

```
*****
57012 Wed Jun 15 19:31:30 2016
new/usr/src/cmd/sgs/elfdump/common/corenote.c
Code review comments from jeffpc
*****
_____unchanged_portion_omitted_

832 /*
833  * Output information from prsecflags_t structure.
834  */
835 static void
836 dump_secflags(note_state_t *state, const char *title)
837 {
838     const sl_prsecflags_layout_t *layout = state->ns_arch->prsecflags;
839     Conv_secflags_buf_t inv;
840     Lword lw;
841 #endif /* ! codereview */
842     Word w;

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

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

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

858         lw = extract_as_lword(state, &layout->pr_inherit);
844         w = extract_as_word(state, &layout->pr_inherit);
859         print_str(state, MSG_ORIG(MSG_CNOTE_T_PR_INHERIT),
860                 conv_prsecflags(lw, 0, &inv));
846         conv_prsecflags(w, 0, &inv);

862         lw = extract_as_lword(state, &layout->pr_lower);
848         w = extract_as_word(state, &layout->pr_lower);
863         print_str(state, MSG_ORIG(MSG_CNOTE_T_PR_LOWER),
864                 conv_prsecflags(lw, 0, &inv));
850         conv_prsecflags(w, 0, &inv);

866         lw = extract_as_lword(state, &layout->pr_upper);
852         w = extract_as_word(state, &layout->pr_upper);
867         print_str(state, MSG_ORIG(MSG_CNOTE_T_PR_UPPER),
868                 conv_prsecflags(lw, 0, &inv));
854         conv_prsecflags(w, 0, &inv);
869     }

871     indent_exit(state);
872 }
_____unchanged_portion_omitted_
```

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

1

12547 Wed Jun 15 19:31:31 2016

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

Code review comments from jeffpc

_____unchanged_portion_omitted_____

```
380 static const sl_prsecflags_layout_t prsecflags_layout = {
381     { 0, 40, 0, 0 }, /* sizeof (prsecflags_t) */
381     { 0, 20, 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 */
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 };
_____unchanged_portion_omitted_____
```

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

1

12501 Wed Jun 15 19:31:32 2016

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

Code review comments from jeffpc

_____unchanged_portion_omitted_____

```
380 static const sl_prsecflags_layout_t prsecflags_layout = {
381     { 0, 40, 0, 0 }, /* sizeof (prsecflags_t) */
381     { 0, 20, 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 */
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 };
_____unchanged_portion_omitted_____
```

```
*****
```

```
87999 Wed Jun 15 19:31:33 2016
```

```
new/usr/src/cmd/sgs/libconv/common/corenote.c
```

```
Code review comments from jeffpc
```

```
*****
```

```
_____unchanged_portion_omitted_____
```

```
2588 #define PROCSECFLSZ   CONV_EXPN_FIELD_DEF_PREFIX_SIZE +      \
2589     MSG_ASLR_SIZE     + CONV_EXPN_FIELD_DEF_SEP_SIZE +      \
2590     MSG_FORBIDNULLMAP_SIZE + CONV_EXPN_FIELD_DEF_SEP_SIZE + \
2591     MSG_NOEXECSTACK_SIZE + CONV_EXPN_FIELD_DEF_SEP_SIZE +   \
2592     CONV_INV_BUFSIZE  + CONV_EXPN_FIELD_DEF_SUFFIX_SIZE

2594 /*
2595  * Ensure that Conv_cnote_pr_secflags_buf_t is large enough:
2596  *
2597  * PROCSECFLSZ is the real minimum size of the buffer required by
2598  * conv_prsecflags(). However, Conv_cnote_pr_secflags_buf_t uses
2599  * CONV_CNOTE_PSECFLAGS_FLAG_BUFSIZE to set the buffer size. We do things this
2600  * way because the definition of PROCSECFLSZ uses information that is not
2601  * available in the environment of other programs that include the conv.h
2602  * header file.
2603  */
2604 #if (CONV_PRSECFLAGS_BUFSIZE != PROCSECFLSZ) && !defined(__lint)
2605 #define REPORT_BUFSIZE PROCSECFLSZ
2606 #include "report_bufsize.h"
2607 #error "CONV_PRSECFLAGS_BUFSIZE does not match PROCSECFLSZ"
2608 #endif

2610 const char *
2611 conv_prsecflags(secflagset_t flags, Conv_fmt_flags_t fmt_flags,
2612               Conv_secflags_buf_t *secflags_buf)
2613 {
2614     /*
2615      * The values are initialized later, based on position in this array
2616      */
2617     static Val_desc vda[] = {
2618         { 0, MSG_ASLR },
2619         { 0, MSG_FORBIDNULLMAP },
2620         { 0, MSG_NOEXECSTACK },
2621         { 0, 0 }
2622     };
2623     static CONV_EXPN_FIELD_ARG conv_arg = {
2624         NULL, sizeof (secflags_buf->buf)
2625     };
2626     int i;

2628     for (i = 0; vda[i].v_msg != 0; i++)
2629         vda[i].v_val = secflag_to_bit(i);

2631     if (flags == 0)
2632         return (MSG_ORIG(MSG_GBL_ZERO));

2634     conv_arg.buf = secflags_buf->buf;
2635     conv_arg.oflags = conv_arg.rflags = flags;
2636     (void) conv_expn_field(&conv_arg, vda, fmt_flags);

2638     return ((const char *)secflags_buf->buf);
2639 }
_____unchanged_portion_omitted_____
```

