus_one);
1312     NO_UNLOAD_STUB(dcopy, dcopy_cmd_alloc, nomod_minus_one);
1313     NO_UNLOAD_STUB(dcopy, dcopy_cmd_free, nomod_void);
1314     NO_UNLOAD_STUB(dcopy, dcopy_cmd_post, nomod_minus_one);
1315     NO_UNLOAD_STUB(dcopy, dcopy_cmd_poll, nomod_minus_one);
1316     END_MODULE(dcopy);
1317 #endif

1319 /*
1320  * Stubs for acpica
1321  */
1322 #ifndef ACPICA_MODULE
1323     MODULE(acpica,misc);
1324     NO_UNLOAD_STUB(acpica, AcpiOsReadPort, nomod_minus_one);
1325     NO_UNLOAD_STUB(acpica, AcpiOsWritePort, nomod_minus_one);
1326     NO_UNLOAD_STUB(acpica, AcpiInstallNotifyHandler, nomod_minus_one);
1327     NO_UNLOAD_STUB(acpica, AcpiRemoveNotifyHandler, nomod_minus_one);
1328     NO_UNLOAD_STUB(acpica, AcpiEvaluateObject, nomod_minus_one);
1329     NO_UNLOAD_STUB(acpica, AcpiEvaluateObjectTyped, nomod_minus_one);
1330     NO_UNLOAD_STUB(acpica, AcpiWriteBitRegister, nomod_minus_one);
1331     NO_UNLOAD_STUB(acpica, AcpiReadBitRegister, nomod_minus_one);
1332     NO_UNLOAD_STUB(acpica, AcpiOsFree, nomod_minus_one);
1333     NO_UNLOAD_STUB(acpica, acpica_get_handle_cpu, nomod_minus_one);
1334     NO_UNLOAD_STUB(acpica, acpica_get_global_FADT, nomod_minus_one);
1335     NO_UNLOAD_STUB(acpica, acpica_write_cpupm_capabilities,
1336                    nomod_minus_one);
1337     NO_UNLOAD_STUB(acpica, __acpi_wbinvd, nomod_minus_one);
1338     NO_UNLOAD_STUB(acpica, acpi_reset_system, nomod_minus_one);
1339     END_MODULE(acpica);
1340 #endif

```

```

1342 /*
1343  * Stubs for acpidev
1344  */
1345 #ifndef ACPIDEV_MODULE
1346     MODULE(acpidev,misc);
1347     NO_UNLOAD_STUB(acpidev, acpidev_dr_get_cpu_numa_info, nomod_minus_one);
1348     NO_UNLOAD_STUB(acpidev, acpidev_dr_free_cpu_numa_info,
1349                    nomod_minus_one);
1350     END_MODULE(acpidev);
1351 #endif

1353 #ifndef IPNET_MODULE
1354     MODULE(ipnet,drv);
1355     STUB(ipnet, ipnet_if_getdev, nomod_zero);
1356     STUB(ipnet, ipnet_walk_if, nomod_zero);
1357     END_MODULE(ipnet);
1358 #endif

1360 #ifndef IOMMULIB_MODULE
1361     MODULE(iommulib,misc);
1362     STUB(iommulib, iommulib_nex_close, nomod_void);
1363     END_MODULE(iommulib);
1364 #endif

1366 /*
1367  * Stubs for rootnex nexus driver.
1368  */
1369 #ifndef ROOTNEX_MODULE
1370     MODULE(rootnex,drv);
1371     STUB(rootnex, immu_init, 0);
1372     STUB(rootnex, immu_startup, 0);
1373     STUB(rootnex, immu_physmem_update, 0);
1374     END_MODULE(rootnex);
1375 #endif

1377 /*
1378  * Stubs for kernel socket, for iscsi
1379  */
1380 #ifndef KSOCKET_MODULE
1381     MODULE(ksocket, misc);
1382     NO_UNLOAD_STUB(ksocket, ksocket_setsockopt, nomod_minus_one);
1383     NO_UNLOAD_STUB(ksocket, ksocket_getsockopt, nomod_minus_one);
1384     NO_UNLOAD_STUB(ksocket, ksocket_getpeername, nomod_minus_one);
1385     NO_UNLOAD_STUB(ksocket, ksocket_getsockname, nomod_minus_one);
1386     NO_UNLOAD_STUB(ksocket, ksocket_socket, nomod_minus_one);
1387     NO_UNLOAD_STUB(ksocket, ksocket_bind, nomod_minus_one);
1388     NO_UNLOAD_STUB(ksocket, ksocket_listen, nomod_minus_one);
1389     NO_UNLOAD_STUB(ksocket, ksocket_accept, nomod_minus_one);
1390     NO_UNLOAD_STUB(ksocket, ksocket_connect, nomod_minus_one);
1391     NO_UNLOAD_STUB(ksocket, ksocket_recv, nomod_minus_one);
1392     NO_UNLOAD_STUB(ksocket, ksocket_recvfrom, nomod_minus_one);
1393     NO_UNLOAD_STUB(ksocket, ksocket_recvmsg, nomod_minus_one);
1394     NO_UNLOAD_STUB(ksocket, ksocket_send, nomod_minus_one);
1395     NO_UNLOAD_STUB(ksocket, ksocket_sendto, nomod_minus_one);
1396     NO_UNLOAD_STUB(ksocket, ksocket_sendmsg, nomod_minus_one);
1397     NO_UNLOAD_STUB(ksocket, ksocket_ioctl, nomod_minus_one);
1398     NO_UNLOAD_STUB(ksocket, ksocket_setcallbacks, nomod_minus_one);
1399     NO_UNLOAD_STUB(ksocket, ksocket_hold, nomod_minus_one);
1400     NO_UNLOAD_STUB(ksocket, ksocket_rele, nomod_minus_one);
1401     NO_UNLOAD_STUB(ksocket, ksocket_shutdown, nomod_minus_one);
1402     NO_UNLOAD_STUB(ksocket, ksocket_close, nomod_minus_one);
1403     END_MODULE(ksocket);
1404 #endif

1406 /*
1407  * Stubs for elfexec

```

```
1408 */
1409 #ifndef ELFEXEC_MODULE
1410     MODULE(elfexec,exec);
1411     STUB(elfexec, elfexec,          nomod_einval);
1412     STUB(elfexec, mapexec_brand,    nomod_einval);
1413 #if defined(__amd64)
1414     STUB(elfexec, elf32exec,        nomod_einval);
1415     STUB(elfexec, mapexec32_brand,  nomod_einval);
1416 #endif
1417     END_MODULE(elfexec);
1418 #endif

1420 /*
1421  * Stub(s) for APIX module.
1422  */
1423 #ifndef APIX_MODULE
1424     MODULE(apix,mach);
1425     WSTUB(apix, apix_loaded, nomod_zero);
1426     END_MODULE(apix);
1427 #endif

1429 / this is just a marker for the area of text that contains stubs

1431     ENTRY_NP(stubs_end)
1432     nop

1434 #endif /* lint */
```

```

*****
3376 Thu Jun 30 22:03:53 2016
new/usr/src/uts/intel/os/name_to_sysnum
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
1 nosys 0
2 rexit 1
3 psecflags 2
4 #endif /* ! codereview */
5 read 3
6 write 4
7 open 5
8 close 6
9 linkat 7
10 link 9
11 unlink 10
12 symlinkat 11
13 chdir 12
14 gtime 13
15 mknod 14
16 chmod 15
17 chown 16
18 brk 17
19 stat 18
20 lseek 19
21 getpid 20
22 mount 21
23 readlinkat 22
24 setuid 23
25 getuid 24
26 stime 25
27 pcsample 26
28 alarm 27
29 fstat 28
30 pause 29
31 stty 31
32 gtty 32
33 access 33
34 nice 34
35 statfs 35
36 syssync 36
37 kill 37
38 fstatfs 38
39 setpgrp 39
40 ucopystr 40
41 pipe 42
42 times 43
43 profil 44
44 faccessat 45
45 setgid 46
46 getgid 47
47 mkmodat 48
48 msgsys 49
49 sysi86 50
50 sysacct 51
51 shmsys 52
52 semsys 53
53 ioctl 54
54 uadmin 55
55 fchowmat 56
56 utssys 57
57 fdsync 58
58 exece 59

```

```

59 umask 60
60 chroot 61
61 fcntl 62
62 ulimit 63
63 renameat 64
64 unlinkat 65
65 fstatat 66
66 fstatat64 67
67 openat 68
68 openat64 69
69 tasksys 70
70 acctctl 71
71 exacctsys 72
72 getpagesizes 73
73 rctlsys 74
74 sidsys 75
75 lwp_park 77
76 sendfilev 78
77 rmdir 79
78 mkdir 80
79 getdents 81
80 privsys 82
81 ucredsys 83
82 sysfs 84
83 getmsg 85
84 putmsg 86
85 lstat 88
86 symlink 89
87 readlink 90
88 setgroups 91
89 getgroups 92
90 fchmod 93
91 fchown 94
92 sigprocmask 95
93 sigsuspend 96
94 sigaltstack 97
95 sigaction 98
96 sigpending 99
97 setcontext 100
98 fchmodat 101
99 mkdirat 102
100 statvfs 103
101 fstatvfs 104
102 getloadavg 105
103 nfs 106
104 waitsys 107
105 sigsendsys 108
106 hrtsys 109
107 utimesys 110
108 sigresend 111
109 priocntlsys 112
110 pathconf 113
111 mincore 114
112 mmap 115
113 mprotect 116
114 munmap 117
115 fpathconf 118
116 vfork 119
117 fchdir 120
118 readv 121
119 writev 122
120 preadv 123
121 pwritev 124
122 getrandom 126
123 mmapobj 127
124 setrlimit 128

```


125	getrlimit	129
126	lchown	130
127	memcntl	131
128	getpmsg	132
129	putpmsg	133
130	rename	134
131	uname	135
132	setegid	136
133	sysconfig	137
134	adjtime	138
135	systeminfo	139
136	sharefs	140
137	seteuid	141
138	forksys	142
139	sigwait	144
140	lwp_info	145
141	yield	146
142	lwp_sema_post	148
143	lwp_sema_trywait	149
144	lwp_detach	150
145	corectl	151
146	modctl	152
147	fchroot	153
148	vhangup	155
149	gettimeofday	156
150	getitimer	157
151	setitimer	158
152	lwp_create	159
153	lwp_exit	160
154	lwp_suspend	161
155	lwp_continue	162
156	lwp_kill	163
157	lwp_self	164
158	lwp_sigmask	165
159	lwp_wait	167
160	lwp_mutex_wakeup	168
161	lwp_cond_wait	170
162	lwp_cond_signal	171
163	lwp_cond_broadcast	172
164	pread	173
165	pwrite	174
166	llseek	175
167	inst_sync	176
168	brandsys	177
169	kaio	178
170	cpc	179
171	meminfosys	180
172	rusagesys	181
173	portfs	182
174	pollsys	183
175	labelsys	184
176	acl	185
177	c2audit	186
178	processor_bind	187
179	processor_info	188
180	p_online	189
181	sigqueue	190
182	clock_gettime	191
183	clock_gettime	192
184	clock_getres	193
185	timer_create	194
186	timer_delete	195
187	timer_settime	196
188	timer_gettime	197
189	timer_getoverrun	198
190	nanosleep	199

191	facl	200
192	doorfs	201
193	setreuid	202
194	setregid	203
195	install_ustrap	204
196	signotify	205
197	schedctl	206
198	pset	207
199	resolvepath	209
200	lwp_mutex_timedlock	210
201	lwp_sema_timedwait	211
202	lwp_rwlock_sys	212
203	getdents64	213
204	mmap64	214
205	stat64	215
206	lstat64	216
207	fstat64	217
208	statvfs64	218
209	fstatvfs64	219
210	setrlimit64	220
211	getrlimit64	221
212	pread64	222
213	pwrite64	223
214	open64	225
215	rpcmod	226
216	zone	227
217	autofs	228
218	getcwd	229
219	so_socket	230
220	so_socketpair	231
221	bind	232
222	listen	233
223	accept	234
224	connect	235
225	shutdown	236
226	recv	237
227	recvfrom	238
228	recvmsg	239
229	send	240
230	sendmsg	241
231	sendto	242
232	getpeername	243
233	getsockname	244
234	getsockopt	245
235	setsockopt	246
236	sockconfig	247
237	ntp_gettime	248
238	ntp_adjtime	249
239	lwp_mutex_unlock	250
240	lwp_mutex_trylock	251
241	lwp_mutex_register	252
242	cladm	253
243	uucopy	254
244	umount2	255

new/usr/src/uts/req.flg

1

```
*****
2103 Thu Jun 30 22:03:54 2016
new/usr/src/uts/req.flg
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
1 #!/bin/sh
2 #
3 #
4 # CDDL HEADER START
5 #
6 # The contents of this file are subject to the terms of the
7 # Common Development and Distribution License (the "License").
8 # You may not use this file except in compliance with the License.
9 #
10 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
11 # or http://www.opensolaris.org/os/licensing.
12 # See the License for the specific language governing permissions
13 # and limitations under the License.
14 #
15 # When distributing Covered Code, include this CDDL HEADER in each
16 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
17 # If applicable, add the following below this CDDL HEADER, with the
18 # fields enclosed by brackets "[]" replaced with your own identifying
19 # information: Portions Copyright [yyyy] [name of copyright owner]
20 #
21 # CDDL HEADER END
22 #
23 # Copyright 2007 Sun Microsystems, Inc. All rights reserved.
24 # Use is subject to license terms.
25 #
26 #ident "%Z%M% %I% %E% SMI"

27 #
28 # Please KEEP THIS (reasonably) ALPHABETIZED BY FILENAME
29 # (to make merging saner)
30 #

32 echo_file usr/src/uts/Makefile
33 echo_file usr/src/uts/Makefile.targ
34 echo_file usr/src/uts/Makefile.uts
35 echo_file usr/src/Targetdirs
36 echo_file usr/src/Makefile

38 # For full builds (open and closed), we want both etc/certs and
39 # etc/keys. For an open source build, there's no etc/keys directory.
40 find_files "s.*" usr/src/cmd/cmd-crypto/etc
41 find_files "s.*" usr/src/common/acl
42 find_files "s.*" usr/src/common/atomic
43 find_files "s.*" usr/src/common/avl
44 find_files "s.*" usr/src/common/bignum
45 find_files "s.*" usr/src/common/crypto
46 find_files "s.*" usr/src/common/ctf
47 find_files "s.*" usr/src/common/devid
48 find_files "s.*" usr/src/common/exacct
49 find_files "s.*" usr/src/common/fs
50 find_files "s.*" usr/src/common/mapfiles
51 find_files "s.*" usr/src/common/mdesc
52 find_files "s.*" usr/src/common/net/wanboot/crypt
53 find_files "s.*" usr/src/common/nvpair
54 find_files "s.*" usr/src/common/pci
55 find_files "s.*" usr/src/common/secflags
56 #endif /* !codereview */
57 find_files "s.*" usr/src/common/smbios
```

new/usr/src/uts/req.flg

2

```
58 find_files "s.*" usr/src/common/tsol
59 find_files "s.*" usr/src/common/util
60 find_files "s.*" usr/src/common/zfs
61 find_files "s.*" usr/src/psm/promif
```

```

*****
43451 Thu Jun 30 22:03:55 2016
new/usr/src/uts/sparc/ml/modstubs.s
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
    unchanged_portion_omitted

274 ! this is just a marker for the area of text that contains stubs
275     .seg ".text"
276     .global stubs_base
277 stubs_base:
278     nop

280 /*
281  * WARNING WARNING WARNING!!!!!!
282  *
283  * On the MODULE macro you MUST NOT use any spaces!!! They are
284  * significant to the preprocessor.  With ansi c there is a way around this
285  * but for some reason (yet to be investigated) ansi didn't work for other
286  * reasons!
287  *
288  * When zero is used as the return function, the system will call
289  * panic if the stub can't be resolved.
290  */

292 /*
293  * Stubs for devfs. A non-unloadable module.
294  */
295 #ifndef DEVFS_MODULE
296     MODULE(devfs,fs);
297     NO_UNLOAD_STUB(devfs, devfs_clean,          nomod_minus_one);
298     NO_UNLOAD_STUB(devfs, devfs_lookupname,     nomod_minus_one);
299     NO_UNLOAD_STUB(devfs, devfs_walk,          nomod_minus_one);
300     NO_UNLOAD_STUB(devfs, devfs_devpolicy,     nomod_minus_one);
301     NO_UNLOAD_STUB(devfs, devfs_reset_perm,    nomod_minus_one);
302     NO_UNLOAD_STUB(devfs, devfs_remdrv_cleanup, nomod_minus_one);
303     END_MODULE(devfs);
304 #endif

306 /*
307  * Stubs for /dev fs.
308  */
309 #ifndef DEV_MODULE
310     MODULE(dev, fs);
311     NO_UNLOAD_STUB(dev, sdev_modctl_readdir,   nomod_minus_one);
312     NO_UNLOAD_STUB(dev, sdev_modctl_readdir_free, nomod_minus_one);
313     NO_UNLOAD_STUB(dev, devname_filename_register, nomod_minus_one);
314     NO_UNLOAD_STUB(dev, sdev_modctl_devexists, nomod_minus_one);
315     NO_UNLOAD_STUB(dev, devname_profile_update, nomod_minus_one);
316     NO_UNLOAD_STUB(dev, sdev_devstate_change,  nomod_minus_one);
317     NO_UNLOAD_STUB(dev, devvt_getvnodeops,    nomod_minus_one);
318     NO_UNLOAD_STUB(dev, devpts_getvnodeops,   nomod_zero);
319     END_MODULE(dev);
320 #endif

322 /*
323  * Stubs for specfs. A non-unloadable module.
324  */

326 #ifndef SPEC_MODULE
327     MODULE(specfs,fs);
328     NO_UNLOAD_STUB(specfs, common_specvp,     nomod_zero);
329     NO_UNLOAD_STUB(specfs, makectty,         nomod_zero);

```

```

330     NO_UNLOAD_STUB(specfs, makespecvp,       nomod_zero);
331     NO_UNLOAD_STUB(specfs, smark,            nomod_zero);
332     NO_UNLOAD_STUB(specfs, spec_segmap,      nomod_einval);
333     NO_UNLOAD_STUB(specfs, specfind,        nomod_zero);
334     NO_UNLOAD_STUB(specfs, specvp,          nomod_zero);
335     NO_UNLOAD_STUB(specfs, devi_stillreferenced, nomod_zero);
336     NO_UNLOAD_STUB(specfs, spec_getvnodeops, nomod_zero);
337     NO_UNLOAD_STUB(specfs, spec_char_map,   nomod_zero);
338     NO_UNLOAD_STUB(specfs, specvp_devfs,    nomod_zero);
339     NO_UNLOAD_STUB(specfs, spec_assoc_vp_with_devi, nomod_void);
340     NO_UNLOAD_STUB(specfs, spec_hold_devi_by_vp, nomod_zero);
341     NO_UNLOAD_STUB(specfs, spec_snode_walk,  nomod_void);
342     NO_UNLOAD_STUB(specfs, spec_devi_open_count, nomod_minus_one);
343     NO_UNLOAD_STUB(specfs, spec_is_clone,   nomod_zero);
344     NO_UNLOAD_STUB(specfs, spec_is_selfclone, nomod_zero);
345     NO_UNLOAD_STUB(specfs, spec_fence_snode, nomod_minus_one);
346     NO_UNLOAD_STUB(specfs, spec_unfence_snode, nomod_minus_one);
347     END_MODULE(specfs);
348 #endif

351 /*
352  * Stubs for sockfs. A non-unloadable module.
353  */
354 #ifndef SOCK_MODULE
355     MODULE(sockfs, fs);
356     SCALL_NU_STUB(sockfs, so_socket,          nomod_zero);
357     SCALL_NU_STUB(sockfs, so_socketpair,     nomod_zero);
358     SCALL_NU_STUB(sockfs, bind,              nomod_zero);
359     SCALL_NU_STUB(sockfs, listen,            nomod_zero);
360     SCALL_NU_STUB(sockfs, accept,            nomod_zero);
361     SCALL_NU_STUB(sockfs, connect,           nomod_zero);
362     SCALL_NU_STUB(sockfs, shutdown,         nomod_zero);
363     SCALL_NU_STUB(sockfs, recv,              nomod_zero);
364     SCALL_NU_STUB(sockfs, recvfrom,         nomod_zero);
365     SCALL_NU_STUB(sockfs, recvmsg,          nomod_zero);
366     SCALL_NU_STUB(sockfs, send,              nomod_zero);
367     SCALL_NU_STUB(sockfs, sendmsg,          nomod_zero);
368     SCALL_NU_STUB(sockfs, sendto,           nomod_zero);
369 #ifdef _SYSCALL32_IMPL
370     SCALL_NU_STUB(sockfs, recv32,            nomod_zero);
371     SCALL_NU_STUB(sockfs, recvfrom32,        nomod_zero);
372     SCALL_NU_STUB(sockfs, send32,            nomod_zero);
373     SCALL_NU_STUB(sockfs, sendto32,          nomod_zero);
374 #endif /* _SYSCALL32_IMPL */
375     SCALL_NU_STUB(sockfs, getpeername,      nomod_zero);
376     SCALL_NU_STUB(sockfs, getsockname,      nomod_zero);
377     SCALL_NU_STUB(sockfs, getsockopt,       nomod_zero);
378     SCALL_NU_STUB(sockfs, setsockopt,       nomod_zero);
379     SCALL_NU_STUB(sockfs, sockconfig,       nomod_zero);
380     NO_UNLOAD_STUB(sockfs, sock_getmsg,    nomod_zero);
381     NO_UNLOAD_STUB(sockfs, sock_putmsg,     nomod_zero);
382     NO_UNLOAD_STUB(sockfs, sosendfile64,    nomod_zero);
383     NO_UNLOAD_STUB(sockfs, snf_segmap,      nomod_einval);
384     NO_UNLOAD_STUB(sockfs, sock_getfasync,  nomod_zero);
385     NO_UNLOAD_STUB(sockfs, nl7c_sendfilev,  nomod_zero);
386     NO_UNLOAD_STUB(sockfs, sotpi_sototpi,   nomod_zero);
387     NO_UNLOAD_STUB(sockfs, socket_sendmblk, nomod_zero);
388     NO_UNLOAD_STUB(sockfs, socket_setsockopt, nomod_zero);
389     END_MODULE(sockfs);
390 #endif