4812 Wed Jun 15 19:31:34 2016

new/usr/src/common/secflags/secflags.c

Code review comments from jeffpc

unchanged_portion_omitted_

```
210 void
211 secflags_to_str(secflagset_t flags, char *buf, size_t buflen)
212 {
213     struct flagdesc *fd;
214
215     if (buflen >= 1)
216         buf[0] = '\0';
217
218     if (flags == 0) {
219         (void) strcpy(buf, "none", buflen);
220         return;
221     }
222
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);
226                 (void) strcat(buf, " ", buflen);
227                 (void) strcat(buf, fd->name, buflen);
228         }
229
230         secflag_clear(&flags, fd->value);
231     }
232
233     if (flags != 0) {
234         /* unknown flags */
234         char hexbuf[19]; /* 0x%16 PRIx64 */
234         char hexbuf[11]; /* 0x%08x */
235
236         (void) snprintf(hexbuf, sizeof (hexbuf), "0x%16" PRIx64, flags);
236         (void) snprintf(hexbuf, sizeof (hexbuf), "0x%08x", flags);
237         if (buf[0] != '\0')
238             (void) strcat(buf, ",", buflen);
238             (void) strcat(buf, " ", buflen);
239             (void) strcat(buf, hexbuf, buflen);
240     }
241 }
```

unchanged_portion_omitted_

```

*****
3512 Wed Jun 15 19:31:35 2016
new/usr/src/lib/libc/inc/priv_private.h
Code review comments from jeffpc
*****
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 2007 Sun Microsystems, Inc. All rights reserved.
24  * Use is subject to license terms.
25  */

27 #ifndef _PRIV_PRIVATE_H
28 #define _PRIV_PRIVATE_H

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

32 #endif /* ! codereview */
33 #include <sys/types.h>
34 #include <sys/priv.h>
35 #include <limits.h>

37 /*
38  * Libc private privilege data.
39  */

41 #ifdef __cplusplus
42 extern "C" {
43 #endif

45 #define LOADPRIVDATA(d)      d = __priv_getdata()
46 #define GETPRIVDATA()      __priv_getdata()
47 #define LOCKPRIVDATA()     { \
48                             /* Data already allocated */ \
49                             (void) lock_data(); \
50                             (void) refresh_data(); \
51                             }
52 #define UNLOCKPRIVDATA()    unlock_data()
53 #define WITHPRIVLOCKED(t, b, x) { \
54                             t __result; \
55                             if (lock_data() != 0) \
56                                 return (b); \
57                             __result = (x); \
58                             if (__result == (b) && refresh_data()) \
59                                 __result = (x); \
60                             unlock_data(); \
61                             return (__result); \