392 /*
393  * IPsec stubs.
394  */

```

```

396 #ifndef IPSECAH_MODULE
397     MODULE(ipsecah,drv);
398     WSTUB(ipsecah, ipsec_construct_inverse_acquire,    nomod_zero);
399     WSTUB(ipsecah, sadb_acquire,                    nomod_zero);
400     WSTUB(ipsecah, ipsecah_algs_changed,            nomod_zero);
401     WSTUB(ipsecah, sadb_alg_update,                nomod_zero);
402     WSTUB(ipsecah, sadb_unlinkassoc,               nomod_zero);
403     WSTUB(ipsecah, sadb_insertassoc,               nomod_zero);
404     WSTUB(ipsecah, ipsecah_in_assocfailure,        nomod_zero);
405     WSTUB(ipsecah, sadb_set_lpkt,                  nomod_zero);
406     WSTUB(ipsecah, ipsecah_icmp_error,            nomod_zero);
407     END_MODULE(ipsecah);
408 #endif

410 #ifndef IPSECESP_MODULE
411     MODULE(ipsecesp,drv);
412     WSTUB(ipsecesp, ipsecesp_fill_defs,            nomod_zero);
413     WSTUB(ipsecesp, ipsecesp_algs_changed,         nomod_zero);
414     WSTUB(ipsecesp, ipsecesp_in_assocfailure,      nomod_zero);
415     WSTUB(ipsecesp, ipsecesp_init_funcs,          nomod_zero);
416     WSTUB(ipsecesp, ipsecesp_icmp_error,          nomod_zero);
417     WSTUB(ipsecesp, ipsecesp_send_keepalive,      nomod_zero);
418     END_MODULE(ipsecesp);
419 #endif

421 #ifndef KEYSOCK_MODULE
422     MODULE(keysock,drv);
423     WSTUB(keysock, keysock_plumb_ipsec,            nomod_zero);
424     WSTUB(keysock, keysock_extended_reg,          nomod_zero);
425     WSTUB(keysock, keysock_next_seq,              nomod_zero);
426     END_MODULE(keysock);
427 #endif

429 #ifndef SPDSOCK_MODULE
430     MODULE(spdssock,drv);
431     WSTUB(spdssock, spdssock_update_pending_algs, nomod_zero);
432     END_MODULE(spdssock);
433 #endif

435 /*
436  * Stubs for nfs common code.
437  * XXX nfs_getvnodeops should go away with removal of kludge in vnode.c
438  */
439 #ifndef NFS_MODULE
440     MODULE(nfs,fs);
441     WSTUB(nfs,    nfs_getvnodeops,    nomod_zero);
442     WSTUB(nfs,    nfs_perror,         nomod_zero);
443     WSTUB(nfs,    nfs_cmh_err,        nomod_zero);
444     WSTUB(nfs,    clcleanup_zone,     nomod_zero);
445     WSTUB(nfs,    clcleanup4_zone,    nomod_zero);
446     END_MODULE(nfs);
447 #endif

449 /*
450  * Stubs for nfs_dlboot (diskless booting).
451  */
452 #ifndef NFS_DLBOOT_MODULE
453     MODULE(nfs_dlboot,misc);
454     STUB(nfs_dlboot,    mount_root,    nomod_minus_one);
455     STUB(nfs_dlboot,    dhcpinit,     nomod_minus_one);
456     END_MODULE(nfs_dlboot);
457 #endif

459 /*
460  * Stubs for nfs server-only code.
461  */

```

```

462 #ifndef NFSSRV_MODULE
463     MODULE(nfssrv,misc);
464     STUB(nfssrv,    exportfs,        nomod_minus_one);
465     STUB(nfssrv,    nfs_getfh,       nomod_minus_one);
466     STUB(nfssrv,    nfs_sl_flush,    nomod_minus_one);
467     STUB(nfssrv,    rfs4_check_delegated, nomod_zero);
468     STUB(nfssrv,    mountd_args,     nomod_minus_one);
469     NO_UNLOAD_STUB(nfssrv,    rdma_start,    nomod_zero);
470     NO_UNLOAD_STUB(nfssrv,    nfs_svc,      nomod_zero);
471     END_MODULE(nfssrv);
472 #endif

474 /*
475  * Stubs for kernel lock manager.
476  */
477 #ifndef KLM_MODULE
478     MODULE(klmmmod,misc);
479     NO_UNLOAD_STUB(klmmmod,    lm_svc,        nomod_zero);
480     NO_UNLOAD_STUB(klmmmod,    lm_shutdown,   nomod_zero);
481     NO_UNLOAD_STUB(klmmmod,    lm_unexport,   nomod_zero);
482     NO_UNLOAD_STUB(klmmmod,    lm_cprresume,  nomod_zero);
483     NO_UNLOAD_STUB(klmmmod,    lm_cprsuspend, nomod_zero);
484     NO_UNLOAD_STUB(klmmmod,    lm_safelock,   nomod_zero);
485     NO_UNLOAD_STUB(klmmmod,    lm_safemap,   nomod_zero);
486     NO_UNLOAD_STUB(klmmmod,    lm_has_sleep,  nomod_zero);
487     NO_UNLOAD_STUB(klmmmod,    lm_free_config, nomod_zero);
488     NO_UNLOAD_STUB(klmmmod,    lm_vp_active,  nomod_zero);
489     NO_UNLOAD_STUB(klmmmod,    lm_get_sysid,  nomod_zero);
490     NO_UNLOAD_STUB(klmmmod,    lm_rel_sysid,  nomod_zero);
491     NO_UNLOAD_STUB(klmmmod,    lm_alloc_sysidt, nomod_minus_one);
492     NO_UNLOAD_STUB(klmmmod,    lm_free_sysidt, nomod_zero);
493     NO_UNLOAD_STUB(klmmmod,    lm_sysidt,     nomod_minus_one);
494     END_MODULE(klmmmod);
495 #endif

497 #ifndef KLMOPS_MODULE
498     MODULE(klmops,misc);
499     NO_UNLOAD_STUB(klmops,    lm_frlock,      nomod_zero);
500     NO_UNLOAD_STUB(klmops,    lm4_frlock,     nomod_zero);
501     NO_UNLOAD_STUB(klmops,    lm_shrlock,     nomod_zero);
502     NO_UNLOAD_STUB(klmops,    lm4_shrlock,    nomod_zero);
503     NO_UNLOAD_STUB(klmops,    lm_nlm_dispatch, nomod_zero);
504     NO_UNLOAD_STUB(klmops,    lm_nlm4_dispatch, nomod_zero);
505     NO_UNLOAD_STUB(klmops,    lm_nlm_reclaim, nomod_zero);
506     NO_UNLOAD_STUB(klmops,    lm_nlm4_reclaim, nomod_zero);
507     NO_UNLOAD_STUB(klmops,    lm_register_lock_locally, nomod_zero);
508     END_MODULE(klmops);
509 #endif

511 /*
512  * Stubs for kernel TLI module
513  * XXX currently we never allow this to unload
514  */
515 #ifndef TLI_MODULE
516     MODULE(tlimod,misc);
517     NO_UNLOAD_STUB(tlimod,    t_kopen,        nomod_minus_one);
518     NO_UNLOAD_STUB(tlimod,    t_kunbind,      nomod_zero);
519     NO_UNLOAD_STUB(tlimod,    t_kadvise,      nomod_zero);
520     NO_UNLOAD_STUB(tlimod,    t_krcvudata,    nomod_zero);
521     NO_UNLOAD_STUB(tlimod,    t_ksndudata,    nomod_zero);
522     NO_UNLOAD_STUB(tlimod,    t_kalloc,       nomod_zero);
523     NO_UNLOAD_STUB(tlimod,    t_kbind,        nomod_zero);
524     NO_UNLOAD_STUB(tlimod,    t_kclose,       nomod_zero);
525     NO_UNLOAD_STUB(tlimod,    t_kspoll,       nomod_zero);
526     NO_UNLOAD_STUB(tlimod,    t_kfree,        nomod_zero);
527     END_MODULE(tlimod);

```

```

528 #endif

530 /*
531  * Stubs for kernel RPC module
532  * XXX currently we never allow this to unload
533  */
534 #ifndef RPC_MODULE
535     MODULE(rpcmod, strmod);
536     NO_UNLOAD_STUB(rpcmod,  clnt_tli_kcreate,      nomod_minus_one);
537     NO_UNLOAD_STUB(rpcmod,  svc_tli_kcreate,      nomod_minus_one);
538     NO_UNLOAD_STUB(rpcmod,  bindresvport,        nomod_minus_one);
539     NO_UNLOAD_STUB(rpcmod,  rdma_register_mod,    nomod_minus_one);
540     NO_UNLOAD_STUB(rpcmod,  rdma_unregister_mod,  nomod_minus_one);
541     NO_UNLOAD_STUB(rpcmod,  svc_queureq,        nomod_minus_one);
542     NO_UNLOAD_STUB(rpcmod,  clist_add,          nomod_minus_one);
543     END_MODULE(rpcmod);
544 #endif

546 /*
547  * Stubs for des
548  */
549 #ifndef DES_MODULE
550     MODULE(des, misc);
551     STUB(des,  cbc_crypt,          nomod_zero);
552     STUB(des,  ecb_crypt,         nomod_zero);
553     STUB(des,  _des_crypt,        nomod_zero);
554     END_MODULE(des);
555 #endif

557 /*
558  * Stubs for procfs. A non-unloadable module.
559  */
560 #ifndef PROC_MODULE
561     MODULE(procfs, fs);
562     NO_UNLOAD_STUB(procfs,  prfree,          nomod_zero);
563     NO_UNLOAD_STUB(procfs,  prexit,          nomod_zero);
564     NO_UNLOAD_STUB(procfs,  prlwpfree,      nomod_zero);
565     NO_UNLOAD_STUB(procfs,  prlwpexit,      nomod_zero);
566     NO_UNLOAD_STUB(procfs,  prinvalidate,   nomod_zero);
567     NO_UNLOAD_STUB(procfs,  prnsegs,       nomod_zero);
568     NO_UNLOAD_STUB(procfs,  prgetcred,      nomod_zero);
569     NO_UNLOAD_STUB(procfs,  prgetpriv,      nomod_zero);
570     NO_UNLOAD_STUB(procfs,  prgetprivsize,  nomod_zero);
571     NO_UNLOAD_STUB(procfs,  prgetsecflags,  nomod_zero);
572 #endif /* ! codereview */
573     NO_UNLOAD_STUB(procfs,  prgetstatus,    nomod_zero);
574     NO_UNLOAD_STUB(procfs,  prgetlwpstatus, nomod_zero);
575     NO_UNLOAD_STUB(procfs,  prgetpsinfo,    nomod_zero);
576     NO_UNLOAD_STUB(procfs,  prgetlwpinfo,   nomod_zero);
577     NO_UNLOAD_STUB(procfs,  oprgetstatus,    nomod_zero);
578     NO_UNLOAD_STUB(procfs,  oprgetpsinfo,   nomod_zero);
579 #ifdef _SYSCALL32_IMPL
580     NO_UNLOAD_STUB(procfs,  prgetstatus32,  nomod_zero);
581     NO_UNLOAD_STUB(procfs,  prgetlwpstatus32, nomod_zero);
582     NO_UNLOAD_STUB(procfs,  prgetpsinfo32,  nomod_zero);
583     NO_UNLOAD_STUB(procfs,  prgetlwpinfo32, nomod_zero);
584     NO_UNLOAD_STUB(procfs,  oprgetstatus32, nomod_zero);
585     NO_UNLOAD_STUB(procfs,  oprgetpsinfo32, nomod_zero);
586     NO_UNLOAD_STUB(procfs,  psinfo_kto32,  nomod_zero);
587     NO_UNLOAD_STUB(procfs,  lwpsinfo_kto32, nomod_zero);
588 #endif /* _SYSCALL32_IMPL */
589     NO_UNLOAD_STUB(procfs,  prnotify,       nomod_zero);
590     NO_UNLOAD_STUB(procfs,  prexecstart,    nomod_zero);
591     NO_UNLOAD_STUB(procfs,  prexecend,      nomod_zero);
592     NO_UNLOAD_STUB(procfs,  prrelvm,       nomod_zero);
593     NO_UNLOAD_STUB(procfs,  prbarrier,      nomod_zero);

```

```

594     NO_UNLOAD_STUB(procfs,  estimate_msacct, nomod_zero);
595     NO_UNLOAD_STUB(procfs,  pr_getprot,      nomod_zero);
596     NO_UNLOAD_STUB(procfs,  pr_getprot_done, nomod_zero);
597     NO_UNLOAD_STUB(procfs,  pr_getsegsz,    nomod_zero);
598     NO_UNLOAD_STUB(procfs,  pr_isobject,     nomod_zero);
599     NO_UNLOAD_STUB(procfs,  pr_isself,       nomod_zero);
600     NO_UNLOAD_STUB(procfs,  pr_allstopped,   nomod_zero);
601     NO_UNLOAD_STUB(procfs,  pr_free_watched_pages, nomod_zero);
602     END_MODULE(procfs);
603 #endif

605 /*
606  * Stubs for fifofs
607  */
608 #ifndef FIFO_MODULE
609     MODULE(fifofs, fs);
610     STUB(fifofs,  fifovp,          0);
611     STUB(fifofs,  fifo_getinfo,    0);
612     STUB(fifofs,  fifo_vfastoff,   0);
613     END_MODULE(fifofs);
614 #endif

616 /*
617  * Stubs for ufs
618  */
619  * This is needed to support the old quotactl system call.
620  * When the old sysent stuff goes away, this will need to be revisited.
621  */
622 #ifndef UFS_MODULE
623     MODULE(ufs, fs);
624     STUB(ufs,  quotactl,  nomod_minus_one);
625     STUB(ufs,  ufs_remountroot, 0);
626     END_MODULE(ufs);
627 #endif

629 /*
630  * Stubs for zfs
631  */
632 #ifndef ZFS_MODULE
633     MODULE(zfs, fs);
634     STUB(zfs,  dsl_prop_get,  nomod_minus_one);
635     STUB(zfs,  spa_boot_init, nomod_minus_one);
636     STUB(zfs,  zfs_prop_to_name, nomod_zero);
637     END_MODULE(zfs);
638 #endif

640 /*
641  * Stubs for dcfs
642  */
643 #ifndef DCFS_MODULE
644     MODULE(dcfs, fs);
645     STUB(dcfs,  decompvp, 0);
646     END_MODULE(dcfs);
647 #endif

649 /*
650  * Stubs for namefs
651  */
652 #ifndef NAMEFS_MODULE
653     MODULE(namefs, fs);
654     STUB(namefs,  nm_unmountall, 0);
655     END_MODULE(namefs);
656 #endif

658 /*
659  * Stubs for sysdc

```

```

660 */
661 #ifndef SDC_MODULE
662     MODULE(SDC,sched);
663     NO_UNLOAD_STUB(SDC, sysdc_thread_enter,      nomod_zero);
664     END_MODULE(SDC);
665 #endif
667 /*
668  * Stubs for ts_dptbl
669 */
670 #ifndef TS_DPTBL_MODULE
671     MODULE(TS_DPTBL,sched);
672     STUB(TS_DPTBL, ts_getdptbl,                0);
673     STUB(TS_DPTBL, ts_getkmdpris,            0);
674     STUB(TS_DPTBL, ts_getmaxumdprpri, 0);
675     END_MODULE(TS_DPTBL);
676 #endif
678 /*
679  * Stubs for rt_dptbl
680 */
681 #ifndef RT_DPTBL_MODULE
682     MODULE(RT_DPTBL,sched);
683     STUB(RT_DPTBL, rt_getdptbl,                0);
684     END_MODULE(RT_DPTBL);
685 #endif
687 /*
688  * Stubs for ia_dptbl
689 */
690 #ifndef IA_DPTBL_MODULE
691     MODULE(IA_DPTBL,sched);
692     STUB(IA_DPTBL, ia_getdptbl,                0);
693     STUB(IA_DPTBL, ia_getkmdpris,            0);
694     STUB(IA_DPTBL, ia_getmaxumdprpri, 0);
695     END_MODULE(IA_DPTBL);
696 #endif
698 /*
699  * Stubs for FSS scheduler
700 */
701 #ifndef FSS_MODULE
702     MODULE(FSS,sched);
703     WSTUB(FSS, fss_allocbuf,                nomod_zero);
704     WSTUB(FSS, fss_freebuf,                nomod_zero);
705     WSTUB(FSS, fss_changeproj,            nomod_zero);
706     WSTUB(FSS, fss_changeproj,            nomod_zero);
707     END_MODULE(FSS);
708 #endif
710 /*
711  * Stubs for fx_dptbl
712 */
713 #ifndef FX_DPTBL_MODULE
714     MODULE(FX_DPTBL,sched);
715     STUB(FX_DPTBL, fx_getdptbl,                0);
716     STUB(FX_DPTBL, fx_getmaxumdprpri, 0);
717     END_MODULE(FX_DPTBL);
718 #endif
720 /*
721  * Stubs for kb (only needed for 'win')
722 */
723 #ifndef KB_MODULE
724     MODULE(kb,strmod);
725     STUB(kb, strsetwithdecimal,            0);

```

```

726     END_MODULE(kb);
727 #endif
729 /*
730  * Stubs for swapgeneric
731 */
732 #ifndef SWAPGENERIC_MODULE
733     MODULE(swapgeneric,misc);
734     STUB(swapgeneric, rootconf,            0);
735     STUB(swapgeneric, svm_rootconf, 0);
736     STUB(swapgeneric, getrootdev,        0);
737     STUB(swapgeneric, getfsname,        0);
738     STUB(swapgeneric, loadrootmodules, 0);
739     END_MODULE(swapgeneric);
740 #endif
742 /*
743  * Stubs for bootdev
744 */
745 #ifndef BOOTDEV_MODULE
746     MODULE(bootdev,misc);
747     STUB(bootdev, i_devname_to_promname, 0);
748     STUB(bootdev, i_promname_to_devname, 0);
749     STUB(bootdev, i_convert_boot_device_name, 0);
750     END_MODULE(bootdev);
751 #endif
753 /*
754  * stubs for strplumb...
755 */
756 #ifndef STRPLUMB_MODULE
757     MODULE(strplumb,misc);
758     STUB(strplumb, strplumb,                0);
759     STUB(strplumb, strplumb_load, 0);
760     STUB(strplumb, strplumb_get_netdev_path, 0);
761     END_MODULE(strplumb);
762 #endif
764 /*
765  * Stubs for console configuration module
766 */
767 #ifndef CONSCONFIG_MODULE
768     MODULE(consconfig,misc);
769     STUB(consconfig, consconfig,            0);
770     STUB(consconfig, consconfig_get_usb_kb_path, 0);
771     STUB(consconfig, consconfig_get_usb_ms_path, 0);
772     STUB(consconfig, consconfig_console_is_ready, 0);
773     END_MODULE(consconfig);
774 #endif
776 /*
777  * Stubs for zs (uart) module
778 */
779 #ifndef ZS_MODULE
780     MODULE(zs,drv);
781     STUB(zs, zsgetspeed,                    0);
782     END_MODULE(zs);
783 #endif
785 /*
786  * Stubs for accounting.
787 */
788 #ifndef SYSACCT_MODULE
789     MODULE(sysacct,sys);
790     WSTUB(sysacct, acct,                    nomod_zero);
791     WSTUB(sysacct, acct_fs_in_use,        nomod_zero);

```

```

792     END_MODULE(sysacct);
793 #endif

795 /*
796 * Stubs for semaphore routines. sem.c
797 */
798 #ifndef SEMSYS_MODULE
799     MODULE(semsys,sys);
800     WSTUB(semsys,semexit,          nomod_zero);
801     END_MODULE(semsys);
802 #endif

804 /*
805 * Stubs for shmем routines. shm.c
806 */
807 #ifndef SHMSYS_MODULE
808     MODULE(shmsys,sys);
809     WSTUB(shmsys,shmexit,          nomod_zero);
810     WSTUB(shmsys,shmfork,          nomod_zero);
811     WSTUB(shmsys,shmgetid,        nomod_minus_one);
812     END_MODULE(shmsys);
813 #endif

815 /*
816 * Stubs for doors
817 */
818 #ifndef DOORFS_MODULE
819     MODULE(doorfs,sys);
820     WSTUB(doorfs,door_slam,        nomod_zero);
821     WSTUB(doorfs,door_exit,        nomod_zero);
822     WSTUB(doorfs,door_revoke_all,  nomod_zero);
823     WSTUB(doorfs,door_fork,        nomod_zero);
824     NO_UNLOAD_STUB(doorfs,door_upcall, nomod_einval);
825     NO_UNLOAD_STUB(doorfs,door_ki_create, nomod_einval);
826     NO_UNLOAD_STUB(doorfs,door_ki_open, nomod_einval);
827     NO_UNLOAD_STUB(doorfs,door_ki_lookup, nomod_einval);
828     WSTUB(doorfs,door_ki_upcall,    nomod_einval);
829     WSTUB(doorfs,door_ki_upcall_limited, nomod_einval);
830     WSTUB(doorfs,door_ki_hold,      nomod_zero);
831     WSTUB(doorfs,door_ki_rele,      nomod_zero);
832     WSTUB(doorfs,door_ki_info,      nomod_einval);
833     END_MODULE(doorfs);
834 #endif

836 /*
837 * Stubs for idmap
838 */
839 #ifndef IDMAP_MODULE
840     MODULE(idmap,misc);
841     STUB(idmap,kidmap_batch_getgidbysid, nomod_zero);
842     STUB(idmap,kidmap_batch_getpidbysid, nomod_zero);
843     STUB(idmap,kidmap_batch_getsidbygid, nomod_zero);
844     STUB(idmap,kidmap_batch_getsidbyuid, nomod_zero);
845     STUB(idmap,kidmap_batch_getuidbysid, nomod_zero);
846     STUB(idmap,kidmap_get_create,        nomod_zero);
847     STUB(idmap,kidmap_get_destroy,       nomod_zero);
848     STUB(idmap,kidmap_get_mappings,     nomod_zero);
849     STUB(idmap,kidmap_getgidbysid,      nomod_zero);
850     STUB(idmap,kidmap_getpidbysid,      nomod_zero);
851     STUB(idmap,kidmap_getsidbygid,      nomod_zero);
852     STUB(idmap,kidmap_getsidbyuid,      nomod_zero);
853     STUB(idmap,kidmap_getuidbysid,      nomod_zero);
854     STUB(idmap,idmap_get_door,          nomod_einval);
855     STUB(idmap,idmap_unreg_dh,          nomod_einval);
856     STUB(idmap,idmap_reg_dh,           nomod_einval);
857     STUB(idmap,idmap_purge_cache,       nomod_einval);

```