```

```

62     }

64 /*
65  * Privilege mask macros.
66  */
67 #define __NBWRD      (CHAR_BIT * sizeof (priv_chunk_t))
68 #define privmask(n)  (1 << (((__NBWRD - 1) - ((n) % __NBWRD)))
69 #define privword(n)  ((n)/__NBWRD)

71 /*
72  * Same as the functions, but for numeric privileges.
73  */
74 #define PRIV_ADDSET(a, p)      (((priv_chunk_t *) (a))[privword(p)] |= \
75                                 privmask(p))
76 #define PRIV_DELSET(a, p)     (((priv_chunk_t *) (a))[privword(p)] &= \
77                                 ~privmask(p))
78 #define PRIV_ISMEMBER(a, p)   (((priv_chunk_t *) (a))[privword(p)] & \
79                                 privmask(p) != 0)

81 /*
82  * The structure is static except for the setsort, privnames and nprivs
83  * field. The pinfo structure initially has sufficient room and the kernel
84  * guarantees no offset changes so we can copy a new structure on top of it.
85  * The locking strategy is this: we lock it when we need to reference any
86  * of the volatile fields.
87  */
88 typedef struct priv_data {
89     size_t      pd_setsize;          /* In bytes */
90     int         pd_nsets, pd_nprivs;
91     uint32_t    pd_ucredsize;
92     char        **pd_setnames;
93     char        **pd_privnames;
94     int         *pd_setsort;
95     priv_impl_info_t *pd_pinfo;
96     priv_set_t  *pd_basicset;
97     priv_set_t  *pd_zoneset;
98 } priv_data_t;

100 extern priv_data_t *__priv_getdata(void);
101 extern priv_data_t *__priv_parse_info(priv_impl_info_t *);
102 extern void __priv_free_info(priv_data_t *);
103 extern priv_data_t *privdata;

105 extern int lock_data(void);
106 extern boolean_t refresh_data(void);
107 extern void unlock_data(void);

109 extern boolean_t __priv_isemptyset(priv_data_t *, const priv_set_t *);
110 extern boolean_t __priv_isfullset(priv_data_t *, const priv_set_t *);
111 extern boolean_t __priv_issubset(priv_data_t *, const priv_set_t *,
112                                 const priv_set_t *);
113 extern const char *__priv_getbynum(const priv_data_t *, int);

115 extern int getprivinfo(priv_impl_info_t *, size_t);

117 extern priv_set_t *priv_basic(void);

119 #ifdef __cplusplus
120 }
121 #endif

123 #endif /* _PRIV_PRIVATE_H */

```

```

*****
4697 Wed Jun 15 19:31:35 2016
new/usr/src/lib/libc/port/sys/sbrk.c
Code review comments from jeffpc
*****
_____unchanged_portion_omitted_____

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     if (!primary_link_map) {
128         errno = ENOTSUP;
129         return ((void *)-1);
130     }
131     if ((low_align & (low_align - 1)) != 0 ||
132         (high_align & (high_align - 1)) != 0) {
133         errno = EINVAL;
134         return ((void *)-1);
135     }
136     low_align = MAX(low_align, ALIGNSZ);
137     high_align = MAX(high_align, ALIGNSZ);

139     lmutex_lock(&__sbrk_lock);

141     if (_nd == NULL)
142     if (_nd == NULL) {
143         _nd = (void *)_brk_unlocked(0);
144     }

144     old_brk = (uintptr_t)BRKALIGN(_nd);
145     ret_brk = P2ROUNDUP(old_brk, low_align);
146     high_brk = ret_brk + min_size;
147     new_brk = P2ROUNDUP(high_brk, high_align);

149     /*
150     * Check for overflow
151     */
152     if (ret_brk < old_brk || high_brk < ret_brk || new_brk < high_brk) {
153         lmutex_unlock(&__sbrk_lock);
154         errno = ENOMEM;
155         return ((void *)-1);
156     }

158     if ((brk_result = _brk_unlocked((void *)new_brk)) == 0)
159     if ((brk_result = (int)_brk_unlocked((void *)new_brk)) == 0)
159         _nd = (void *)new_brk;
160     lmutex_unlock(&__sbrk_lock);

162     if (brk_result != 0)

```

```

163         return ((void *)-1);

165     if (actual_size != NULL)
166         *actual_size = (new_brk - ret_brk);
167     return ((void *)ret_brk);
168 }

170 int
171 brk(void *new_brk)
172 {
173     intptr_t result;
174     int result;

175     /*
176     * brk(2) will return the current brk if given an argument of 0, so we
177     * need to fail it here
178     */
179     if (new_brk == 0) {
180         errno = ENOMEM;
181         return (-1);
182     }

184     if (!primary_link_map) {
185         errno = ENOTSUP;
186         return (-1);
187     }
188     /*
189     * Need to align this here; _brk_unlocked won't do it for us.
190     */
191     new_brk = BRKALIGN(new_brk);

193     lmutex_lock(&__sbrk_lock);
194     if ((result = _brk_unlocked(new_brk)) == 0)
195     if ((result = (int)_brk_unlocked(new_brk)) == 0)
195         _nd = new_brk;
196     lmutex_unlock(&__sbrk_lock);

198     return (result);
199 }
_____unchanged_portion_omitted_____