```

858     END_MODULE(idmap);
859 #endif

861 /*
862 * Stubs for dma routines. dmaga.c
863 * (These are only needed for cross-checks, not autoloading)
864 */
865 #ifndef DMA_MODULE
866     MODULE(dma,drv);
867     WSTUB(dma,dma_alloc,          nomod_zero); /* (DMAGA *)0 */
868     WSTUB(dma,dma_free,          nomod_zero); /* (DMAGA *)0 */
869     END_MODULE(dma);
870 #endif

872 /*
873 * Stubs for auditing.
874 */
875 #ifndef C2AUDIT_MODULE
876     MODULE(c2audit,sys);
877     NO_UNLOAD_STUB(c2audit,audit_init_module, nomod_zero);
878     NO_UNLOAD_STUB(c2audit,audit_start,        nomod_zero);
879     NO_UNLOAD_STUB(c2audit,audit_finish,       nomod_zero);
880     NO_UNLOAD_STUB(c2audit,audit,              nomod_zero);
881     NO_UNLOAD_STUB(c2audit,auditdoor,         nomod_zero);
882     NO_UNLOAD_STUB(c2audit,audit_closef,      nomod_zero);
883     NO_UNLOAD_STUB(c2audit,audit_core_start,  nomod_zero);
884     NO_UNLOAD_STUB(c2audit,audit_core_finish, nomod_zero);
885     NO_UNLOAD_STUB(c2audit,audit_strputmsg,    nomod_zero);
886     NO_UNLOAD_STUB(c2audit,audit_savepath,    nomod_zero);
887     NO_UNLOAD_STUB(c2audit,audit_anchorpath,  nomod_zero);
888     NO_UNLOAD_STUB(c2audit,audit_exit,        nomod_zero);
889     NO_UNLOAD_STUB(c2audit,audit_exec,        nomod_zero);
890     NO_UNLOAD_STUB(c2audit,audit_symlink,     nomod_zero);
891     NO_UNLOAD_STUB(c2audit,audit_symlink_create, nomod_zero);
892     NO_UNLOAD_STUB(c2audit,audit_vncreate_start, nomod_zero);
893     NO_UNLOAD_STUB(c2audit,audit_vncreate_finish, nomod_zero);
894     NO_UNLOAD_STUB(c2audit,audit_enterprom,  nomod_zero);
895     NO_UNLOAD_STUB(c2audit,audit_exitprom,    nomod_zero);
896     NO_UNLOAD_STUB(c2audit,audit_chdirec,     nomod_zero);
897     NO_UNLOAD_STUB(c2audit,audit_setf,        nomod_zero);
898     NO_UNLOAD_STUB(c2audit,audit_sock,        nomod_zero);
899     NO_UNLOAD_STUB(c2audit,audit_strgetmsg,    nomod_zero);
900     NO_UNLOAD_STUB(c2audit,audit_ipc,         nomod_zero);
901     NO_UNLOAD_STUB(c2audit,audit_ipcget,      nomod_zero);
902     NO_UNLOAD_STUB(c2audit,audit_fdsend,     nomod_zero);
903     NO_UNLOAD_STUB(c2audit,audit_fdrecv,     nomod_zero);
904     NO_UNLOAD_STUB(c2audit,audit_priv,        nomod_zero);
905     NO_UNLOAD_STUB(c2audit,audit_setppriv,    nomod_zero);
906     NO_UNLOAD_STUB(c2audit,audit_psecflags,    nomod_zero);
907 #endif /* ! codereview */
908     NO_UNLOAD_STUB(c2audit,audit_devpolicy,   nomod_zero);
909     NO_UNLOAD_STUB(c2audit,audit_setsfat_path, nomod_zero);
910     NO_UNLOAD_STUB(c2audit,audit_cryptoadm,   nomod_zero);
911     NO_UNLOAD_STUB(c2audit,audit_kssl,        nomod_zero);
912     NO_UNLOAD_STUB(c2audit,audit_pf_policy,   nomod_zero);
913     NO_UNLOAD_STUB(c2audit,au_doomsg,        nomod_zero);
914     NO_UNLOAD_STUB(c2audit,au_uwrite,        nomod_zero);
915     NO_UNLOAD_STUB(c2audit,au_to_arg32,      nomod_zero);
916     NO_UNLOAD_STUB(c2audit,au_free_rec,      nomod_zero);
917     END_MODULE(c2audit);
918 #endif

920 /*
921 * Stubs for kernel rpc security service module
922 */
923 #ifndef RPCSEC_MODULE

```

```

924     MODULE(rpcsec,misc);
925     NO_UNLOAD_STUB(rpcsec, sec_clnt_revoke,      nomod_zero);
926     NO_UNLOAD_STUB(rpcsec, authkern_create,     nomod_zero);
927     NO_UNLOAD_STUB(rpcsec, sec_svc_msg,        nomod_zero);
928     NO_UNLOAD_STUB(rpcsec, sec_svc_control,    nomod_zero);
929     END_MODULE(rpcsec);
930 #endif

932 /*
933  * Stubs for rpc RPCSEC_GSS security service module
934  */
935 #ifndef RPCSEC_GSS_MODULE
936     MODULE(rpcsec_gss,misc);
937     NO_UNLOAD_STUB(rpcsec_gss, __svcrpcsec_gss,  nomod_zero);
938     NO_UNLOAD_STUB(rpcsec_gss, rpc_gss_getcred,  nomod_zero);
939     NO_UNLOAD_STUB(rpcsec_gss, rpc_gss_set_callback, nomod_zero);
940     NO_UNLOAD_STUB(rpcsec_gss, rpc_gss_secget,  nomod_zero);
941     NO_UNLOAD_STUB(rpcsec_gss, rpc_gss_secfree, nomod_zero);
942     NO_UNLOAD_STUB(rpcsec_gss, rpc_gss_seccreate, nomod_zero);
943     NO_UNLOAD_STUB(rpcsec_gss, rpc_gss_set_defaults, nomod_zero);
944     NO_UNLOAD_STUB(rpcsec_gss, rpc_gss_revauth, nomod_zero);
945     NO_UNLOAD_STUB(rpcsec_gss, rpc_gss_secpurge, nomod_zero);
946     NO_UNLOAD_STUB(rpcsec_gss, rpc_gss_cleanup, nomod_zero);
947     NO_UNLOAD_STUB(rpcsec_gss, rpc_gss_get_versions, nomod_zero);
948     NO_UNLOAD_STUB(rpcsec_gss, rpc_gss_max_data_length, nomod_zero);
949     NO_UNLOAD_STUB(rpcsec_gss, rpc_gss_svc_max_data_length, nomod_zero);
950     NO_UNLOAD_STUB(rpcsec_gss, rpc_gss_get_service_type, nomod_zero);
951     END_MODULE(rpcsec_gss);
952 #endif

954 #ifndef IWSCN_MODULE
955     MODULE(iwscn,drv);
956     STUB(iwscn, srpop, 0);
957     END_MODULE(iwscn);
958 #endif

960 /*
961  * Stubs for checkpoint-resume module
962  */
963 #ifndef CPR_MODULE
964     MODULE(cpr,misc);
965     STUB(cpr, cpr, 0);
966     END_MODULE(cpr);
967 #endif

969 /*
970  * Stubs for VIS module
971  */
972 #ifndef VIS_MODULE
973     MODULE(vis,misc);
974     STUB(vis, vis_fpu_simulator, 0);
975     STUB(vis, vis fldst, 0);
976     STUB(vis, vis_rdgxr, 0);
977     STUB(vis, vis_wrgsr, 0);
978     END_MODULE(vis);
979 #endif

981 /*
982  * Stubs for kernel probes (tnf module). Not unloadable.
983  */
984 #ifndef TNF_MODULE
985     MODULE(tnf,drv);
986     NO_UNLOAD_STUB(tnf, tnf_ref32_1,      nomod_zero);
987     NO_UNLOAD_STUB(tnf, tnf_string_1,     nomod_zero);
988     NO_UNLOAD_STUB(tnf, tnf_opaque_array_1, nomod_zero);
989     NO_UNLOAD_STUB(tnf, tnf_opaque32_array_1, nomod_zero);

```

```

990     NO_UNLOAD_STUB(tnf, tnf_struct_tag_1,  nomod_zero);
991     NO_UNLOAD_STUB(tnf, tnf_allocate,      nomod_zero);
992     END_MODULE(tnf);
993 #endif

995 /*
996  * Clustering: stubs for bootstrapping.
997  */
998 #ifndef CL_BOOTSTRAP
999     MODULE(cl_bootstrap,misc);
1000     NO_UNLOAD_WSTUB(cl_bootstrap, clboot_modload, nomod_minus_one);
1001     NO_UNLOAD_WSTUB(cl_bootstrap, clboot_loadrootmodules, nomod_zero);
1002     NO_UNLOAD_WSTUB(cl_bootstrap, clboot_rootconf, nomod_zero);
1003     NO_UNLOAD_WSTUB(cl_bootstrap, clboot_mountroot, nomod_zero);
1004     NO_UNLOAD_WSTUB(cl_bootstrap, clconf_init, nomod_zero);
1005     NO_UNLOAD_WSTUB(cl_bootstrap, clconf_get_nodeid, nomod_zero);
1006     NO_UNLOAD_WSTUB(cl_bootstrap, clconf_maximum_nodeid, nomod_zero);
1007     NO_UNLOAD_WSTUB(cl_bootstrap, cluster, nomod_zero);
1008     END_MODULE(cl_bootstrap);
1009 #endif

1011 /*
1012  * Clustering: stubs for cluster infrastructure.
1013  */
1014 #ifndef CL_COMM_MODULE
1015     MODULE(cl_comm,misc);
1016     NO_UNLOAD_STUB(cl_comm, cladmin, nomod_minus_one);
1017     END_MODULE(cl_comm);
1018 #endif

1020 /*
1021  * Clustering: stubs for global file system operations.
1022  */
1023 #ifndef PXFS_MODULE
1024     MODULE(pxfs,fs);
1025     NO_UNLOAD_WSTUB(pxfs, clpxfs_aio_read, nomod_zero);
1026     NO_UNLOAD_WSTUB(pxfs, clpxfs_aio_write, nomod_zero);
1027     NO_UNLOAD_WSTUB(pxfs, cl_flk_state_transition_notify, nomod_zero);
1028     END_MODULE(pxfs);
1029 #endif

1031 /*
1032  * Stubs for PCI configurator module (misc/pcicfg).
1033  */
1034 #ifndef PCICFG_MODULE
1035     MODULE(pcicfg,misc);
1036     STUB(pcicfg, pcicfg_configure, 0);
1037     STUB(pcicfg, pcicfg_unconfigure, 0);
1038     END_MODULE(pcicfg);
1039 #endif

1041 #ifndef PCIHP_MODULE
1042     MODULE(pcihp,misc);
1043     WSTUB(pcihp, pcihp_init, nomod_minus_one);
1044     WSTUB(pcihp, pcihp_uninit, nomod_minus_one);
1045     WSTUB(pcihp, pcihp_info, nomod_minus_one);
1046     WSTUB(pcihp, pcihp_get_cb_ops, nomod_zero);
1047     END_MODULE(pcihp);
1048 #endif

1050 /*
1051  * Stubs for kernel cryptographic framework module (misc/kcf).
1052  */
1053 #ifndef KCF_MODULE
1054     MODULE(kcf,misc);
1055     NO_UNLOAD_STUB(kcf, crypto_mech2id, nomod_minus_one);

```



```

1056 NO_UNLOAD_STUB(kcf, crypto_register_provider, nomod_minus_one);
1057 NO_UNLOAD_STUB(kcf, crypto_unregister_provider, nomod_minus_one);
1058 NO_UNLOAD_STUB(kcf, crypto_provider_notification, nomod_minus_one);
1059 NO_UNLOAD_STUB(kcf, crypto_op_notification, nomod_minus_one);
1060 NO_UNLOAD_STUB(kcf, crypto_kmflag, nomod_minus_one);
1061 NO_UNLOAD_STUB(kcf, crypto_digest, nomod_minus_one);
1062 NO_UNLOAD_STUB(kcf, crypto_digest_prov, nomod_minus_one);
1063 NO_UNLOAD_STUB(kcf, crypto_digest_init, nomod_minus_one);
1064 NO_UNLOAD_STUB(kcf, crypto_digest_init_prov, nomod_minus_one);
1065 NO_UNLOAD_STUB(kcf, crypto_digest_update, nomod_minus_one);
1066 NO_UNLOAD_STUB(kcf, crypto_digest_final, nomod_minus_one);
1067 NO_UNLOAD_STUB(kcf, crypto_digest_key_prov, nomod_minus_one);
1068 NO_UNLOAD_STUB(kcf, crypto_encrypt, nomod_minus_one);
1069 NO_UNLOAD_STUB(kcf, crypto_encrypt_prov, nomod_minus_one);
1070 NO_UNLOAD_STUB(kcf, crypto_encrypt_init, nomod_minus_one);
1071 NO_UNLOAD_STUB(kcf, crypto_encrypt_init_prov, nomod_minus_one);
1072 NO_UNLOAD_STUB(kcf, crypto_encrypt_update, nomod_minus_one);
1073 NO_UNLOAD_STUB(kcf, crypto_encrypt_final, nomod_minus_one);
1074 NO_UNLOAD_STUB(kcf, crypto_decrypt, nomod_minus_one);
1075 NO_UNLOAD_STUB(kcf, crypto_decrypt_prov, nomod_minus_one);
1076 NO_UNLOAD_STUB(kcf, crypto_decrypt_init, nomod_minus_one);
1077 NO_UNLOAD_STUB(kcf, crypto_decrypt_init_prov, nomod_minus_one);
1078 NO_UNLOAD_STUB(kcf, crypto_decrypt_update, nomod_minus_one);
1079 NO_UNLOAD_STUB(kcf, crypto_decrypt_final, nomod_minus_one);
1080 NO_UNLOAD_STUB(kcf, crypto_get_all_mech_info, nomod_minus_one);
1081 NO_UNLOAD_STUB(kcf, crypto_key_check, nomod_minus_one);
1082 NO_UNLOAD_STUB(kcf, crypto_key_check_prov, nomod_minus_one);
1083 NO_UNLOAD_STUB(kcf, crypto_key_derive, nomod_minus_one);
1084 NO_UNLOAD_STUB(kcf, crypto_key_generate, nomod_minus_one);
1085 NO_UNLOAD_STUB(kcf, crypto_key_generate_pair, nomod_minus_one);
1086 NO_UNLOAD_STUB(kcf, crypto_key_unwrap, nomod_minus_one);
1087 NO_UNLOAD_STUB(kcf, crypto_key_wrap, nomod_minus_one);
1088 NO_UNLOAD_STUB(kcf, crypto_mac, nomod_minus_one);
1089 NO_UNLOAD_STUB(kcf, crypto_mac_prov, nomod_minus_one);
1090 NO_UNLOAD_STUB(kcf, crypto_mac_verify, nomod_minus_one);
1091 NO_UNLOAD_STUB(kcf, crypto_mac_verify_prov, nomod_minus_one);
1092 NO_UNLOAD_STUB(kcf, crypto_mac_init, nomod_minus_one);
1093 NO_UNLOAD_STUB(kcf, crypto_mac_init_prov, nomod_minus_one);
1094 NO_UNLOAD_STUB(kcf, crypto_mac_update, nomod_minus_one);
1095 NO_UNLOAD_STUB(kcf, crypto_mac_final, nomod_minus_one);
1096 NO_UNLOAD_STUB(kcf, crypto_mac_decrypt, nomod_minus_one);
1097 NO_UNLOAD_STUB(kcf, crypto_mac_decrypt_prov, nomod_minus_one);
1098 NO_UNLOAD_STUB(kcf, crypto_mac_verify_decrypt, nomod_minus_one);
1099 NO_UNLOAD_STUB(kcf, crypto_mac_verify_decrypt_prov, nomod_minus_one);
1100 NO_UNLOAD_STUB(kcf, crypto_mac_decrypt_init, nomod_minus_one);
1101 NO_UNLOAD_STUB(kcf, crypto_mac_decrypt_init_prov, nomod_minus_one);
1102 NO_UNLOAD_STUB(kcf, crypto_mac_decrypt_update, nomod_minus_one);
1103 NO_UNLOAD_STUB(kcf, crypto_mac_decrypt_final, nomod_minus_one);
1104 NO_UNLOAD_STUB(kcf, crypto_object_copy, nomod_minus_one);
1105 NO_UNLOAD_STUB(kcf, crypto_object_create, nomod_minus_one);
1106 NO_UNLOAD_STUB(kcf, crypto_object_destroy, nomod_minus_one);
1107 NO_UNLOAD_STUB(kcf, crypto_object_find_final, nomod_minus_one);
1108 NO_UNLOAD_STUB(kcf, crypto_object_find_init, nomod_minus_one);
1109 NO_UNLOAD_STUB(kcf, crypto_object_find, nomod_minus_one);
1110 NO_UNLOAD_STUB(kcf, crypto_object_get_attribute_value, nomod_minus_one);
1111 NO_UNLOAD_STUB(kcf, crypto_object_get_size, nomod_minus_one);
1112 NO_UNLOAD_STUB(kcf, crypto_object_set_attribute_value, nomod_minus_one);
1113 NO_UNLOAD_STUB(kcf, crypto_session_close, nomod_minus_one);
1114 NO_UNLOAD_STUB(kcf, crypto_session_login, nomod_minus_one);
1115 NO_UNLOAD_STUB(kcf, crypto_session_logout, nomod_minus_one);
1116 NO_UNLOAD_STUB(kcf, crypto_session_open, nomod_minus_one);
1117 NO_UNLOAD_STUB(kcf, crypto_encrypt_mac, nomod_minus_one);
1118 NO_UNLOAD_STUB(kcf, crypto_encrypt_mac_prov, nomod_minus_one);
1119 NO_UNLOAD_STUB(kcf, crypto_encrypt_mac_init, nomod_minus_one);
1120 NO_UNLOAD_STUB(kcf, crypto_encrypt_mac_init_prov, nomod_minus_one);
1121 NO_UNLOAD_STUB(kcf, crypto_encrypt_mac_update, nomod_minus_one);

```

```

1122 NO_UNLOAD_STUB(kcf, crypto_encrypt_mac_final, nomod_minus_one);
1123 NO_UNLOAD_STUB(kcf, crypto_create_ctx_template, nomod_minus_one);
1124 NO_UNLOAD_STUB(kcf, crypto_destroy_ctx_template, nomod_minus_one);
1125 NO_UNLOAD_STUB(kcf, crypto_get_mech_list, nomod_minus_one);
1126 NO_UNLOAD_STUB(kcf, crypto_free_mech_list, nomod_minus_one);
1127 NO_UNLOAD_STUB(kcf, crypto_cancel_req, nomod_minus_one);
1128 NO_UNLOAD_STUB(kcf, crypto_cancel_ctx, nomod_minus_one);
1129 NO_UNLOAD_STUB(kcf, crypto_bufcall_alloc, nomod_minus_one);
1130 NO_UNLOAD_STUB(kcf, crypto_bufcall_free, nomod_minus_one);
1131 NO_UNLOAD_STUB(kcf, crypto_bufcall, nomod_minus_one);
1132 NO_UNLOAD_STUB(kcf, crypto_unbufcall, nomod_minus_one);
1133 NO_UNLOAD_STUB(kcf, crypto_notify_events, nomod_minus_one);
1134 NO_UNLOAD_STUB(kcf, crypto_unnotify_events, nomod_minus_one);
1135 NO_UNLOAD_STUB(kcf, crypto_get_provider, nomod_minus_one);
1136 NO_UNLOAD_STUB(kcf, crypto_get_provinfo, nomod_minus_one);
1137 NO_UNLOAD_STUB(kcf, crypto_release_provider, nomod_minus_one);
1138 NO_UNLOAD_STUB(kcf, crypto_sign, nomod_minus_one);
1139 NO_UNLOAD_STUB(kcf, crypto_sign_prov, nomod_minus_one);
1140 NO_UNLOAD_STUB(kcf, crypto_sign_init, nomod_minus_one);
1141 NO_UNLOAD_STUB(kcf, crypto_sign_init_prov, nomod_minus_one);
1142 NO_UNLOAD_STUB(kcf, crypto_sign_update, nomod_minus_one);
1143 NO_UNLOAD_STUB(kcf, crypto_sign_final, nomod_minus_one);
1144 NO_UNLOAD_STUB(kcf, crypto_sign_recover, nomod_minus_one);
1145 NO_UNLOAD_STUB(kcf, crypto_sign_recover_prov, nomod_minus_one);
1146 NO_UNLOAD_STUB(kcf, crypto_sign_recover_init_prov, nomod_minus_one);
1147 NO_UNLOAD_STUB(kcf, crypto_verify, nomod_minus_one);
1148 NO_UNLOAD_STUB(kcf, crypto_verify_prov, nomod_minus_one);
1149 NO_UNLOAD_STUB(kcf, crypto_verify_init, nomod_minus_one);
1150 NO_UNLOAD_STUB(kcf, crypto_verify_init_prov, nomod_minus_one);
1151 NO_UNLOAD_STUB(kcf, crypto_verify_update, nomod_minus_one);
1152 NO_UNLOAD_STUB(kcf, crypto_verify_final, nomod_minus_one);
1153 NO_UNLOAD_STUB(kcf, crypto_verify_recover, nomod_minus_one);
1154 NO_UNLOAD_STUB(kcf, crypto_verify_recover_prov, nomod_minus_one);
1155 NO_UNLOAD_STUB(kcf, crypto_verify_recover_init_prov, nomod_minus_one);
1156 NO_UNLOAD_STUB(kcf, random_add_entropy, nomod_minus_one);
1157 NO_UNLOAD_STUB(kcf, random_add_pseudo_entropy, nomod_minus_one);
1158 NO_UNLOAD_STUB(kcf, random_get_blocking_bytes, nomod_minus_one);
1159 NO_UNLOAD_STUB(kcf, random_get_bytes, nomod_minus_one);
1160 NO_UNLOAD_STUB(kcf, random_get_pseudo_bytes, nomod_minus_one);
1161 END_MODULE(kcf);
1162 #endif

1164 /*
1165  * Stubs for sha1. A non-unloadable module.
1166  */
1167 #ifndef SHA1_MODULE
1168 MODULE(sha1,crypto);
1169 NO_UNLOAD_STUB(sha1, SHA1Init, nomod_void);
1170 NO_UNLOAD_STUB(sha1, SHA1Update, nomod_void);
1171 NO_UNLOAD_STUB(sha1, SHA1Final, nomod_void);
1172 END_MODULE(sha1);
1173 #endif

1175 /*
1176  * The following stubs are used by the mac module.
1177  * Since dld already depends on mac, these
1178  * stubs are needed to avoid circular dependencies.
1179  */
1180 #ifndef DLD_MODULE
1181 MODULE(dld,drv);
1182 STUB(dld, dld_init_ops, nomod_void);
1183 STUB(dld, dld_fini_ops, nomod_void);
1184 STUB(dld, dld_autopush, nomod_minus_one);
1185 STUB(dld, dld_devt_to_instance, nomod_minus_one);
1186 STUB(dld, dld_ioc_register, nomod_einval);
1187 STUB(dld, dld_ioc_unregister, nomod_void);

```

```

1188     END_MODULE(dld);
1189 #endif

1191 /*
1192 * The following stubs are used by the mac module.
1193 * Since dls already depends on mac, these
1194 * stubs are needed to avoid circular dependencies.
1195 */
1196 #ifndef DLS_MODULE
1197     MODULE(dls,misc);
1198     STUB(dls, dls_devnet_mac, nomod_zero);
1199     STUB(dls, dls_devnet_hold_tmp, nomod_einval);
1200     STUB(dls, dls_devnet_rele_tmp, nomod_void);
1201     STUB(dls, dls_devnet_hold_link, nomod_einval);
1202     STUB(dls, dls_devnet_rele_link, nomod_void);
1203     STUB(dls, dls_devnet_prop_task_wait, nomod_void);
1204     STUB(dls, dls_mgmt_get_linkid, nomod_einval);
1205     STUB(dls, dls_devnet_macname2linkid, nomod_einval);
1206     STUB(dls, dls_mgmt_get_linkinfo, nomod_einval);
1207     END_MODULE(dls);
1208 #endif

1210 #ifndef SOFTMAC_MODULE
1211     MODULE(softmac,drv);
1212     STUB(softmac, softmac_hold_device, nomod_einval);
1213     STUB(softmac, softmac_rele_device, nomod_void);
1214     STUB(softmac, softmac_recreate, nomod_void);
1215     END_MODULE(softmac);
1216 #endif

1218 #ifndef IPTUN_MODULE
1219     MODULE(iptun,drv);
1220     STUB(iptun, iptun_create, nomod_einval);
1221     STUB(iptun, iptun_delete, nomod_einval);
1222     STUB(iptun, iptun_set_policy, nomod_einval);
1223     END_MODULE(iptun);
1224 #endif

1226 /*
1227 * Stubs for dcopy, for Intel IOAT KAPIS
1228 */
1229 #ifndef DCOPY_MODULE
1230     MODULE(dcopy,misc);
1231     NO_UNLOAD_STUB(dcopy, dcopy_query, nomod_minus_one);
1232     NO_UNLOAD_STUB(dcopy, dcopy_query_channel, nomod_minus_one);
1233     NO_UNLOAD_STUB(dcopy, dcopy_alloc, nomod_minus_one);
1234     NO_UNLOAD_STUB(dcopy, dcopy_free, nomod_minus_one);
1235     NO_UNLOAD_STUB(dcopy, dcopy_cmd_alloc, nomod_minus_one);
1236     NO_UNLOAD_STUB(dcopy, dcopy_cmd_free, nomod_void);
1237     NO_UNLOAD_STUB(dcopy, dcopy_cmd_post, nomod_minus_one);
1238     NO_UNLOAD_STUB(dcopy, dcopy_cmd_poll, nomod_minus_one);
1239     END_MODULE(dcopy);
1240 #endif

1242 #ifndef IPNET_MODULE
1243     MODULE(ipnet,drv);
1244     STUB(ipnet, ipnet_if_getdev, nomod_zero);
1245     STUB(ipnet, ipnet_walk_if, nomod_zero);
1246     END_MODULE(ipnet);
1247 #endif

1249 /*
1250 * Stubs for kernel socket, for iscsi
1251 */
1252 #ifndef KSOCKET_MODULE
1253     MODULE(ksocket, misc);

```

```

1254     NO_UNLOAD_STUB(ksocket, ksocket_getsockopt, nomod_minus_one);
1255     NO_UNLOAD_STUB(ksocket, ksocket_getsockopt, nomod_minus_one);
1256     NO_UNLOAD_STUB(ksocket, ksocket_getpeername, nomod_minus_one);
1257     NO_UNLOAD_STUB(ksocket, ksocket_getsockname, nomod_minus_one);
1258     NO_UNLOAD_STUB(ksocket, ksocket_socket, nomod_minus_one);
1259     NO_UNLOAD_STUB(ksocket, ksocket_bind, nomod_minus_one);
1260     NO_UNLOAD_STUB(ksocket, ksocket_listen, nomod_minus_one);
1261     NO_UNLOAD_STUB(ksocket, ksocket_accept, nomod_minus_one);
1262     NO_UNLOAD_STUB(ksocket, ksocket_connect, nomod_minus_one);
1263     NO_UNLOAD_STUB(ksocket, ksocket_recv, nomod_minus_one);
1264     NO_UNLOAD_STUB(ksocket, ksocket_recvfrom, nomod_minus_one);
1265     NO_UNLOAD_STUB(ksocket, ksocket_recvmsg, nomod_minus_one);
1266     NO_UNLOAD_STUB(ksocket, ksocket_send, nomod_minus_one);
1267     NO_UNLOAD_STUB(ksocket, ksocket_sendto, nomod_minus_one);
1268     NO_UNLOAD_STUB(ksocket, ksocket_sendmsg, nomod_minus_one);
1269     NO_UNLOAD_STUB(ksocket, ksocket_ioctl, nomod_minus_one);
1270     NO_UNLOAD_STUB(ksocket, ksocket_setcallbacks, nomod_minus_one);
1271     NO_UNLOAD_STUB(ksocket, ksocket_hold, nomod_minus_one);
1272     NO_UNLOAD_STUB(ksocket, ksocket_rele, nomod_minus_one);
1273     NO_UNLOAD_STUB(ksocket, ksocket_shutdown, nomod_minus_one);
1274     NO_UNLOAD_STUB(ksocket, ksocket_close, nomod_minus_one);
1275     END_MODULE(ksocket);
1276 #endif

1278 /*
1279 * Stubs for elfexec
1280 */
1281 #ifndef ELFEEXEC_MODULE
1282     MODULE(elfexec,exec);
1283     STUB(elfexec, elfexec, nomod_einval);
1284     STUB(elfexec, elf32exec, nomod_einval);
1285     STUB(elfexec, mapexec_brand, nomod_einval);
1286     STUB(elfexec, mapexec32_brand, nomod_einval);
1287     END_MODULE(elfexec);
1288 #endif

1290 ! this is just a marker for the area of text that contains stubs
1291     .seg ".text"
1292     .global stubs_end
1293 stubs_end:
1294     nop

1296 #endif /* lint */

```

```

*****
3375 Thu Jun 30 22:03:56 2016
new/usr/src/uts/sparc/os/name_to_sysnum
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****

```

1	nosys	0
2	rexit	1
3	psecflags	2
4	#endif /* ! codereview */	
5	read	3
6	write	4
7	open	5
8	close	6
9	linkat	7
10	link	9
11	unlink	10
12	symlinkat	11
13	chdir	12
14	gtime	13
15	mknod	14
16	chmod	15
17	chown	16
18	brk	17
19	stat	18
20	lseek	19
21	getpid	20
22	mount	21
23	readlinkat	22
24	setuid	23
25	getuid	24
26	stime	25
27	pcsample	26
28	alarm	27
29	fstat	28
30	pause	29
31	stty	31
32	gtty	32
33	access	33
34	nice	34
35	statfs	35
36	sysync	36
37	kill	37
38	fstatfs	38
39	setpgrp	39
40	uucopystr	40
41	pipe	42
42	times	43
43	profil	44
44	faccessat	45
45	setgid	46
46	getgid	47
47	mkmodat	48
48	msgsys	49
49	sysacct	51
50	shmsys	52
51	semsys	53
52	ioctl	54
53	uadmin	55
54	fchowmat	56
55	utssys	57
56	fdsync	58
57	exece	59
58	umask	60

59	chroot	61
60	fcntl	62
61	ulimit	63
62	renameat	64
63	unlinkat	65
64	fstatat	66
65	fstatat64	67
66	openat	68
67	openat64	69
68	tasksys	70
69	acctctl	71
70	exacctsys	72
71	getpagesizes	73
72	rctlsys	74
73	sidsys	75
74	lwp_park	77
75	sendfilev	78
76	rmdir	79
77	mkdir	80
78	getdents	81
79	privsys	82
80	ucredsys	83
81	sysfs	84
82	getmsg	85
83	putmsg	86
84	lstat	88
85	symlink	89
86	readlink	90
87	setgroups	91
88	getgroups	92
89	fchmod	93
90	fchown	94
91	sigprocmask	95
92	sigsuspend	96
93	sigaltstack	97
94	sigaction	98
95	sigpending	99
96	setcontext	100
97	fchmodat	101
98	mkdirat	102
99	statvfs	103
100	fstatvfs	104
101	getloadavg	105
102	nfs	106
103	waitsys	107
104	sigsendsys	108
105	utimesys	110
106	sigresend	111
107	prctlsys	112
108	pathconf	113
109	mincore	114
110	mmap	115
111	mprotect	116
112	munmap	117
113	fpathconf	118
114	vfork	119
115	fchdir	120
116	readv	121
117	writev	122
118	preadv	123
119	pwritev	124
120	getrandom	126
121	mmapobj	127
122	setrlimit	128
123	getrlimit	129
124	lchown	130

125	memcntl	131
126	getpmsg	132
127	putpmsg	133
128	rename	134
129	uname	135
130	setegid	136
131	sysconfig	137
132	adjtime	138
133	systeminfo	139
134	sharefs	140
135	seteuid	141
136	forksys	142
137	sigwait	144
138	lwp_info	145
139	yield	146
140	lwp_sema_post	148
141	lwp_sema_trywait	149
142	lwp_detach	150
143	corectl	151
144	modctl	152
145	fchroot	153
146	vhangup	155
147	gettimeofday	156
148	getitimer	157
149	setitimer	158
150	lwp_create	159
151	lwp_exit	160
152	lwp_suspend	161
153	lwp_continue	162
154	lwp_kill	163
155	lwp_self	164
156	lwp_sigmask	165
157	lwp_wait	167
158	lwp_mutex_wakeup	168
159	lwp_cond_wait	170
160	lwp_cond_signal	171
161	lwp_cond_broadcast	172
162	pread	173
163	pwrite	174
164	llseek	175
165	inst_sync	176
166	brandsys	177
167	kaio	178
168	cpc	179
169	meminfosys	180
170	rusagesys	181
171	portfs	182
172	pollsys	183
173	labelsys	184
174	acl	185
175	c2audit	186
176	processor_bind	187
177	processor_info	188
178	p_online	189
179	sigqueue	190
180	clock_gettime	191
181	clock_settime	192
182	clock_getres	193
183	timer_create	194
184	timer_delete	195
185	timer_settime	196
186	timer_gettime	197
187	timer_getoverrun	198
188	nanosleep	199
189	facl	200
190	doorfs	201

191	setreuid	202
192	setregid	203
193	install_utrap	204
194	signotify	205
195	schedctl	206
196	pset	207
197	sparc_utrap_install	208
198	resolvepath	209
199	lwp_mutex_timedlock	210
200	lwp_sema_timedwait	211
201	lwp_rwlock_sys	212
202	getdents64	213
203	mmap64	214
204	stat64	215
205	lstat64	216
206	fstat64	217
207	statvfs64	218
208	fstatvfs64	219
209	setrlimit64	220
210	getrlimit64	221
211	pread64	222
212	pwrite64	223
213	open64	225
214	rpcmod	226
215	zone	227
216	autofs	228
217	getcwd	229
218	so_socket	230
219	so_socketpair	231
220	bind	232
221	listen	233
222	accept	234
223	connect	235
224	shutdown	236
225	recv	237
226	recvfrom	238
227	recvmsg	239
228	send	240
229	sendmsg	241
230	sendto	242
231	getpeername	243
232	getsockname	244
233	getsockopt	245
234	setsockopt	246
235	sockconfig	247
236	ntp_gettime	248
237	ntp_adjtime	249
238	lwp_mutex_unlock	250
239	lwp_mutex_trylock	251
240	lwp_mutex_register	252
241	cladm	253
242	uucopy	254
243	umount2	255

```

*****
14174 Thu Jun 30 22:03:57 2016
new/usr/src/uts/sun4/os/mlsetup.c
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
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 */

26 #include <sys/types.h>
27 #include <sys/system.h>
28 #include <sys/archsystem.h>
29 #include <sys/machsystem.h>
30 #include <sys/disp.h>
31 #include <sys/autoconf.h>
32 #include <sys/promif.h>
33 #include <sys/prom_plat.h>
34 #include <sys/promimpl.h>
35 #include <sys/platform_module.h>
36 #include <sys/clock.h>
37 #include <sys/pte.h>
38 #include <sys/sch.h>
39 #include <sys/cpu.h>
40 #include <sys/stack.h>
41 #include <sys/intreg.h>
42 #include <sys/ivintr.h>
43 #include <vm/as.h>
44 #include <vm/hat_sfmmu.h>
45 #include <sys/reboot.h>
46 #include <sys/sysmacros.h>
47 #include <sys/vtrace.h>
48 #include <sys/trap.h>
49 #include <sys/machtrap.h>
50 #include <sys/privregs.h>
51 #include <sys/machpcb.h>
52 #include <sys/proc.h>
53 #include <sys/cpupart.h>
54 #include <sys/pset.h>
55 #include <sys/cpu_module.h>
56 #include <sys/copyops.h>
57 #include <sys/panic.h>
58 #include <sys/bootconf.h>      /* for bootops */

```

```

59 #include <sys/pg.h>
60 #include <sys/kdi.h>
61 #include <sys/fpras.h>

63 #include <sys/prom_debug.h>
64 #include <sys/debug.h>

66 #include <sys/sunddi.h>
67 #include <sys/lgrp.h>
68 #include <sys/traptrace.h>

70 #include <sys/kobj_impl.h>
71 #include <sys/kdi_machimpl.h>

73 /*
74  * External Routines:
75  */
76 extern void map_wellknown_devices(void);
77 extern void hsvc_setup(void);
78 extern void mach_descrip_startup_init(void);
79 extern void mach_soft_state_init(void);

81 int      dcache_size;
82 int      dcache_linesize;
83 int      icache_size;
84 int      icache_linesize;
85 int      ecache_size;
86 int      ecache_alignsize;
87 int      ecache_associativity;
88 int      ecache_setsize;      /* max possible e$ setsize */
89 int      cpu_setsize;        /* max e$ setsize of configured cpus */
90 int      dcache_line_mask;  /* spifire only */
91 int      vac_size;          /* cache size in bytes */
92 uint_t   vac_mask;          /* VAC alignment consistency mask */
93 int      vac_shift;         /* log2(vac_size) for pmapout() */
94 int      vac = 0;           /* virtual address cache type (none == 0) */

96 /*
97  * fpRAS. An individual sun4* machine class (or perhaps subclass,
98  * eg sun4u/cheetah) must set fpras_implemented to indicate that it implements
99  * the fpRAS feature. The feature can be suppressed by setting fpras_disable
100 * or the mechanism can be disabled for individual copy operations with
101 * fpras_disableids. All these are checked in post_startup() code so
102 * fpras_disable and fpras_disableids can be set in /etc/system.
103 * If/when fpRAS is implemented on non-sun4 architectures these
104 * definitions will need to move up to the common level.
105 */
106 int      fpras_implemented;
107 int      fpras_disable;
108 int      fpras_disableids;

110 /*
111  * Static Routines:
112  */
113 static void kern_splr_preprom(void);
114 static void kern_splx_postprom(void);

116 /*
117  * Setup routine called right before main(). Interposing this function
118  * before main() allows us to call it in a machine-independent fashion.
119  */

121 void
122 mlsetup(struct regs *rp, kfpu_t *fp)
123 {
124     struct machpcb *mpcb;

```

```

126     extern char t0stack[];
127     extern struct classfuncs sys_classfuncs;
128     extern disp_t cpu0_disp;
129     unsigned long long pa;

131 #ifdef TRAPTRACE
132     TRAP_TRACE_CTL *ctlp;
133 #endif /* TRAPTRACE */

135     /* drop into kmdb on boot -d */
136     if (boothowto & RB_DEBUGENTER)
137         kmdb_enter();

139     /*
140      * initialize cpu_self
141      */
142     cpu0.cpu_self = &cpu0;

144     /*
145      * initialize t0
146      */
147     t0.t_stk = (caddr_t)rp - REGOFF;
148     /* Can't use va_to_pa here - wait until prom_ initialized */
149     t0.t_stkbase = t0stack;
150     t0.t_pri = maxclsypri - 3;
151     t0.t_schedflag = TS_LOAD | TS_DONT_SWAP;
152     t0.t_procp = &p0;
153     t0.t_plockp = &p0lock.pl_lock;
154     t0.t_lwp = &lwp0;
155     t0.t_forw = &t0;
156     t0.t_back = &t0;
157     t0.t_next = &t0;
158     t0.t_prev = &t0;
159     t0.t_cpu = &cpu0;          /* loaded by _start */
160     t0.t_disp_queue = &cpu0_disp;
161     t0.t_bind_cpu = PBIND_NONE;
162     t0.t_bind_pset = PS_NONE;
163     t0.t_bindflag = (uchar_t)default_binding_mode;
164     t0.t_cpupart = &cp_default;
165     t0.t_clfuncs = &sys_classfuncs.thread;
166     t0.t_copyops = NULL;
167     THREAD_ONPROC(&t0, CPU);

169     lwp0.lwp_thread = &t0;
170     lwp0.lwp_procp = &p0;
171     lwp0.lwp_regs = (void *)rp;
172     t0.t_tid = p0.p_lwpcnt = p0.p_lwprcnt = p0.p_lwpid = 1;

174     mpcb = lwptompcb(&lwp0);
175     mpcb->mpcb_fpu = fp;
176     mpcb->mpcb_fpu->fpu_q = mpcb->mpcb_fpu_q;
177     mpcb->mpcb_thread = &t0;
178     lwp0.lwp_fpu = (void *)mpcb->mpcb_fpu;

180     p0.p_exec = NULL;
181     p0.p_stat = SRUN;
182     p0.p_flag = SSYS;
183     p0.p_tlist = &t0;
184     p0.p_stksize = 2*PAGESIZE;
185     p0.p_stkpageszc = 0;
186     p0.p_as = &kas;
187     p0.p_lockp = &p0lock;
188     p0.p_utraps = NULL;
189     p0.p_brkpageszc = 0;
190     p0.p_tl_lgrp_id = LGRP_NONE;

```

```

191     p0.p_tr_lgrp_id = LGRP_NONE;
192     psecflags_default(&p0.p_secflags);
193 #endif /* ! codereview */
194     sigorset(&p0.p_ignore, &ignoredefault);

197 #endif /* ! codereview */
198     CPU->cpu_thread = &t0;
199     CPU->cpu_dispthread = &t0;
200     bzero(&cpu0_disp, sizeof (disp_t));
201     CPU->cpu_disp = &cpu0_disp;
202     CPU->cpu_disp->disp_cpu = CPU;
203     CPU->cpu_idle_thread = &t0;
204     CPU->cpu_flags = CPU_RUNNING;
205     CPU->cpu_id = getprocessorid();
206     CPU->cpu_dispatch_pri = t0.t_pri;

208     /*
209      * Initialize thread/cpu microstate accounting
210      */
211     init_mstate(&t0, LMS_SYSTEM);
212     init_cpu_mstate(CPU, CMS_SYSTEM);

214     /*
215      * Initialize lists of available and active CPUs.
216      */
217     cpu_list_init(CPU);

219     cpu_vm_data_init(CPU);

221     pg_cpu_bootstrap(CPU);

223     (void) prom_set_preprom(kern_splr_preprom);
224     (void) prom_set_postprom(kern_splx_postprom);
225     PRM_INFO("mlsetup: now ok to call prom_printf");

227     mpcb->mpcb_pa = va_to_pa(t0.t_stk);

229     /*
230      * Claim the physical and virtual resources used by panicbuf,
231      * then map panicbuf. This operation removes the phys and
232      * virtual addresses from the free lists.
233      */
234     if (prom_claim_virt(PANICBUFSIZE, panicbuf) != panicbuf)
235         prom_panic("Can't claim panicbuf virtual address");

237     if (prom_retain("panicbuf", PANICBUFSIZE, MMU_PAGESIZE, &pa) != 0)
238         prom_panic("Can't allocate retained panicbuf physical address");

240     if (prom_map_phys(-1, PANICBUFSIZE, panicbuf, pa) != 0)
241         prom_panic("Can't map panicbuf");

243     PRM_DEBUG(panicbuf);
244     PRM_DEBUG(pa);

246     /*
247      * Negotiate hypervisor services, if any
248      */
249     hsvc_setup();
250     mach_soft_state_init();

252 #ifndef TRAPTRACE
253     /*
254      * initialize the trap trace buffer for the boot cpu
255      * XXX todo, dynamically allocate this buffer too
256      */

```

```

257     ctlp = &trap_trace_ctl[CPU->cpu_id];
258     ctlp->d.vaddr_base = trap_tr0;
259     ctlp->d.offset = ctlp->d.last_offset = 0;
260     ctlp->d.limit = TRAP_TSIZE; /* XXX dynamic someday */
261     ctlp->d.paddr_base = va_to_pa(trap_tr0);
262 #endif /* TRAPTRACE */

264     /*
265      * Initialize the Machine Description kernel framework
266      */

268     mach_descrip_startup_init();

270     /*
271      * initialize HV trap trace buffer for the boot cpu
272      */
273     mach_htraptrace_setup(CPU->cpu_id);
274     mach_htraptrace_configure(CPU->cpu_id);

276     /*
277      * lgroup framework initialization. This must be done prior
278      * to devices being mapped.
279      */
280     lgrp_init(LGRP_INIT_STAGE1);

282     cpu_setup();

284     if (boothowto & RB_HALT) {
285         prom_printf("unix: kernel halted by -h flag\n");
286         prom_enter_mon();
287     }

289     setcputype();
290     map_wellknown_devices();
291     setcpudelay();
292 }

294 /*
295 * These routines are called immediately before and
296 * immediately after calling into the firmware. The
297 * firmware is significantly confused by preemption -
298 * particularly on MP machines - but also on UP's too.
299 */

301 static int saved_spl;

303 static void
304 kern_splr_preprom(void)
305 {
306     saved_spl = spl7();
307 }

309 static void
310 kern_splx_postprom(void)
311 {
312     splx(saved_spl);
313 }

316 /*
317 * WARNING
318 * The code fom here to the end of mlsetup.c runs before krtld has
319 * knitted unix and genunix together. It can call routines in unix,
320 * but calls into genunix will fail spectacularly. More specifically,
321 * calls to prom_*, bop_* and str* will work, everything else is
322 * caveat emptor.

```

```

323 *
324 * Also note that while #ifdef sun4u is generally a bad idea, they
325 * exist here to concentrate the dangerous code into a single file.
326 */

328 static char *
329 getcpulist(void)
330 {
331     pnode_t node;
332     /* big enough for OBP_NAME and for a reasonably sized OBP_COMPATIBLE. */
333     static char cpubuf[5 * OBP_MAXDRVNAME];
334     int nlen, clen, i;
335 #ifdef sun4u
336     char dname[OBP_MAXDRVNAME];
337 #endif

339     node = prom_findnode_bydevtype(prom_rootnode(), OBP_CPU);
340     if (node != OBP_NONODE && node != OBP_BADNODE) {
341         if ((nlen = prom_getproplen(node, OBP_NAME)) <= 0 ||
342             nlen > sizeof (cpubuf) ||
343             prom_getprop(node, OBP_NAME, cpubuf) <= 0)
344             prom_panic("no name in cpu node");

346         /* nlen includes the terminating null character */
347 #ifdef sun4v
348         if ((clen = prom_getproplen(node, OBP_COMPATIBLE)) > 0) {
349 #else /* sun4u */
350         /*
351          * For the CMT case, need check the parent "core"
352          * node for the compatible property.
353          */
354         if ((clen = prom_getproplen(node, OBP_COMPATIBLE)) > 0 ||
355             ((node = prom_parentnode(node)) != OBP_NONODE &&
356              node != OBP_BADNODE &&
357              (clen = prom_getproplen(node, OBP_COMPATIBLE)) > 0 &&
358              prom_getprop(node, OBP_DEVICETYPE, dname) > 0 &&
359              strcmp(dname, "core") == 0)) {
360 #endif
361             if ((clen + nlen) > sizeof (cpubuf))
362                 prom_panic("cpu node \"compatible\" too long");
363             /* read in compatible, leaving space for ':' */
364             if (prom_getprop(node, OBP_COMPATIBLE,
365                 &cpubuf[nlen]) != clen)
366                 prom_panic("cpu node \"compatible\" error");
367             clen += nlen; /* total length */
368             /* convert all null characters to ':' */
369             clen--; /* except the final one... */
370             for (i = 0; i < clen; i++)
371                 if (cpubuf[i] == '\0')
372                     cpubuf[i] = ':';
373         }
374 #ifdef sun4u
375         /*
376          * Some PROMS return SUNW,UltraSPARC when they actually have
377          * SUNW,UltraSPARC-II cpus. Since we're now filtering out all
378          * SUNW,UltraSPARC systems during the boot phase, we can safely
379          * point the auxv CPU value at SUNW,UltraSPARC-II.
380          */
381         if (strcmp("SUNW,UltraSPARC", cpubuf) == 0)
382             (void) strcpy(cpubuf, "SUNW,UltraSPARC-II");
383 #endif
384         return (cpubuf);
385     } else
386         return (NULL);
387 }

```

```

389 /*
390  * called immediately from _start to stitch the
391  * primary modules together
392  */
393 void
394 kobj_start(void *cif)
395 {
396     Ehdr *ehdr;
397     Phdr *phdr;
398     uint32_t eadr, padr;
399     val_t bootaux[BA_NUM];
400     int i;

402     prom_init("kernel", cif);
403     bop_init();
404 #ifdef DEBUG
405     if (bop_getproplen("stop-me") != -1)
406         prom_enter_mon();
407 #endif

409     if (bop_getprop("elfheader-address", (caddr_t)&eadr) == -1)
410         prom_panic("no ELF image");
411     ehdr = (Ehdr *) (uintptr_t) eadr;
412     for (i = 0; i < BA_NUM; i++)
413         bootaux[i].ba_val = NULL;
414     bootaux[BA_PHNUM].ba_val = ehdr->e_phnum;
415     bootaux[BA_PHEMT].ba_val = ehdr->e_phentsize;
416     bootaux[BA_LDNAME].ba_ptr = NULL;

418     padr = eadr + ehdr->e_phoff;
419     bootaux[BA_PHDR].ba_ptr = (void *) (uintptr_t) padr;
420     for (i = 0; i < ehdr->e_phnum; i++) {
421         phdr = (Phdr *) ((uintptr_t) padr + i * ehdr->e_phentsize);
422         if (phdr->p_type == PT_DYNAMIC) {
423             bootaux[BA_DYNAMIC].ba_ptr = (void *) phdr->p_vaddr;
424             break;
425         }
426     }

428     bootaux[BA_LPAGESZ].ba_val = MMU_PAGESIZE4M;
429     bootaux[BA_PAGESZ].ba_val = MMU_PAGESIZE;
430     bootaux[BA_IFLUSH].ba_val = 1;
431     bootaux[BA_CPU].ba_ptr = getcpulist();
432     bootaux[BA_MMU].ba_ptr = NULL;

434     kobj_init(cif, NULL, bootops, bootaux);

436     /* kernel stitched together; we can now test #pragma's */
437     if (&plat_setprop_enter != NULL) {
438         prom_setprop_enter = &plat_setprop_enter;
439         prom_setprop_exit = &plat_setprop_exit;
440         ASSERT(prom_setprop_exit != NULL);
441     }

443 }

445 /*
446  * Create modpath from kernel name.
447  * If we booted:
448  * /platform/'uname -i'/kernel/sparcv9/unix
449  * or
450  * /platform/'uname -m'/kernel/sparcv9/unix
451  * then make the modpath:
452  * /platform/'uname -i'/kernel /platform/'uname -m'/kernel
454  */

```

```

455  * otherwise, make the modpath the dir the kernel was
456  * loaded from, minus any sparcv9 extension
457  *
458  * note the sparcv9 dir is optional since a unix -> sparcv9/unix
459  * symlink is available as a shortcut.
460  */
461 void
462 mach_modpath(char *path, const char *fname)
463 {
464     char *p;
465     int len, compat;
466     const char prefix[] = "/platform/";
467     char platname[MAXPATHLEN];
468 #ifdef sun4u
469     char defname[] = "sun4u";
470 #else
471     char defname[] = "sun4v";
472 #endif
473     const char suffix[] = "/kernel";
474     const char isastr[] = "/sparcv9";

476     /*
477      * check for /platform
478      */
479     p = (char *) fname;
480     if (strncmp(p, prefix, sizeof (prefix) - 1) != 0)
481         goto nopath;
482     p += sizeof (prefix) - 1;

484     /*
485      * check for the default name or the platform name.
486      * also see if we used the 'compatible' name
487      * (platname == default)
488      */
489     (void) bop_getprop("impl-arch-name", platname);
490     compat = strcmp(platname, defname) == 0;
491     len = strlen(platname);
492     if (strncmp(p, platname, len) == 0)
493         p += len;
494     else if (strncmp(p, defname, sizeof (defname) - 1) == 0)
495         p += sizeof (defname) - 1;
496     else
497         goto nopath;

499     /*
500      * check for /kernel/sparcv9 or just /kernel
501      */
502     if (strncmp(p, suffix, sizeof (suffix) - 1) != 0)
503         goto nopath;
504     p += sizeof (suffix) - 1;
505     if (strncmp(p, isastr, sizeof (isastr) - 1) == 0)
506         p += sizeof (isastr) - 1;

508     /*
509      * check we're at the last component
510      */
511     if (p != strrchr(fname, '/'))
512         goto nopath;

514     /*
515      * everything is kosher; setup modpath
516      */
517     (void) strcpy(path, "/platform/");
518     (void) strcat(path, platname);
519     (void) strcat(path, "/kernel");
520     if (!compat) {

```



```
521         (void) strcat(path, " /platform/");
522         (void) strcat(path, defname);
523         (void) strcat(path, "/kernel");
524     }
525     return;

527 nopath:
528     /*
529     * Construct the directory path from the filename.
530     */
531     if ((p = strrchr(fname, '/')) == NULL)
532         return;

534     while (p > fname && *(p - 1) == '/')
535         p--; /* remove trailing '/' characters */
536     if (p == fname)
537         p++; /* so "/" -is- the modpath in this case */

539     /*
540     * Remove optional isa-dependent directory name - the module
541     * subsystem will put this back again (!)
542     */
543     len = p - fname;
544     if (len > sizeof (isastr) - 1 &&
545         strncmp(&fname[len - (sizeof (isastr) - 1)], isastr,
546             sizeof (isastr) - 1) == 0)
547         p -= sizeof (isastr) - 1;
548     (void) strncpy(path, fname, p - fname);
549 }
```

```

*****
26858 Thu Jun 30 22:03:58 2016
new/usr/src/uts/sun4/vm/vm_dep.c
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
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 */

26 /*
27 * UNIX machine dependent virtual memory support.
28 */

30 #include <sys/vm.h>
31 #include <sys/exec.h>

33 #include <sys/exechnr.h>
34 #include <vm/seg_kmem.h>
35 #include <sys/atomic.h>
36 #include <sys/archsystem.h>
37 #include <sys/machsystem.h>
38 #include <sys/kdi.h>
39 #include <sys/cpu_module.h>
40 #include <sys/secflags.h>
41 #endif /* ! codereview */

43 #include <vm/hat_sfmmu.h>

45 #include <sys/memnode.h>

47 #include <sys/mem_config.h>
48 #include <sys/mem_cage.h>
49 #include <vm/vm_dep.h>
50 #include <vm/page.h>
51 #include <sys/platform_module.h>

53 /*
54 * These variables are set by module specific config routines.
55 * They are only set by modules which will use physical cache page coloring.
56 */
57 int do_pg_coloring = 0;

```

```

59 /*
60 * These variables can be conveniently patched at kernel load time to
61 * prevent do_pg_coloring from being enabled by
62 * module specific config routines.
63 */

65 int use_page_coloring = 1;

67 /*
68 * initialized by page_coloring_init()
69 */
70 extern uint_t page_colors;
71 extern uint_t page_colors_mask;
72 extern uint_t page_coloring_shift;
73 int cpu_page_colors;
74 uint_t vac_colors = 0;
75 uint_t vac_colors_mask = 0;

77 /* cpu specific coloring initialization */
78 extern void page_coloring_init_cpu();
79 #pragma weak page_coloring_init_cpu

81 /*
82 * get the ecache setsize for the current cpu.
83 */
84 #define CPUSETSIZE() (cpunodes[CPU->cpu_id].ecache_setsize)

86 plcnt_t plcnt; /* page list count */

88 /*
89 * This variable is set by the cpu module to contain the lowest
90 * address not affected by the SF_ERRATA_57 workaround. It should
91 * remain 0 if the workaround is not needed.
92 */
93 #if defined(SF_ERRATA_57)
94 caddr_t errata57_limit;
95 #endif

97 extern void page_relocate_hash(page_t *, page_t *);

99 /*
100 * these must be defined in platform specific areas
101 */
102 extern void map_addr_proc(caddr_t *, size_t, offset_t, int, caddr_t,
103 struct proc *, uint_t);
104 extern page_t *page_get_freelist(struct vnode *, u_offset_t, struct seg *,
105 caddr_t, size_t, uint_t, struct lgrp *);
106 /*
107 * Convert page frame number to an OBMEM page frame number
108 * (i.e. put in the type bits -- zero for this implementation)
109 */
110 pfn_t
111 impl_obmem_pfnnum(pfn_t pf)
112 {
113 return (pf);
114 }

116 /*
117 * Use physmax to determine the highest physical page of DRAM memory
118 * It is assumed that any physical addresses above physmax is in IO space.
119 * We don't bother checking the low end because we assume that memory space
120 * begins at physical page frame 0.
121 *
122 * Return 1 if the page frame is onboard DRAM memory, else 0.
123 * Returns 0 for nvram so it won't be cached.
124 */

```

```

125 int
126 pf_is_memory(pfn_t pf)
127 {
128     /* We must be IO space */
129     if (pf > physmax)
130         return (0);
131
132     /* We must be memory space */
133     return (1);
134 }
135
136 /*
137  * Handle a pagefault.
138  */
139 faultcode_t
140 pagefault(caddr_t addr, enum fault_type type, enum seg_rw rw, int iskernel)
141 {
142     struct as *as;
143     struct proc *p;
144     faultcode_t res;
145     caddr_t base;
146     size_t len;
147     int err;
148
149     if (INVALID_VADDR(addr))
150         return (FC_NOMAP);
151
152     if (iskernel) {
153         as = &kas;
154     } else {
155         p = curproc;
156         as = p->p_as;
157 #if defined(SF_ERRATA_57)
158         /*
159          * Prevent infinite loops due to a segment driver
160          * setting the execute permissions and the sfmmu hat
161          * silently ignoring them.
162          */
163         if (rw == S_EXEC && AS_TYPE_64BIT(as) &&
164             addr < errata57_limit) {
165             res = FC_NOMAP;
166             goto out;
167         }
168 #endif
169     }
170
171     /*
172     * Dispatch pagefault.
173     */
174     res = as_fault(as->a_hat, as, addr, 1, type, rw);
175
176     /*
177     * If this isn't a potential unmapped hole in the user's
178     * UNIX data or stack segments, just return status info.
179     */
180     if (!(res == FC_NOMAP && iskernel == 0))
181         goto out;
182
183     /*
184     * Check to see if we happened to faulted on a currently unmapped
185     * part of the UNIX data or stack segments. If so, create a zfod
186     * mapping there and then try calling the fault routine again.
187     */
188     base = p->p_brkbase;
189     len = p->p_brksize;

```

```

191     if (addr < base || addr >= base + len) { /* data seg? */
192         base = (caddr_t)(p->p_usrstack - p->p_stksize);
193         len = p->p_stksize;
194         if (addr < base || addr >= p->p_usrstack) { /* stack seg? */
195             /* not in either UNIX data or stack segments */
196             res = FC_NOMAP;
197             goto out;
198         }
199     }
200
201     /* the rest of this function implements a 3.X 4.X 5.X compatibility */
202     /* This code is probably not needed anymore */
203
204     /* expand the gap to the page boundaries on each side */
205     len = (((uintptr_t)base + len + PAGEOFFSET) & PAGEMASK) -
206           ((uintptr_t)base & PAGEMASK);
207     base = (caddr_t)((uintptr_t)base & PAGEMASK);
208
209     as_rangelock(as);
210     as_purge(as);
211     if (as_gap(as, PAGE_SIZE, &base, &len, AH_CONTAIN, addr) == 0) {
212         err = as_map(as, base, len, segvn_create, zfod_argsp);
213         as_rangeunlock(as);
214         if (err) {
215             res = FC_MAKE_ERR(err);
216             goto out;
217         }
218     } else {
219         /*
220          * This page is already mapped by another thread after we
221          * returned from as_fault() above. We just fallthrough
222          * as_fault() below.
223          */
224         as_rangeunlock(as);
225     }
226
227     res = as_fault(as->a_hat, as, addr, 1, F_INVAL, rw);
228
229 out:
230
231     return (res);
232 }
233
234 /*
235  * This is the routine which defines the address limit implied
236  * by the flag 'MAP_LOW32'. USERLIMIT32 matches the highest
237  * mappable address in a 32-bit process on this platform (though
238  * perhaps we should make it be UINT32_MAX here?)
239  */
240 void
241 map_addr(caddr_t *addrp, size_t len, offset_t off, int vacalign, uint_t flags)
242 {
243     struct proc *p = curproc;
244     caddr_t userlimit = flags & MAP_LOW32 ?
245         (caddr_t)USERLIMIT32 : p->p_as->a_userlimit;
246     map_addr_proc(addrp, len, off, vacalign, userlimit, p, flags);
247 }
248
249 /*
250  * Some V9 CPUs have holes in the middle of the 64-bit virtual address range.
251  */
252 caddr_t hole_start, hole_end;
253
254 /*
255  * kpm mapping window
256  */

```

```

257 caddr_t kpm_vbase;
258 size_t kpm_size;
259 uchar_t kpm_size_shift;

261 int valid_va_range_aligned_wraparound;
262 /*
263  * Determine whether [*basep, *basep + *lenp) contains a mappable range of
264  * addresses at least "minlen" long, where the base of the range is at "off"
265  * phase from an "align" boundary and there is space for a "redzone"-sized
266  * redzone on either side of the range. On success, 1 is returned and *basep
267  * and *lenp are adjusted to describe the acceptable range (including
268  * the redzone). On failure, 0 is returned.
269  */
270 int
271 valid_va_range_aligned(caddr_t *basep, size_t *lenp, size_t minlen, int dir,
272 size_t align, size_t redzone, size_t off)
273 {
274     caddr_t hi, lo;
275     size_t tot_len;

277     ASSERT(align == 0 ? off == 0 : off < align);
278     ASSERT(ISP2(align));
279     ASSERT(align == 0 || align >= PAGESIZE);

281     lo = *basep;
282     hi = lo + *lenp;
283     tot_len = minlen + 2 * redzone; /* need at least this much space */

285     /* If hi rolled over the top try cutting back. */
286     if (hi < lo) {
287         *lenp = 0UL - (uintptr_t)lo - 1UL;
288         /* Trying to see if this really happens, and then if so, why */
289         valid_va_range_aligned_wraparound++;
290         hi = lo + *lenp;
291     }
292     if (*lenp < tot_len) {
293         return (0);
294     }

296     /*
297      * Deal with a possible hole in the address range between
298      * hole_start and hole_end that should never be mapped by the MMU.
299      */

301     if (lo < hole_start) {
302         if (hi > hole_start)
303             if (hi < hole_end)
304                 hi = hole_start;
305             else
306                 /* lo < hole_start && hi >= hole_end */
307                 if (dir == AH_LO) {
308                     /*
309                      * prefer lowest range
310                      */
311                     if (hole_start - lo >= tot_len)
312                         hi = hole_start;
313                     else if (hi - hole_end >= tot_len)
314                         lo = hole_end;
315                     else
316                         return (0);
317                 } else {
318                     /*
319                      * prefer highest range
320                      */
321                     if (hi - hole_end >= tot_len)
322                         lo = hole_end;

```

```

323     else if (hole_start - lo >= tot_len)
324         hi = hole_start;
325     else
326         return (0);
327     }
328     } else {
329         /* lo >= hole_start */
330         if (hi < hole_end)
331             return (0);
332         if (lo < hole_end)
333             lo = hole_end;
334     }

336     /* Check if remaining length is too small */
337     if (hi - lo < tot_len) {
338         return (0);
339     }
340     if (align > 1) {
341         caddr_t tlo = lo + redzone;
342         caddr_t thi = hi - redzone;
343         tlo = (caddr_t)P2PHASEUP((uintptr_t)tlo, align, off);
344         if (tlo < lo + redzone) {
345             return (0);
346         }
347         if (thi < tlo || thi - tlo < minlen) {
348             return (0);
349         }
350     }
351     *basep = lo;
352     *lenp = hi - lo;
353     return (1);
354 }

356 /*
357  * Determine whether [*basep, *basep + *lenp) contains a mappable range of
358  * addresses at least "minlen" long. On success, 1 is returned and *basep
359  * and *lenp are adjusted to describe the acceptable range. On failure, 0
360  * is returned.
361  */
362 int
363 valid_va_range(caddr_t *basep, size_t *lenp, size_t minlen, int dir)
364 {
365     return (valid_va_range_aligned(basep, lenp, minlen, dir, 0, 0, 0));
366 }

368 /*
369  * Default to forbidding the first 64k of address space. This protects most
370  * reasonably sized structures from dereferences through NULL:
371  * ((foo_t *)0)->bar
372  */
373 uintptr_t forbidden_null_mapping_sz = 0x10000;

375 /*
376 #endif /* ! codereview */
377  * Determine whether [addr, addr+len] with protections 'prot' are valid
378  * for a user address space.
379  */
380 /*ARGSUSED*/
381 int
382 valid_usr_range(caddr_t addr, size_t len, uint_t prot, struct as *as,
383 caddr_t userlimit)
384 {
385     caddr_t eaddr = addr + len;

387     if (eaddr <= addr || addr >= userlimit || eaddr > userlimit)
388         return (RANGE_BADADDR);

```

```

390     if ((addr <= (caddr_t)forbidden_null_mapping_sz) &&
391         secflag_enabled(as->a_proc, PROC_SEC_FORBIDNULLMAP))
392         return (RANGE_BADADDR);

394 #endif /* ! codereview */
395 /*
396  * Determine if the address range falls within an illegal
397  * range of the MMU.
398  */
399     if (eaddr > hole_start && addr < hole_end)
400         return (RANGE_BADADDR);

402 #if defined(SF_ERRATA_57)
403 /*
404  * Make sure USERLIMIT isn't raised too high
405  */
406     ASSERT64(addr <= (caddr_t)0xffffffff80000000ul ||
407             errata57_limit == 0);

409     if (AS_TYPE_64BIT(as) &&
410         (addr < errata57_limit) &&
411         (prot & PROT_EXEC))
412         return (RANGE_BADPROT);
413 #endif /* SF_ERRATA57 */
414     return (RANGE_OKAY);
415 }

417 /*
418  * Routine used to check to see if an a.out can be executed
419  * by the current machine/architecture.
420  */
421 int
422 chkaout(struct exdata *exp)
423 {
424     if (exp->ux_mach == M_SPARC)
425         return (0);
426     else
427         return (ENOEXEC);
428 }

430 /*
431  * The following functions return information about an a.out
432  * which is used when a program is executed.
433  */

435 /*
436  * Return the load memory address for the data segment.
437  */
438 caddr_t
439 getdmem(struct exec *exp)
440 {
441     /*
442      * XXX - Sparc Reference Hack approaching
443      * Remember that we are loading
444      * 8k executables into a 4k machine
445      * DATA_ALIGN == 2 * PAGE_SIZE
446      */
447     if (exp->a_text)
448         return ((caddr_t)(roundup(USRTEXT + exp->a_text, DATA_ALIGN)));
449     else
450         return ((caddr_t)USRTEXT);
451 }

453 /*
454  * Return the starting disk address for the data segment.

```

```

455 */
456 ulong_t
457 getdfile(struct exec *exp)
458 {
459     if (exp->a_magic == ZMAGIC)
460         return (exp->a_text);
461     else
462         return (sizeof (struct exec) + exp->a_text);
463 }

465 /*
466  * Return the load memory address for the text segment.
467  */

469 /*ARGSUSED*/
470 caddr_t
471 gettmem(struct exec *exp)
472 {
473     return ((caddr_t)USRTEXT);
474 }

476 /*
477  * Return the file byte offset for the text segment.
478  */
479 uint_t
480 gettfile(struct exec *exp)
481 {
482     if (exp->a_magic == ZMAGIC)
483         return (0);
484     else
485         return (sizeof (struct exec));
486 }

488 void
489 getexinfo(
490     struct exdata *edp_in,
491     struct exdata *edp_out,
492     int *pagetext,
493     int *pagedata)
494 {
495     *edp_out = *edp_in; /* structure copy */

497     if ((edp_in->ux_mag == ZMAGIC) &&
498         ((edp_in->vp->v_flag & VNOMAP) == 0)) {
499         *pagetext = 1;
500         *pagedata = 1;
501     } else {
502         *pagetext = 0;
503         *pagedata = 0;
504     }
505 }

507 /*
508  * Return non 0 value if the address may cause a VAC alias with KPM mappings.
509  * KPM selects an address such that it's equal offset modulo shm_alignment and
510  * assumes it can't be in VAC conflict with any larger than PAGE_SIZE mapping.
511  */
512 int
513 map_addr_vacalign_check(caddr_t addr, u_offset_t off)
514 {
515     if (vac) {
516         return (((uintptr_t)addr ^ off) & shm_alignment - 1);
517     } else {
518         return (0);
519     }
520 }

```

```

522 /*
523  * Sanity control. Don't use large pages regardless of user
524  * settings if there's less than priv or shm_lpg_min_physmem memory installed.
525  * The units for this variable is 8K pages.
526  */
527 pgcnt_t shm_lpg_min_physmem = 131072;          /* 1GB */
528 pgcnt_t privm_lpg_min_physmem = 131072;      /* 1GB */

530 static size_t
531 map_pgszheap(struct proc *p, caddr_t addr, size_t len)
532 {
533     size_t      pgsz = MMU_PAGESIZE;
534     int         szc;

536     /*
537      * If len is zero, retrieve from proc and don't demote the page size.
538      * Use atleast the default pagesize.
539      */
540     if (len == 0) {
541         len = p->p_brkbase + p->p_brksize - p->p_bssbase;
542     }
543     len = MAX(len, default_uheap_lpsize);

545     for (szc = mmu_page_sizes - 1; szc >= 0; szc--) {
546         pgsz = hw_page_array[szc].hp_size;
547         if ((disable_auto_data_large_pages & (1 << szc)) ||
548             pgsz > max_uheap_lpsize)
549             continue;
550         if (len >= pgsz) {
551             break;
552         }
553     }

555     /*
556      * If addr == 0 we were called by memcntl() when the
557      * size code is 0. Don't set pgsz less than current size.
558      */
559     if (addr == 0 && (pgsz < hw_page_array[p->p_brkpageszc].hp_size)) {
560         pgsz = hw_page_array[p->p_brkpageszc].hp_size;
561     }

563     return (pgsz);
564 }

566 static size_t
567 map_pgszstk(struct proc *p, caddr_t addr, size_t len)
568 {
569     size_t      pgsz = MMU_PAGESIZE;
570     int         szc;

572     /*
573      * If len is zero, retrieve from proc and don't demote the page size.
574      * Use atleast the default pagesize.
575      */
576     if (len == 0) {
577         len = p->p_stksize;
578     }
579     len = MAX(len, default_ustack_lpsize);

581     for (szc = mmu_page_sizes - 1; szc >= 0; szc--) {
582         pgsz = hw_page_array[szc].hp_size;
583         if ((disable_auto_data_large_pages & (1 << szc)) ||
584             pgsz > max_ustack_lpsize)
585             continue;
586         if (len >= pgsz) {

```

```

587         break;
588     }
589 }

591 /*
592  * If addr == 0 we were called by memcntl() or exec_args() when the
593  * size code is 0. Don't set pgsz less than current size.
594  */
595     if (addr == 0 && (pgsz < hw_page_array[p->p_stkpageszc].hp_size)) {
596         pgsz = hw_page_array[p->p_stkpageszc].hp_size;
597     }

599     return (pgsz);
600 }

602 static size_t
603 map_pgszism(caddr_t addr, size_t len)
604 {
605     uint_t szc;
606     size_t pgsz;

608     for (szc = mmu_page_sizes - 1; szc >= TTE4M; szc--) {
609         if (disable_ism_large_pages & (1 << szc))
610             continue;

612         pgsz = hw_page_array[szc].hp_size;
613         if ((len >= pgsz) && IS_P2ALIGNED(addr, pgsz))
614             return (pgsz);
615     }

617     return (DEFAULT_ISM_PAGESIZE);
618 }

620 /*
621  * Suggest a page size to be used to map a segment of type maptype and length
622  * len. Returns a page size (not a size code).
623  */
624 /* ARGSUSED */
625 size_t
626 map_pgsz(int maptype, struct proc *p, caddr_t addr, size_t len, int memcntl)
627 {
628     size_t pgsz = MMU_PAGESIZE;

630     ASSERT(maptype != MAPPGSZ_VA);

632     if (maptype != MAPPGSZ_ISM && physmem < privm_lpg_min_physmem) {
633         return (MMU_PAGESIZE);
634     }

636     switch (maptype) {
637     case MAPPGSZ_ISM:
638         pgsz = map_pgszism(addr, len);
639         break;

641     case MAPPGSZ_STK:
642         if (max_ustack_lpsize > MMU_PAGESIZE) {
643             pgsz = map_pgszstk(p, addr, len);
644         }
645         break;

647     case MAPPGSZ_HEAP:
648         if (max_uheap_lpsize > MMU_PAGESIZE) {
649             pgsz = map_pgszheap(p, addr, len);
650         }
651         break;
652     }

```

```

653     return (pgsz);
654 }

657 /* assumes TTE8K...TTE4M == szc */

659 static uint_t
660 map_szcvec(caddr_t addr, size_t size, uintptr_t off, int disable_lpgs,
661           size_t max_lpsize, size_t min_physmem)
662 {
663     caddr_t eaddr = addr + size;
664     uint_t szcvec = 0;
665     caddr_t raddr;
666     caddr_t readdr;
667     size_t pgsz;
668     int i;

670     if (physmem < min_physmem || max_lpsize <= MMU_PAGESIZE) {
671         return (0);
672     }
673     for (i = mmu_page_sizes - 1; i > 0; i--) {
674         if (disable_lpgs & (1 << i)) {
675             continue;
676         }
677         pgsz = page_get_pagesize(i);
678         if (pgsz > max_lpsize) {
679             continue;
680         }
681         raddr = (caddr_t)P2ROUNDUP((uintptr_t)addr, pgsz);
682         readdr = (caddr_t)P2ALIGN((uintptr_t)eaddr, pgsz);
683         if (raddr < addr || raddr >= readdr) {
684             continue;
685         }
686         if (P2PHASE((uintptr_t)addr ^ off, pgsz)) {
687             continue;
688         }
689         szcvec |= (1 << i);
690         /*
691          * And or in the remaining enabled page sizes.
692          */
693         szcvec |= P2PHASE(~disable_lpgs, (1 << i));
694         szcvec &= ~1; /* no need to return 8K pagesize */
695         break;
696     }
697     return (szcvec);
698 }

700 /*
701  * Return a bit vector of large page size codes that
702  * can be used to map [addr, addr + len) region.
703  */
704 /* ARGSUSED */
705 uint_t
706 map_pgszcvec(caddr_t addr, size_t size, uintptr_t off, int flags, int type,
707             int memcntl)
708 {
709     if (flags & MAP_TEXT) {
710         return (map_szcvec(addr, size, off,
711                           disable_auto_text_large_pages,
712                           max_utevt_lpsize, shm_lpg_min_physmem));
713     }
714     } else if (flags & MAP_INITDATA) {
715         return (map_szcvec(addr, size, off,
716                           disable_auto_data_large_pages,
717                           max_uidata_lpsize, privm_lpg_min_physmem));

```

```

719     } else if (type == MAPPGSZC_SHM) {
720         return (map_szcvec(addr, size, off,
721                           disable_auto_data_large_pages,
722                           max_shm_lpsize, shm_lpg_min_physmem));
723     }
724     } else if (type == MAPPGSZC_HEAP) {
725         return (map_szcvec(addr, size, off,
726                           disable_auto_data_large_pages,
727                           max_uheap_lpsize, privm_lpg_min_physmem));
728     }
729     } else if (type == MAPPGSZC_STACK) {
730         return (map_szcvec(addr, size, off,
731                           disable_auto_data_large_pages,
732                           max_ustack_lpsize, privm_lpg_min_physmem));
733     }
734     } else {
735         return (map_szcvec(addr, size, off,
736                           disable_auto_data_large_pages,
737                           max_privmap_lpsize, privm_lpg_min_physmem));
738     }
739 }

741 /*
742  * Anchored in the table below are counters used to keep track
743  * of free contiguous physical memory. Each element of the table contains
744  * the array of counters, the size of array which is allocated during
745  * startup based on physmax and a shift value used to convert a pagenum
746  * into a counter array index or vice versa. The table has page size
747  * for rows and region size for columns:
748  *
749  *     page_counters[page_size][region_size]
750  *
751  *     page_size:     TTE size code of pages on page_size freelist.
752  *
753  *     region_size:   TTE size code of a candidate larger page made up
754  *                     made up of contiguous free page_size pages.
755  *
756  * As you go across a page_size row increasing region_size each
757  * element keeps track of how many (region_size - 1) size groups
758  * made up of page_size free pages can be coalesced into a
759  * region_size page. Yuck! Lets try an example:
760  *
761  *     page_counters[1][3] is the table element used for identifying
762  *     candidate 4M pages from contiguous pages off the 64K free list.
763  *     Each index in the page_counters[1][3].array spans 4M. Its the
764  *     number of free 512K size (region_size - 1) groups of contiguous
765  *     64K free pages. So when page_counters[1][3].counters[n] == 8
766  *     we know we have a candidate 4M page made up of 512K size groups
767  *     of 64K free pages.
768  */

770 /*
771  * Per page size free lists. 3rd (max_mem_nodes) and 4th (page coloring bins)
772  * dimensions are allocated dynamically.
773  */
774 page_t ***page_freelists[MMU_PAGE_SIZES][MAX_MEM_TYPES];

776 /*
777  * For now there is only a single size cache list.
778  * Allocated dynamically.
779  */
780 page_t ***page_cachelists[MAX_MEM_TYPES];

782 kmutex_t *fpc_mutex[NPC_Mutex];
783 kmutex_t *cpc_mutex[NPC_Mutex];

```

```

785 /*
786  * Calculate space needed for page freelists and counters
787  */
788 size_t
789 calc_free_pagelist_sz(void)
790 {
791     int szc;
792     size_t alloc_sz, cache_sz, free_sz;
793
794     /*
795      * one cachelist per color, node, and type
796      */
797     cache_sz = (page_get_pagecolors(0) * sizeof (page_t *)) +
798               sizeof (page_t **);
799     cache_sz *= max_mem_nodes * MAX_MEM_TYPES;
800
801     /*
802      * one freelist per size, color, node, and type
803      */
804     free_sz = sizeof (page_t **);
805     for (szc = 0; szc < mmu_page_sizes; szc++)
806         free_sz += sizeof (page_t *) * page_get_pagecolors(szc);
807     free_sz *= max_mem_nodes * MAX_MEM_TYPES;
808
809     alloc_sz = cache_sz + free_sz + page_ctrs_sz();
810     return (alloc_sz);
811 }
812
813 caddr_t
814 alloc_page_freelists(caddr_t alloc_base)
815 {
816     int mnode, mtype;
817     int szc, clr;
818
819     /*
820      * We only support small pages in the cachelist.
821      */
822     for (mtype = 0; mtype < MAX_MEM_TYPES; mtype++) {
823         page_cachelists[mtype] = (page_t ***)alloc_base;
824         alloc_base += (max_mem_nodes * sizeof (page_t **));
825         for (mnode = 0; mnode < max_mem_nodes; mnode++) {
826             page_cachelists[mtype][mnode] = (page_t **)alloc_base;
827             alloc_base +=
828                 (page_get_pagecolors(0) * sizeof (page_t *));
829         }
830     }
831
832     /*
833      * Allocate freelists bins for all
834      * supported page sizes.
835      */
836     for (szc = 0; szc < mmu_page_sizes; szc++) {
837         clr = page_get_pagecolors(szc);
838         for (mtype = 0; mtype < MAX_MEM_TYPES; mtype++) {
839             page_freelists[szc][mtype] = (page_t ***)alloc_base;
840             alloc_base += (max_mem_nodes * sizeof (page_t **));
841             for (mnode = 0; mnode < max_mem_nodes; mnode++) {
842                 page_freelists[szc][mtype][mnode] =
843                     (page_t **)alloc_base;
844                 alloc_base += (clr * (sizeof (page_t *)));
845             }
846         }
847     }
848
849     alloc_base = page_ctrs_alloc(alloc_base);
850     return (alloc_base);

```

```

851 }
852
853 /*
854  * Allocate page_freelists locks for a memnode from the nucleus data
855  * area. This is the first time that mmu_page_sizes is used during
856  * bootup, so check mmu_page_sizes initialization.
857  */
858 int
859 ndata_alloc_page_mutexs(struct memlist *ndata)
860 {
861     size_t alloc_sz;
862     caddr_t alloc_base;
863     int i;
864     void page_coloring_init();
865
866     page_coloring_init();
867     if (&mmu_init_mmu_page_sizes) {
868         if (!mmu_init_mmu_page_sizes(0)) {
869             cmn_err(CE_PANIC, "mmu_page_sizes %d not initialized",
870                  mmu_page_sizes);
871         }
872     }
873     ASSERT(mmu_page_sizes >= DEFAULT_MMU_PAGE_SIZES);
874
875     /* fpc_mutex and cpc_mutex */
876     alloc_sz = 2 * NPC_MUTEX * max_mem_nodes * sizeof (kmutex_t);
877
878     alloc_base = ndata_alloc(ndata, alloc_sz, ecache_alignsize);
879     if (alloc_base == NULL)
880         return (-1);
881
882     ASSERT(((uintptr_t)alloc_base & (ecache_alignsize - 1)) == 0);
883
884     for (i = 0; i < NPC_MUTEX; i++) {
885         fpc_mutex[i] = (kmutex_t *)alloc_base;
886         alloc_base += (sizeof (kmutex_t) * max_mem_nodes);
887         cpc_mutex[i] = (kmutex_t *)alloc_base;
888         alloc_base += (sizeof (kmutex_t) * max_mem_nodes);
889     }
890     return (0);
891 }
892
893 /*
894  * To select our starting bin, we stride through the bins with a stride
895  * of 337. Why 337? It's prime, it's largeish, and it performs well both
896  * in simulation and practice for different workloads on varying cache sizes.
897  */
898 uint32_t color_start_current = 0;
899 uint32_t color_start_stride = 337;
900 int color_start_random = 0;
901
902 /* ARGSUSED */
903 uint_t
904 get_color_start(struct as *as)
905 {
906     uint32_t old, new;
907
908     if (consistent_coloring == 2 || color_start_random) {
909         return ((uint_t)((gettick()) << (vac_shift - MMU_PAGESHIFT)) &
910             (hw_page_array[0].hp_colors - 1));
911     }
912
913     do {
914         old = color_start_current;
915         new = old + (color_start_stride << (vac_shift - MMU_PAGESHIFT));
916     } while (atomic_cas_32(&color_start_current, old, new) != old);

```



```

918     return ((uint_t)(new));
919 }

921 /*
922  * Called once at startup from kphysm_init() -- before memialloc()
923  * is invoked to do the 1st page_free()/page_freelist_add().
924  *
925  * initializes page_colors and page_colors_mask based on ecache_setsize.
926  *
927  * Also initializes the counter locks.
928  */
929 void
930 page_coloring_init()
931 {
932     int     a, i;
933     uint_t colors;

935     if (do_pg_coloring == 0) {
936         page_colors = 1;
937         for (i = 0; i < mmu_page_sizes; i++) {
938             colorequivszc[i] = 0;
939             hw_page_array[i].hp_colors = 1;
940         }
941         return;
942     }

944     /*
945     * Calculate page_colors from ecache_setsize. ecache_setsize contains
946     * the max ecache setsize of all cpus configured in the system or, for
947     * cheetah+ systems, the max possible ecache setsize for all possible
948     * cheetah+ cpus.
949     */
950     page_colors = ecache_setsize / MMU_PAGESIZE;
951     page_colors_mask = page_colors - 1;

953     vac_colors = vac_size / MMU_PAGESIZE;
954     vac_colors_mask = vac_colors - 1;

956     page_coloring_shift = 0;
957     a = ecache_setsize;
958     while (a >>= 1) {
959         page_coloring_shift++;
960     }

962     /* initialize number of colors per page size */
963     for (i = 0; i < mmu_page_sizes; i++) {
964         hw_page_array[i].hp_colors = (page_colors_mask >>
965             (hw_page_array[i].hp_shift - hw_page_array[0].hp_shift))
966             + 1;
967         colorequivszc[i] = 0;
968     }

970     /*
971     * initialize cpu_page_colors if ecache setsizes are homogenous.
972     * cpu_page_colors set to -1 during DR operation or during startup
973     * if setsizes are heterogenous.
974     *
975     * The value of cpu_page_colors determines if additional color bins
976     * need to be checked for a particular color in the page_get routines.
977     */
978     if (cpu_setsize > 0 && cpu_page_colors == 0 &&
979         cpu_setsize < ecache_setsize) {
980         cpu_page_colors = cpu_setsize / MMU_PAGESIZE;
981         a = lowbit(page_colors) - lowbit(cpu_page_colors);
982         ASSERT(a > 0);

```

```

983         ASSERT(a < 16);

985         for (i = 0; i < mmu_page_sizes; i++) {
986             if ((colors = hw_page_array[i].hp_colors) <= 1) {
987                 continue;
988             }
989             while ((colors >> a) == 0)
990                 a--;
991             ASSERT(a >= 0);

993             /* higher 4 bits encodes color equiv mask */
994             colorequivszc[i] = (a << 4);
995         }
996     }

998     /* do cpu specific color initialization */
999     if (&page_coloring_init_cpu) {
1000         page_coloring_init_cpu();
1001     }
1002 }

1004 int
1005 bp_color(struct buf *bp)
1006 {
1007     int color = -1;

1009     if (vac) {
1010         if ((bp->b_flags & B_PAGEIO) != 0) {
1011             color = sfmmu_get_ppvcolor(bp->b_pages);
1012         } else if (bp->b_un.b_addr != NULL) {
1013             color = sfmmu_get_addrvcolor(bp->b_un.b_addr);
1014         }
1015     }
1016     return (color < 0 ? 0 : ptob(color));
1017 }

1019 /*
1020  * Function for flushing D-cache when performing module relocations
1021  * to an alternate mapping. Stubbed out on all platforms except sun4u,
1022  * at least for now.
1023  */
1024 void
1025 dcache_flushall()
1026 {
1027     sfmmu_cache_flushall();
1028 }

1030 static int
1031 kdi_range_overlap(uintptr_t val, size_t sz1, uintptr_t va2, size_t sz2)
1032 {
1033     if (val < va2 && val + sz1 <= va2)
1034         return (0);

1036     if (va2 < val && va2 + sz2 <= val)
1037         return (0);

1039     return (1);
1040 }

1042 /*
1043  * Return the number of bytes, relative to the beginning of a given range, that
1044  * are non-toxic (can be read from and written to with relative impunity).
1045  */
1046 size_t
1047 kdi_range_is_nontoxic(uintptr_t va, size_t sz, int write)
1048 {

```

```
1049 /* OBP reads are harmless, but we don't want people writing there */
1050 if (write && kdi_range_overlap(va, sz, OFW_START_ADDR, OFW_END_ADDR -
1051     OFW_START_ADDR + 1))
1052     return (va < OFW_START_ADDR ? OFW_START_ADDR - va : 0);
1054 if (kdi_range_overlap(va, sz, PIOMAPBASE, PIOMAPSIZE))
1055     return (va < PIOMAPBASE ? PIOMAPBASE - va : 0);
1057 return (sz); /* no overlap */
1058 }
1060 /*
1061 * Minimum physmem required for enabling large pages for kernel heap
1062 * Currently we do not enable lp for kmem on systems with less
1063 * than 1GB of memory. This value can be changed via /etc/system
1064 */
1065 size_t segkmem_lpinphysmem = 0x40000000; /* 1GB */
1067 /*
1068 * this function chooses large page size for kernel heap
1069 */
1070 size_t
1071 get_segkmem_lpsize(size_t lpsize)
1072 {
1073     size_t memtotal = physmem * PAGE_SIZE;
1074     size_t mmusz;
1075     uint_t szc;
1077     if (memtotal < segkmem_lpinphysmem)
1078         return (PAGE_SIZE);
1080     if (plat_lpkmem_is_supported != NULL &&
1081         plat_lpkmem_is_supported() == 0)
1082         return (PAGE_SIZE);
1084     mmusz = mmu_get_kernel_lpsize(lpsize);
1085     szc = page_szc(mmusz);
1087     while (szc) {
1088         if (!(disable_large_pages & (1 << szc)))
1089             return (page_get_pagesize(szc));
1090         szc--;
1091     }
1092     return (PAGE_SIZE);
1093 }
```

new/usr/src/uts/sun4u/vm/mach_vm_dep.c

1

```
*****
11477 Thu Jun 30 22:03:59 2016
new/usr/src/uts/sun4u/vm/mach_vm_dep.c
Code review comments from pmooney (sundry), and igork (screwups in zonecfg refac
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
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 */

26 /* Copyright (c) 1984, 1986, 1987, 1988, 1989 AT&T */
27 /* All Rights Reserved */

29 /*
30 * Portions of this source code were derived from Berkeley 4.3 BSD
31 * under license from the Regents of the University of California.
32 */

34 /*
35 * UNIX machine dependent virtual memory support.
36 */

38 #include <sys/vm.h>
39 #include <sys/exec.h>
40 #include <sys/cmn_err.h>
41 #include <sys/cpu_module.h>
42 #include <sys/cpu.h>
43 #include <sys/elf_SPARC.h>
44 #include <sys/archsystem.h>
45 #include <vm/hat_sfmnu.h>
46 #include <sys/memnode.h>
47 #include <sys/mem_cage.h>
48 #include <vm/vm_dep.h>
49 #include <sys/random.h>
50 #endif /* ! codereview */

52 #if defined(__sparcv9) && defined(SF_ERRATA_57)
53 caddr_t errata57_limit;
54 #endif

56 uint_t page_colors = 0;
57 uint_t page_colors_mask = 0;
```

new/usr/src/uts/sun4u/vm/mach_vm_dep.c

2

```
58 uint_t page_coloring_shift = 0;
59 int consistent_coloring;
60 int update_proc_pgcolorbase_after_fork = 0;

62 uint_t mmu_page_sizes = DEFAULT_MMU_PAGE_SIZES;
63 uint_t max_mmu_page_sizes = MMU_PAGE_SIZES;
64 uint_t mmu_hashcnt = DEFAULT_MAX_HASHCNT;
65 uint_t max_mmu_hashcnt = MAX_HASHCNT;
66 size_t mmu_ism_pagesize = DEFAULT_ISM_PAGESIZE;

68 /*
69  * The sun4u hardware mapping sizes which will always be supported are
70  * 8K, 64K, 512K and 4M. If sun4u based machines need to support other
71  * page sizes, platform or cpu specific routines need to modify the value.
72  * The base pagesize (p_szc == 0) must always be supported by the hardware.
73 */
74 int mmu_exported_pagesize_mask = (1 << TTE8K) | (1 << TTE64K) |
75 (1 << TTE512K) | (1 << TTE4M);
76 uint_t mmu_exported_page_sizes;

78 uint_t szc_2_userszc[MMU_PAGE_SIZES];
79 uint_t userszc_2_szc[MMU_PAGE_SIZES];

81 extern uint_t vac_colors_mask;
82 extern int vac_shift;

84 hw_pagesize_t hw_page_array[] = {
85     {MMU_PAGESIZE, MMU_PAGESHIFT, 0, MMU_PAGESIZE >> MMU_PAGESHIFT},
86     {MMU_PAGESIZE64K, MMU_PAGESHIFT64K, 0,
87      MMU_PAGESIZE64K >> MMU_PAGESHIFT},
88     {MMU_PAGESIZE512K, MMU_PAGESHIFT512K, 0,
89      MMU_PAGESIZE512K >> MMU_PAGESHIFT},
90     {MMU_PAGESIZE4M, MMU_PAGESHIFT4M, 0, MMU_PAGESIZE4M >> MMU_PAGESHIFT},
91     {MMU_PAGESIZE32M, MMU_PAGESHIFT32M, 0,
92      MMU_PAGESIZE32M >> MMU_PAGESHIFT},
93     {MMU_PAGESIZE256M, MMU_PAGESHIFT256M, 0,
94      MMU_PAGESIZE256M >> MMU_PAGESHIFT},
95     {0, 0, 0, 0}
96 };

98 /*
99  * Maximum page size used to map 64-bit memory segment kmem64_base..kmem64_end
100 */
101 int max_bootlp_tteszc = TTE4M;

103 /*
104  * use_text_pgsz64k and use_text_pgsz512k allow the user to turn on these
105  * additional text page sizes for USIII-IV+ and OPL by changing the default
106  * values via /etc/system.
107 */
108 int use_text_pgsz64K = 0;
109 int use_text_pgsz512K = 0;

111 /*
112  * Maximum and default segment size tunables for user heap, stack, private
113  * and shared anonymous memory, and user text and initialized data.
114 */
115 size_t max_uheap_lpsize = MMU_PAGESIZE4M;
116 size_t default_uheap_lpsize = MMU_PAGESIZE;
117 size_t max_ustack_lpsize = MMU_PAGESIZE4M;
118 size_t default_ustack_lpsize = MMU_PAGESIZE;
119 size_t max_privmap_lpsize = MMU_PAGESIZE4M;
120 size_t max_uidata_lpsize = MMU_PAGESIZE;
121 size_t max_utext_lpsize = MMU_PAGESIZE4M;
122 size_t max_shm_lpsize = MMU_PAGESIZE4M;
```

```

124 void
125 adjust_data_maxlpsize(size_t ismpagesize)
126 {
127     if (max_uheap_lpsize == MMU_PAGESIZE4M) {
128         max_uheap_lpsize = ismpagesize;
129     }
130     if (max_ustack_lpsize == MMU_PAGESIZE4M) {
131         max_ustack_lpsize = ismpagesize;
132     }
133     if (max_privmap_lpsize == MMU_PAGESIZE4M) {
134         max_privmap_lpsize = ismpagesize;
135     }
136     if (max_shm_lpsize == MMU_PAGESIZE4M) {
137         max_shm_lpsize = ismpagesize;
138     }
139 }

141 /*
142  * The maximum amount a randomized mapping will be slewed. We should perhaps
143  * arrange things so these tunables can be separate for mmap, mmapobj, and
144  * ld.so
145  */
146 size_t aslr_max_map_skew = 256 * 1024 * 1024; /* 256MB */

148 /*
149 #endif /* !codereview */
150 * map_addr_proc() is the routine called when the system is to
151 * choose an address for the user. We will pick an address
152 * range which is just below the current stack limit. The
153 * algorithm used for cache consistency on machines with virtual
154 * address caches is such that offset 0 in the vnode is always
155 * on a shm_alignment'ed aligned address. Unfortunately, this
156 * means that vnodes which are demand paged will not be mapped
157 * cache consistently with the executable images. When the
158 * cache alignment for a given object is inconsistent, the
159 * lower level code must manage the translations so that this
160 * is not seen here (at the cost of efficiency, of course).
161 *
162 * Every mapping will have a redzone of a single page on either side of
163 * the request. This is done to leave one page unmapped between segments.
164 * This is not required, but it's useful for the user because if their
165 * program strays across a segment boundary, it will catch a fault
166 * immediately making debugging a little easier. Currently the redzone
167 * is mandatory.
168 *
169 *
170 * addrp is a value/result parameter.
171 * On input it is a hint from the user to be used in a completely
172 * machine dependent fashion. For MAP_ALIGN, addrp contains the
173 * minimal alignment, which must be some "power of two" multiple of
174 * pagesize.
175 *
176 * On output it is NULL if no address can be found in the current
177 * processes address space or else an address that is currently
178 * not mapped for len bytes with a page of red zone on either side.
179 * If vacalign is true, then the selected address will obey the alignment
180 * constraints of a vac machine based on the given off value.
181 */
182 /*ARGSUSED4*/
183 void
184 map_addr_proc(caddr_t *addrp, size_t len, offset_t off, int vacalign,
185             caddr_t userlimit, struct proc *p, uint_t flags)
186 {
187     struct as *as = p->p_as;
188     caddr_t addr;
189     caddr_t base;

```

```

190     size_t slen;
191     uintptr_t align_amount;
192     int allow_largepage_alignment = 1;

194     base = p->p_brkbase;
195     if (userlimit < as->a_userlimit) {
196         /*
197          * This happens when a program wants to map something in
198          * a range that's accessible to a program in a smaller
199          * address space. For example, a 64-bit program might
200          * be calling mmap32(2) to guarantee that the returned
201          * address is below 4Gbytes.
202          */
203         ASSERT(userlimit > base);
204         slen = userlimit - base;
205     } else {
206         slen = p->p_usrstack - base -
207             ((p->p_stk_ctl + PAGEOFFSET) & PAGEMASK);
208     }

210     /* Make len be a multiple of PAGE_SIZE */
211     len = (len + PAGEOFFSET) & PAGEMASK;

213     /*
214      * If the request is larger than the size of a particular
215      * mmu level, then we use that level to map the request.
216      * But this requires that both the virtual and the physical
217      * addresses be aligned with respect to that level, so we
218      * do the virtual bit of nastiness here.
219      *
220      * For 32-bit processes, only those which have specified
221      * MAP_ALIGN or an addr will be aligned on a page size > 4MB. Otherwise
222      * we can potentially waste up to 256MB of the 4G process address
223      * space just for alignment.
224      */
225     if (p->p_model == DATAMODEL_ILP32 && ((flags & MAP_ALIGN) == 0 ||
226         ((uintptr_t)*addrp) != 0)) {
227         allow_largepage_alignment = 0;
228     }
229     if ((mmu_page_sizes == max_mmu_page_sizes) &&
230         allow_largepage_alignment &&
231         (len >= MMU_PAGESIZE256M)) { /* 256MB mappings */
232         align_amount = MMU_PAGESIZE256M;
233     } else if ((mmu_page_sizes == max_mmu_page_sizes) &&
234         allow_largepage_alignment &&
235         (len >= MMU_PAGESIZE32M)) { /* 32MB mappings */
236         align_amount = MMU_PAGESIZE32M;
237     } else if (len >= MMU_PAGESIZE4M) { /* 4MB mappings */
238         align_amount = MMU_PAGESIZE4M;
239     } else if (len >= MMU_PAGESIZE512K) { /* 512KB mappings */
240         align_amount = MMU_PAGESIZE512K;
241     } else if (len >= MMU_PAGESIZE64K) { /* 64KB mappings */
242         align_amount = MMU_PAGESIZE64K;
243     } else {
244         /*
245          * Align virtual addresses on a 64K boundary to ensure
246          * that ELF shared libraries are mapped with the appropriate
247          * alignment constraints by the run-time linker.
248          */
249         align_amount = ELF_SPARC_MAXPGSZ;
250         if ((flags & MAP_ALIGN) && ((uintptr_t)*addrp != 0) &&
251             ((uintptr_t)*addrp < align_amount))
252             align_amount = (uintptr_t)*addrp;
253     }

255     /*

```

```

256     * 64-bit processes require 1024K alignment of ELF shared libraries.
257     */
258     if (p->p_model == DATAMODEL_LP64)
259         align_amount = MAX(align_amount, ELF_SPARCV9_MAXPGSZ);
260 #ifndef VAC
261     if (vac && vacalign && (align_amount < shm_alignment))
262         align_amount = shm_alignment;
263 #endif

265     if ((flags & MAP_ALIGN) && ((uintptr_t)*addrp > align_amount)) {
266         align_amount = (uintptr_t)*addrp;
267     }

269     ASSERT(ISP2(align_amount));
270     ASSERT(align_amount == 0 || align_amount >= PAGE_SIZE);

272     /*
273     * Look for a large enough hole starting below the stack limit.
274     * After finding it, use the upper part.
275     */
276     as_purge(as);
277     off = off & (align_amount - 1);

279 #endif /* ! codereview */
280     if (as_gap_aligned(as, len, &base, &slen, AH_HI, NULL, align_amount,
281         PAGE_SIZE, off) == 0) {
282         caddr_t as_addr;

284         /*
285         * addr is the highest possible address to use since we have
286         * a PAGE_SIZE redzone at the beginning and end.
287         */
288         addr = base + slen - (PAGE_SIZE + len);
289         as_addr = addr;
290         /*
291         * Round address DOWN to the alignment amount and
292         * add the offset in.
293         * If addr is greater than as_addr, len would not be large
294         * enough to include the redzone, so we must adjust down
295         * by the alignment amount.
296         */
297         addr = (caddr_t)((uintptr_t)addr & ~(align_amount - 1));
298         addr += (long)off;
299         if (addr > as_addr) {
300             addr -= align_amount;
301         }

303         /*
304         * If randomization is requested, slew the allocation
305         * backwards, within the same gap, by a random amount.
306         */
307         if (flags & _MAP_RANDOMIZE) {
308             uint32_t slew;
309             uint32_t maxslew;

311             (void) random_get_pseudo_bytes((uint8_t *)&slew,
312                 sizeof(slew));

314             maxslew = MIN(aslr_max_map_skew, (addr - base));
315             /*
316             * Don't allow ASLR to cause mappings to fail below
317             * because of SF erratum #57
318             */
319             maxslew = MIN(maxslew, (addr - errata57_limit));

321             slew = slew % maxslew;

```

```

322         addr -= P2ALIGN(slew, align_amount);
323     }

325 #endif /* ! codereview */
326     ASSERT(addr > base);
327     ASSERT(addr + len < base + slen);
328     ASSERT(((uintptr_t)addr & (align_amount - 1)) ==
329         ((uintptr_t)(off)));
330     *addrp = addr;

332 #if defined(SF_ERRATA_57)
333     if (AS_TYPE_64BIT(as) && addr < errata57_limit) {
334         *addrp = NULL;
335     }
336 #endif
337     } else {
338         *addrp = NULL; /* no more virtual space */
339     }
340 }

342 /*
343 * Platform-dependent page scrub call.
344 */
345 void
346 pagescrub(page_t *pp, uint_t off, uint_t len)
347 {
348     /*
349     * For now, we rely on the fact that pagezero() will
350     * always clear UEs.
351     */
352     pagezero(pp, off, len);
353 }

355 /*ARGSUSED*/
356 void
357 sync_data_memory(caddr_t va, size_t len)
358 {
359     cpu_flush_ecache();
360 }

362 /*
363 * platform specific large pages for kernel heap support
364 */
365 void
366 mmu_init_kcontext()
367 {
368     extern void set_kcontextreg();

370     if (kcontextreg)
371         set_kcontextreg();
372 }

374 void
375 contig_mem_init(void)
376 {
377     /* not applicable to sun4u */
378 }

380 /*ARGSUSED*/
381 caddr_t
382 contig_mem_prealloc(caddr_t alloc_base, pgcnt_t npages)
383 {
384     /* not applicable to sun4u */
385     return (alloc_base);
386 }

```

```
50 size_t
51 exec_get_spslew(void)
52 {
53     return (0);
54 }
```

```

*****
25099 Thu Jun 30 22:04:00 2016
new/usr/src/uts/sun4v/vm/mach_vm_dep.c
7029 want per-process exploit mitigation features (secflags)
7030 want basic address space layout randomization (aslr)
7031 noexec_user_stack should be a secflag
7032 want a means to forbid mappings around NULL.
*****
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 */

26 /* Copyright (c) 1984, 1986, 1987, 1988, 1989 AT&T */
27 /* All Rights Reserved */

29 /*
30 * Portions of this source code were derived from Berkeley 4.3 BSD
31 * under license from the Regents of the University of California.
32 */

34 /*
35 * UNIX machine dependent virtual memory support.
36 */

38 #include <sys/vm.h>
39 #include <sys/exec.h>
40 #include <sys/cmn_err.h>
41 #include <sys/cpu_module.h>
42 #include <sys/cpu.h>
43 #include <sys/elf_SPARC.h>
44 #include <sys/archsystem.h>
45 #include <vm/hat_sfmmu.h>
46 #include <sys/memnode.h>
47 #include <sys/mem_cage.h>
48 #include <vm/vm_dep.h>
49 #include <sys/error.h>
50 #include <sys/machsystem.h>
51 #include <vm/seg_kmem.h>
52 #include <sys/stack.h>
53 #include <sys/atomic.h>
54 #include <sys/promif.h>
55 #include <sys/random.h>
56 #endif /* !codereview */

58 uint_t page_colors = 0;

```

```

59 uint_t page_colors_mask = 0;
60 uint_t page_coloring_shift = 0;
61 int consistent_coloring;
62 int update_proc_pgcolorbase_after_fork = 1;

64 uint_t mmu_page_sizes = MMU_PAGE_SIZES;
65 uint_t max_mmu_page_sizes = MMU_PAGE_SIZES;
66 uint_t mmu_hashcnt = MAX_HASHCNT;
67 uint_t max_mmu_hashcnt = MAX_HASHCNT;
68 size_t mmu_ism_pagesize = DEFAULT_ISM_PAGESIZE;

70 /*
71 * A bitmask of the page sizes supported by hardware based upon szc.
72 * The base pagesize (p_szc == 0) must always be supported by the hardware.
73 */
74 int mmu_exported_pagesize_mask;
75 uint_t mmu_exported_page_sizes;

77 uint_t szc_2_userszcs[MMU_PAGE_SIZES];
78 uint_t userszcs_2_szc[MMU_PAGE_SIZES];

80 extern uint_t vac_colors_mask;
81 extern int vac_shift;

83 hw_pagesize_t hw_page_array[] = {
84     {MMU_PAGESIZE, MMU_PAGESHIFT, 0, MMU_PAGESIZE >> MMU_PAGESHIFT},
85     {MMU_PAGESIZE64K, MMU_PAGESHIFT64K, 0,
86      MMU_PAGESIZE64K >> MMU_PAGESHIFT},
87     {MMU_PAGESIZE512K, MMU_PAGESHIFT512K, 0,
88      MMU_PAGESIZE512K >> MMU_PAGESHIFT},
89     {MMU_PAGESIZE4M, MMU_PAGESHIFT4M, 0, MMU_PAGESIZE4M >> MMU_PAGESHIFT},
90     {MMU_PAGESIZE32M, MMU_PAGESHIFT32M, 0,
91      MMU_PAGESIZE32M >> MMU_PAGESHIFT},
92     {MMU_PAGESIZE256M, MMU_PAGESHIFT256M, 0,
93      MMU_PAGESIZE256M >> MMU_PAGESHIFT},
94     {0, 0, 0, 0}
95 };

97 /*
98 * Maximum page size used to map 64-bit memory segment kmem64_base..kmem64_end
99 */
100 int max_bootlp_tteszc = TTE256M;

102 /*
103 * Maximum and default segment size tunables for user heap, stack, private
104 * and shared anonymous memory, and user text and initialized data.
105 */
106 size_t max_uheap_lpsize = MMU_PAGESIZE64K;
107 size_t default_uheap_lpsize = MMU_PAGESIZE64K;
108 size_t max_ustack_lpsize = MMU_PAGESIZE64K;
109 size_t default_ustack_lpsize = MMU_PAGESIZE64K;
110 size_t max_privmap_lpsize = MMU_PAGESIZE64K;
111 size_t max_uidata_lpsize = MMU_PAGESIZE64K;
112 size_t max_utext_lpsize = MMU_PAGESIZE4M;
113 size_t max_shm_lpsize = MMU_PAGESIZE4M;

115 /*
116 * Contiguous memory allocator data structures and variables.
117 *
118 * The sun4v kernel must provide a means to allocate physically
119 * contiguous, non-relocatable memory. The contig_mem_arena
120 * and contig_mem_slab_arena exist for this purpose. Allocations
121 * that require physically contiguous non-relocatable memory should
122 * be made using contig_mem_alloc() or contig_mem_alloc_align()
123 * which return memory from contig_mem_arena or contig_mem_reloc_arena.
124 * These arenas import memory from the contig_mem_slab_arena one

```

```

125 * contiguous chunk at a time.
126 *
127 * When importing slabs, an attempt is made to allocate a large page
128 * to use as backing. As a result of the non-relocatable requirement,
129 * slabs are allocated from the kernel cage freelists. If the cage does
130 * not contain any free contiguous chunks large enough to satisfy the
131 * slab allocation, the slab size will be downsized and the operation
132 * retried. Large slab sizes are tried first to minimize cage
133 * fragmentation. If the slab allocation is unsuccessful still, the slab
134 * is allocated from outside the kernel cage. This is undesirable because,
135 * until slabs are freed, it results in non-relocatable chunks scattered
136 * throughout physical memory.
137 *
138 * Allocations from the contig_mem_arena are backed by slabs from the
139 * cage. Allocations from the contig_mem_reloc_arena are backed by
140 * slabs allocated outside the cage. Slabs are left share locked while
141 * in use to prevent non-cage slabs from being relocated.
142 *
143 * Since there is no guarantee that large pages will be available in
144 * the kernel cage, contiguous memory is reserved and added to the
145 * contig_mem_arena at boot time, making it available for later
146 * contiguous memory allocations. This reserve will be used to satisfy
147 * contig_mem allocations first and it is only when the reserve is
148 * completely allocated that new slabs will need to be imported.
149 */
150 static vmem_t      *contig_mem_slab_arena;
151 static vmem_t      *contig_mem_arena;
152 static vmem_t      *contig_mem_reloc_arena;
153 static kmutex_t    contig_mem_lock;
154 #define CONTIG_MEM_ARENA_QUANTUM      64
155 #define CONTIG_MEM_SLAB_ARENA_QUANTUM MMU_PAGESIZE64K

157 /* contig_mem_arena import slab sizes, in decreasing size order */
158 static size_t contig_mem_import_sizes[] = {
159     MMU_PAGESIZE4M,
160     MMU_PAGESIZE512K,
161     MMU_PAGESIZE64K
162 };
163 #define NUM_IMPORT_SIZES \
164     (sizeof (contig_mem_import_sizes) / sizeof (size_t))
165 static size_t contig_mem_import_size_max = MMU_PAGESIZE4M;
166 size_t contig_mem_slab_size = MMU_PAGESIZE4M;

168 /* Boot-time allocated buffer to pre-populate the contig_mem_arena */
169 static size_t contig_mem_prealloc_size;
170 static void *contig_mem_prealloc_buf;

172 /*
173 * The maximum amount a randomized mapping will be slewed. We should perhaps
174 * arrange things so these tunables can be separate for mmap, mmapobj, and
175 * ld.so
176 */
177 size_t aslr_max_map_skew = 256 * 1024 * 1024; /* 256MB */

179 /*
180 #endif /* ! codereview */
181 * map_addr_proc() is the routine called when the system is to
182 * choose an address for the user. We will pick an address
183 * range which is just below the current stack limit. The
184 * algorithm used for cache consistency on machines with virtual
185 * address caches is such that offset 0 in the vnode is always
186 * on a shm_alignment'ed aligned address. Unfortunately, this
187 * means that vnodes which are demand paged will not be mapped
188 * cache consistently with the executable images. When the
189 * cache alignment for a given object is inconsistent, the
190 * lower level code must manage the translations so that this

```

```

191 * is not seen here (at the cost of efficiency, of course).
192 *
193 * Every mapping will have a redzone of a single page on either side of
194 * the request. This is done to leave one page unmapped between segments.
195 * This is not required, but it's useful for the user because if their
196 * program strays across a segment boundary, it will catch a fault
197 * immediately making debugging a little easier. Currently the redzone
198 * is mandatory.
199 *
200 * addrp is a value/result parameter.
201 * On input it is a hint from the user to be used in a completely
202 * machine dependent fashion. For MAP_ALIGN, addrp contains the
203 * minimal alignment, which must be some "power of two" multiple of
204 * pagesize.
205 *
206 * On output it is NULL if no address can be found in the current
207 * processes address space or else an address that is currently
208 * not mapped for len bytes with a page of red zone on either side.
209 * If vacalign is true, then the selected address will obey the alignment
210 * constraints of a vac machine based on the given off value.
211 */
212 /*ARGSUSED3*/
213 void
214 map_addr_proc(caddr_t *addrp, size_t len, offset_t off, int vacalign,
215     caddr_t userlimit, struct proc *p, uint_t flags)
216 {
217     struct as *as = p->p_as;
218     caddr_t addr;
219     caddr_t base;
220     size_t slen;
221     uintptr_t align_amount;
222     int allow_largepage_alignment = 1;

224     base = p->p_brkbase;
225     if (userlimit < as->a_userlimit) {
226         /*
227          * This happens when a program wants to map something in
228          * a range that's accessible to a program in a smaller
229          * address space. For example, a 64-bit program might
230          * be calling mmap32(2) to guarantee that the returned
231          * address is below 4Gbytes.
232          */
233         ASSERT(userlimit > base);
234         slen = userlimit - base;
235     } else {
236         slen = p->p_usrstack - base -
237             ((p->p_stk_ctl + PAGEOFFSET) & PAGEMASK);
238     }
239     /* Make len be a multiple of PAGESIZE */
240     len = (len + PAGEOFFSET) & PAGEMASK;

242     /*
243     * If the request is larger than the size of a particular
244     * mmu level, then we use that level to map the request.
245     * But this requires that both the virtual and the physical
246     * addresses be aligned with respect to that level, so we
247     * do the virtual bit of nastiness here.
248     *
249     * For 32-bit processes, only those which have specified
250     * MAP_ALIGN or an addr will be aligned on a page size > 4MB. Otherwise
251     * we can potentially waste up to 256MB of the 4G process address
252     * space just for alignment.
253     *
254     * XXXQ Should iterate through hw_page_array here to catch
255     * all supported pagesizes
256     */

```



```

257     if (p->p_model == DATAMODEL_ILP32 && ((flags & MAP_ALIGN) == 0 ||
258         ((uintptr_t)*addrp) != 0)) {
259         allow_largepage_alignment = 0;
260     }
261     if ((mmu_page_sizes == max_mmu_page_sizes) &&
262         allow_largepage_alignment &&
263         (len >= MMU_PAGESIZE256M)) { /* 256MB mappings */
264         align_amount = MMU_PAGESIZE256M;
265     } else if ((mmu_page_sizes == max_mmu_page_sizes) &&
266         allow_largepage_alignment &&
267         (len >= MMU_PAGESIZE32M)) { /* 32MB mappings */
268         align_amount = MMU_PAGESIZE32M;
269     } else if (len >= MMU_PAGESIZE4M) { /* 4MB mappings */
270         align_amount = MMU_PAGESIZE4M;
271     } else if (len >= MMU_PAGESIZE512K) { /* 512KB mappings */
272         align_amount = MMU_PAGESIZE512K;
273     } else if (len >= MMU_PAGESIZE64K) { /* 64KB mappings */
274         align_amount = MMU_PAGESIZE64K;
275     } else {
276         /*
277          * Align virtual addresses on a 64K boundary to ensure
278          * that ELF shared libraries are mapped with the appropriate
279          * alignment constraints by the run-time linker.
280          */
281         align_amount = ELF_SPARC_MAXPGSZ;
282         if ((flags & MAP_ALIGN) && ((uintptr_t)*addrp != 0) &&
283             ((uintptr_t)*addrp < align_amount))
284             align_amount = (uintptr_t)*addrp;
285     }
286
287     /*
288      * 64-bit processes require 1024K alignment of ELF shared libraries.
289      */
290     if (p->p_model == DATAMODEL_LP64)
291         align_amount = MAX(align_amount, ELF_SPARCV9_MAXPGSZ);
292 #ifdef VAC
293     if (vac && vacalign && (align_amount < shm_alignment))
294         align_amount = shm_alignment;
295 #endif
296
297     if ((flags & MAP_ALIGN) && ((uintptr_t)*addrp > align_amount)) {
298         align_amount = (uintptr_t)*addrp;
299     }
300
301     ASSERT(ISP2(align_amount));
302     ASSERT(align_amount == 0 || align_amount >= PAGESIZE);
303
304     /*
305      * Look for a large enough hole starting below the stack limit.
306      * After finding it, use the upper part.
307      */
308     as_purge(as);
309     off = off & (align_amount - 1);
310     if (as_gap_aligned(as, len, &base, &slen, AH_HI, NULL, align_amount,
311         PAGESIZE, off) == 0) {
312         caddr_t as_addr;
313
314         /*
315          * addr is the highest possible address to use since we have
316          * a PAGESIZE redzone at the beginning and end.
317          */
318         addr = base + slen - (PAGESIZE + len);
319         as_addr = addr;
320     }
321     /*
322      * Round address DOWN to the alignment amount and
323      * add the offset in.

```

```

323     * If addr is greater than as_addr, len would not be large
324     * enough to include the redzone, so we must adjust down
325     * by the alignment amount.
326     */
327     addr = (caddr_t)((uintptr_t)addr & ~(align_amount - 1));
328     addr += (long)off;
329     if (addr > as_addr) {
330         addr -= align_amount;
331     }
332
333     /*
334     * If randomization is requested, slew the allocation
335     * backwards, within the same gap, by a random amount.
336     */
337     if (flags & _MAP_RANDOMIZE) {
338         uint32_t slew;
339
340         (void) random_get_pseudo_bytes((uint8_t *)&slew,
341             sizeof (slew));
342
343         slew = slew % MIN(aslr_max_map_skew, (addr - base));
344         addr -= P2ALIGN(slew, align_amount);
345     }
346
347 #endif /* ! codereview */
348     ASSERT(addr > base);
349     ASSERT(addr + len < base + slen);
350     ASSERT(((uintptr_t)addr & (align_amount - 1)) ==
351         ((uintptr_t)(off)));
352     *addrp = addr;
353
354     } else {
355         *addrp = NULL; /* no more virtual space */
356     }
357 }
358
359 /*
360  * Platform-dependent page scrub call.
361  * We call hypervisor to scrub the page.
362  */
363 void
364 pagescrub(page_t *pp, uint_t off, uint_t len)
365 {
366     uint64_t pa, length;
367
368     pa = (uint64_t)(pp->p_pagenum << MMU_PAGESHIFT + off);
369     length = (uint64_t)len;
370
371     (void) mem_scrub(pa, length);
372 }
373
374 void
375 sync_data_memory(caddr_t va, size_t len)
376 {
377     /* Call memory sync function */
378     (void) mem_sync(va, len);
379 }
380
381 size_t
382 mmu_get_kernel_lpsize(size_t lpsize)
383 {
384     extern int mmu_exported_pagesize_mask;
385     uint_t tte;
386
387     if (lpsize == 0) {
388         /* no setting for segkmem_lpsize in /etc/system: use default */

```

```

389     if (mmu_exported_pagesize_mask & (1 << TTE256M)) {
390         lpsize = MMU_PAGESIZE256M;
391     } else if (mmu_exported_pagesize_mask & (1 << TTE4M)) {
392         lpsize = MMU_PAGESIZE4M;
393     } else if (mmu_exported_pagesize_mask & (1 << TTE64K)) {
394         lpsize = MMU_PAGESIZE64K;
395     } else {
396         lpsize = MMU_PAGESIZE;
397     }
399     return (lpsize);
400 }
402 for (tte = TTE8K; tte <= TTE256M; tte++) {
404     if ((mmu_exported_pagesize_mask & (1 << tte)) == 0)
405         continue;
407     if (lpsize == TTEBYTES(tte))
408         return (lpsize);
409 }
411 lpsize = TTEBYTES(TTE8K);
412 return (lpsize);
413 }
415 void
416 mmu_init_kcontext()
417 {
418 }
420 /*ARGSUSED*/
421 void
422 mmu_init_kernel_pgsz(struct hat *hat)
423 {
424 }
426 static void *
427 contig_mem_span_alloc(vmem_t *vmp, size_t size, int vmflag)
428 {
429     page_t *ppl;
430     page_t *rootpp;
431     caddr_t addr = NULL;
432     pgcnt_t npages = btopr(size);
433     page_t **ppa;
434     int pgflags;
435     spgcnt_t i = 0;
438     ASSERT(size <= contig_mem_import_size_max);
439     ASSERT((size & (size - 1)) == 0);
441     if ((addr = vmem_xalloc(vmp, size, size, 0, 0,
442         NULL, NULL, vmflag)) == NULL) {
443         return (NULL);
444     }
446     /* The address should be slab-size aligned. */
447     ASSERT(((uintptr_t)addr & (size - 1)) == 0);
449     if (page_resv(npages, vmflag & VM_KMFLAGS) == 0) {
450         vmem_xfree(vmp, addr, size);
451         return (NULL);
452     }
454     pgflags = PG_EXCL;

```

```

455     if (vmflag & VM_NOELOC)
456         pgflags |= PG_NOELOC;
458     ppl = page_create_va_large(&kvp, (u_offset_t)(uintptr_t)addr, size,
459         pgflags, &kvseg, addr, NULL);
461     if (ppl == NULL) {
462         vmem_xfree(vmp, addr, size);
463         page_unresv(npages);
464         return (NULL);
465     }
467     rootpp = ppl;
468     ppa = kmem_zalloc(npages * sizeof (page_t *), KM_SLEEP);
469     while (ppl != NULL) {
470         page_t *pp = ppl;
471         ppa[i++] = pp;
472         page_sub(&ppl, pp);
473         ASSERT(page_iolock_assert(pp));
474         ASSERT(PAGE_EXCL(pp));
475         page_io_unlock(pp);
476     }
478     /*
479      * Load the locked entry. It's OK to preload the entry into
480      * the TSB since we now support large mappings in the kernel TSB.
481      */
482     hat_memload_array(kas.a_hat, (caddr_t)rootpp->p_offset, size,
483         ppa, (PROT_ALL & ~PROT_USER) | HAT_NOSYNC, HAT_LOAD_LOCK);
485     ASSERT(i == page_get_pagecnt(ppa[0]->p_szc));
486     for (--i; i >= 0; --i) {
487         ASSERT(ppa[i]->p_szc == ppa[0]->p_szc);
488         ASSERT(page_pptonum(ppa[i]) == page_pptonum(ppa[0]) + i);
489         (void) page_pp_lock(ppa[i], 0, 1);
490         /*
491          * Leave the page share locked. For non-cage pages,
492          * this would prevent memory DR if it were supported
493          * on sun4v.
494          */
495         page_downgrade(ppa[i]);
496     }
498     kmem_free(ppa, npages * sizeof (page_t *));
499     return (addr);
500 }
502 /*
503  * Allocates a slab by first trying to use the largest slab size
504  * in contig_mem_import_sizes and then falling back to smaller slab
505  * sizes still large enough for the allocation. The sizep argument
506  * is a pointer to the requested size. When a slab is successfully
507  * allocated, the slab size, which must be >= *sizep and <=
508  * contig_mem_import_size_max, is returned in the *sizep argument.
509  * Returns the virtual address of the new slab.
510  */
511 static void *
512 span_alloc_downsize(vmem_t *vmp, size_t *sizep, size_t align, int vmflag)
513 {
514     int i;
516     ASSERT(*sizep <= contig_mem_import_size_max);
518     for (i = 0; i < NUM_IMPORT_SIZES; i++) {
519         size_t page_size = contig_mem_import_sizes[i];

```

```

521     /*
522     * Check that the alignment is also less than the
523     * import (large page) size. In the case where the
524     * alignment is larger than the size, a large page
525     * large enough for the allocation is not necessarily
526     * physical-address aligned to satisfy the requested
527     * alignment. Since alignment is required to be a
528     * power-of-2, any large page >= size && >= align will
529     * suffice.
530     */
531     if (*sizep <= page_size && align <= page_size) {
532         void *addr;
533         addr = contig_mem_span_alloc(vmp, page_size, vmflag);
534         if (addr == NULL)
535             continue;
536         *sizep = page_size;
537         return (addr);
538     }
539     return (NULL);
540 }

542     return (NULL);
543 }

545 static void *
546 contig_mem_span_xalloc(vmem_t *vmp, size_t *sizep, size_t align, int vmflag)
547 {
548     return (span_alloc_downsize(vmp, sizep, align, vmflag | VM_NORELOC));
549 }

551 static void *
552 contig_mem_reloc_span_xalloc(vmem_t *vmp, size_t *sizep, size_t align,
553     int vmflag)
554 {
555     ASSERT((vmflag & VM_NORELOC) == 0);
556     return (span_alloc_downsize(vmp, sizep, align, vmflag));
557 }

559 /*
560 * Free a span, which is always exactly one large page.
561 */
562 static void
563 contig_mem_span_free(vmem_t *vmp, void *inaddr, size_t size)
564 {
565     page_t *pp;
566     caddr_t addr = inaddr;
567     caddr_t eaddr;
568     pgcnt_t npages = btopr(size);
569     page_t *rootpp = NULL;

571     ASSERT(size <= contig_mem_import_size_max);
572     /* All slabs should be size aligned */
573     ASSERT(((uintptr_t)addr & (size - 1)) == 0);

575     hat_unload(kas.a_hat, addr, size, HAT_UNLOAD_UNLOCK);

577     for (eaddr = addr + size; addr < eaddr; addr += PAGE_SIZE) {
578         pp = page_find(&kvp, (u_offset_t)(uintptr_t)addr);
579         if (pp == NULL) {
580             panic("contig_mem_span_free: page not found");
581         }
582         if (!page_tryupgrade(pp)) {
583             page_unlock(pp);
584             pp = page_lookup(&kvp,
585                 (u_offset_t)(uintptr_t)addr, SE_EXCL);
586             if (pp == NULL)

```

```

587         panic("contig_mem_span_free: page not found");
588     }

590     ASSERT(PAGE_EXCL(pp));
591     ASSERT(size == page_get_pagesize(pp->p_szc));
592     ASSERT(rootpp == NULL || rootpp->p_szc == pp->p_szc);
593     ASSERT(rootpp == NULL || (page_pptonum(rootpp) +
594         (pgcnt_t)bttop(addr - (caddr_t)inaddr) == page_pptonum(pp)));

596     page_pp_unlock(pp, 0, 1);

598     if (rootpp == NULL)
599         rootpp = pp;
600 }
601 page_destroy_pages(rootpp);
602 page_unresv(npages);

604     if (vmp != NULL)
605         vmem_xfree(vmp, inaddr, size);
606 }

608 static void *
609 contig_vmem_xalloc_aligned_wrapper(vmem_t *vmp, size_t *sizep, size_t align,
610     int vmflag)
611 {
612     ASSERT((align & (align - 1)) == 0);
613     return (vmem_xalloc(vmp, *sizep, align, 0, 0, NULL, NULL, vmflag));
614 }

616 /*
617 * contig_mem_alloc, contig_mem_alloc_align
618 *
619 * Caution: contig_mem_alloc and contig_mem_alloc_align should be
620 * used only when physically contiguous non-relocatable memory is
621 * required. Furthermore, use of these allocation routines should be
622 * minimized as well as should the allocation size. As described in the
623 * contig_mem_arena comment block above, slab allocations fall back to
624 * being outside of the cage. Therefore, overuse of these allocation
625 * routines can lead to non-relocatable large pages being allocated
626 * outside the cage. Such pages prevent the allocation of a larger page
627 * occupying overlapping pages. This can impact performance for
628 * applications that utilize e.g. 256M large pages.
629 */

631 /*
632 * Allocates size aligned contiguous memory up to contig_mem_import_size_max.
633 * Size must be a power of 2.
634 */
635 void *
636 contig_mem_alloc(size_t size)
637 {
638     ASSERT((size & (size - 1)) == 0);
639     return (contig_mem_alloc_align(size, size));
640 }

642 /*
643 * contig_mem_alloc_align allocates real contiguous memory with the
644 * specified alignment up to contig_mem_import_size_max. The alignment must
645 * be a power of 2 and no greater than contig_mem_import_size_max. We assert
646 * the alignment is a power of 2. For non-debug, vmem_xalloc will panic
647 * for non power of 2 alignments.
648 */
649 void *
650 contig_mem_alloc_align(size_t size, size_t align)
651 {
652     void *buf;

```

```

654     ASSERT(size <= contig_mem_import_size_max);
655     ASSERT(align <= contig_mem_import_size_max);
656     ASSERT((align & (align - 1)) == 0);

658     if (align < CONTIG_MEM_ARENA_QUANTUM)
659         align = CONTIG_MEM_ARENA_QUANTUM;

661     /*
662     * We take the lock here to serialize span allocations.
663     * We do not lose concurrency for the common case, since
664     * allocations that don't require new span allocations
665     * are serialized by vmem_xalloc. Serializing span
666     * allocations also prevents us from trying to allocate
667     * more spans than necessary.
668     */
669     mutex_enter(&contig_mem_lock);

671     buf = vmem_xalloc(contig_mem_arena, size, align, 0, 0,
672                     NULL, NULL, VM_NOSLEEP | VM_NORELOC);

674     if ((buf == NULL) && (size <= MMU_PAGESIZE)) {
675         mutex_exit(&contig_mem_lock);
676         return (vmem_xalloc(static_alloc_arena, size, align, 0, 0,
677                             NULL, NULL, VM_NOSLEEP));
678     }

680     if (buf == NULL) {
681         buf = vmem_xalloc(contig_mem_reloc_arena, size, align, 0, 0,
682                             NULL, NULL, VM_NOSLEEP);
683     }

685     mutex_exit(&contig_mem_lock);

687     return (buf);
688 }

690 void
691 contig_mem_free(void *vaddr, size_t size)
692 {
693     if (vmem_contains(contig_mem_arena, vaddr, size)) {
694         vmem_xfree(contig_mem_arena, vaddr, size);
695     } else if (size > MMU_PAGESIZE) {
696         vmem_xfree(contig_mem_reloc_arena, vaddr, size);
697     } else {
698         vmem_xfree(static_alloc_arena, vaddr, size);
699     }
700 }

702 /*
703 * We create a set of stacked vmem arenas to enable us to
704 * allocate large >PAGESIZE chunks of contiguous Real Address space.
705 * The vmem_xcreate interface is used to create the contig_mem_arena
706 * allowing the import routine to downsize the requested slab size
707 * and return a smaller slab.
708 */
709 void
710 contig_mem_init(void)
711 {
712     mutex_init(&contig_mem_lock, NULL, MUTEX_DEFAULT, NULL);

714     contig_mem_slab_arena = vmem_xcreate("contig_mem_slab_arena", NULL, 0,
715                                         CONTIG_MEM_SLAB_ARENA_QUANTUM, contig_vmem_xalloc_aligned_wrapper,
716                                         vmem_xfree, heap_arena, 0, VM_SLEEP | VMC_XALIGN);

718     contig_mem_arena = vmem_xcreate("contig_mem_arena", NULL, 0,

```

```

719     CONTIG_MEM_ARENA_QUANTUM, contig_mem_span_xalloc,
720     contig_mem_span_free, contig_mem_slab_arena, 0,
721     VM_SLEEP | VM_BESTFIT | VMC_XALIGN);

723     contig_mem_reloc_arena = vmem_xcreate("contig_mem_reloc_arena", NULL, 0,
724     CONTIG_MEM_ARENA_QUANTUM, contig_mem_reloc_span_xalloc,
725     contig_mem_span_free, contig_mem_slab_arena, 0,
726     VM_SLEEP | VM_BESTFIT | VMC_XALIGN);

728     if (contig_mem_prealloc_buf == NULL || vmem_add(contig_mem_arena,
729     contig_mem_prealloc_buf, contig_mem_prealloc_size, VM_SLEEP)
730     == NULL) {
731         cmn_err(CE_WARN, "Failed to pre-populate contig_mem_arena");
732     }
733 }

735 /*
736 * In calculating how much memory to pre-allocate, we include a small
737 * amount per-CPU to account for per-CPU buffers in line with measured
738 * values for different size systems. contig_mem_prealloc_base_size is
739 * a cpu specific amount to be pre-allocated before considering per-CPU
740 * requirements and memory size. We always pre-allocate a minimum amount
741 * of memory determined by PREALLOC_MIN. Beyond that, we take the minimum
742 * of contig_mem_prealloc_base_size and a small percentage of physical
743 * memory to prevent allocating too much on smaller systems.
744 * contig_mem_prealloc_base_size is global, allowing for the CPU module
745 * to increase its value if necessary.
746 */
747 #define PREALLOC_PER_CPU      (256 * 1024)          /* 256K */
748 #define PREALLOC_PERCENT     (4)                   /* 4% */
749 #define PREALLOC_MIN         (16 * 1024 * 1024)    /* 16M */
750 size_t contig_mem_prealloc_base_size = 0;

752 /*
753 * Called at boot-time allowing pre-allocation of contiguous memory.
754 * The argument 'alloc_base' is the requested base address for the
755 * allocation and originates in startup_memlist.
756 */
757 caddr_t
758 contig_mem_prealloc(caddr_t alloc_base, pgcnt_t npages)
759 {
760     caddr_t chunkp;

762     contig_mem_prealloc_size = MIN((PREALLOC_PER_CPU * ncpu_guest_max) +
763     contig_mem_prealloc_base_size,
764     (ptob(npages) * PREALLOC_PERCENT) / 100);
765     contig_mem_prealloc_size = MAX(contig_mem_prealloc_size, PREALLOC_MIN);
766     contig_mem_prealloc_size = P2ROUNDUP(contig_mem_prealloc_size,
767     MMU_PAGESIZE4M);

769     alloc_base = (caddr_t)roundup((uintptr_t)alloc_base, MMU_PAGESIZE4M);
770     if (prom_alloc(alloc_base, contig_mem_prealloc_size,
771     MMU_PAGESIZE4M) != alloc_base) {

773         /*
774         * Failed. This may mean the physical memory has holes in it
775         * and it will be more difficult to get large contiguous
776         * pieces of memory. Since we only guarantee contiguous
777         * pieces of memory contig_mem_import_size_max or smaller,
778         * loop, getting contig_mem_import_size_max at a time, until
779         * failure or contig_mem_prealloc_size is reached.
780         */
781         for (chunkp = alloc_base;
782              (chunkp - alloc_base) < contig_mem_prealloc_size;
783              chunkp += contig_mem_import_size_max) {

```

```
785         if (prom_alloc(chunkp, contig_mem_import_size_max,
786             MMU_PAGESIZE4M) != chunkp) {
787             break;
788         }
789     }
790     contig_mem_prealloc_size = chunkp - alloc_base;
791     ASSERT(contig_mem_prealloc_size != 0);
792 }
793
794 if (contig_mem_prealloc_size != 0) {
795     contig_mem_prealloc_buf = alloc_base;
796 } else {
797     contig_mem_prealloc_buf = NULL;
798 }
799 alloc_base += contig_mem_prealloc_size;
800
801 return (alloc_base);
802 }
803
804
805 56 static uint_t sp_color_stride = 16;
806 57 static uint_t sp_color_mask = 0x1f;
807 58 static uint_t sp_current_color = (uint_t)-1;
808
809 60 size_t
810 61 exec_get_spslew(void)
811 62 {
812 63     uint_t spcolor = atomic_inc_32_nv(&sp_current_color);
813 64     return ((size_t)((spcolor & sp_color_mask) * SA(sp_color_stride));
814 65 }
```