```

new/usr/src/lib/libscf/common/highlevel.c

1

```
*****
10652 Wed Jun 15 19:31:36 2016
new/usr/src/lib/libscf/common/highlevel.c
Code review comments from jeffpc
*****
_____unchanged_portion_omitted_____

370 int
371 scf_default_secflags(scf_handle_t *hndl, psecflags_t *flags)
372 {
373 #if !defined(NATIVE_BUILD)
374     scf_property_t *prop;
375     scf_value_t *val;
376     const char *flagname;
377     int flag;
378     struct group_desc *g;
379     struct group_desc groups[] = {
380         {NULL, "svc:/system/process-security/"
381          " :properties/default"},
382         {NULL, "svc:/system/process-security/"
383          " :properties/lower"},
384         {NULL, "svc:/system/process-security/"
385          " :properties/upper"},
386         {NULL, NULL}
387     };

389     groups[0].set = &flags->psf_inherit;
390     groups[1].set = &flags->psf_lower;
391     groups[2].set = &flags->psf_upper;

393     /* Ensure sane defaults */
394     psecflags_default(flags);

396     for (g = groups; g->set != NULL; g++) {
397         for (flag = 0; (flagname = secflag_to_str(flag)) != NULL;
398              flag++) {
399             char *pfmri;
400             uint8_t flagval = 0;

402             if ((val = scf_value_create(hndl)) == NULL)
403                 return (-1);

405             if ((prop = scf_property_create(hndl)) == NULL) {
406                 scf_value_destroy(val);
407                 return (-1);
408             }

410             if ((pfmri = uu_msprintf("%s/%s", g->fmri,
411                                     flagname)) == NULL)
412                 uu_die("Allocation failure\n");

414             if (scf_handle_decode_fmri(hndl, pfmri,
415                                       NULL, NULL, NULL, NULL, prop, NULL) != 0)
416                 goto next;

418             if (scf_property_get_value(prop, val) != 0)
419                 goto next;

421             (void) scf_value_get_boolean(val, &flagval);

423             if (flagval != 0)
424                 secflag_set(g->set, flag);
425             else
426                 secflag_clear(g->set, flag);

428 next:
```

new/usr/src/lib/libscf/common/highlevel.c

2

```
429         uu_free(pfmri);
430         scf_value_destroy(val);
431         scf_property_destroy(prop);
432     }
433 }

435     if (!psecflags_validate(flags))
436         return (-1);

438     return (0);
439 #else
440     assert(0);
441     abort();
442 #endif /* !codereview */
443 #endif /* !NATIVE_BUILD */
438     return (0);
444 }
_____unchanged_portion_omitted_____
```



```

*****
59567 Wed Jun 15 19:31:37 2016
new/usr/src/man/man1/ld.1
Code review comments from jeffpc
*****
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] [\fB-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-C\fR \fIname\fR] [\fB-C\fR] [\fB-d\fR
17 [\fB-D\fR \fIToken\fR,...] [\fB-e\fR \fIepsym\fR] [\fB-f\fR \fIname\fR | \fB-F\f
18 [\fB-i\fR] [\fB-I\fR \fIname\fR] [\fB-l\fR \fIx\fR] [\fB-L\fR \fIpath\fR] [\fB-m
19 [\fB-N\fR \fIstring\fR] [\fB-o\fR \fIoutfile\fR] [\fB-p\fR \fIauditlib\fR] [\fB-
20 [\fB-Q\fR y | n] [\fB-R\fR \fIpath\fR] [\fB-s\fR] [\fB-S\fR \fIsupportlib\fR] [\
21 [\fB-u\fR \fIsymname\fR] [\fB-V\fR] [\fB-Y P\fR \fI,dirlist\fR] [\fB-z\fR absexec
22 [\fB-z\fR allextact | defaultextract | weakextract ] [\fB-z\fR altexec64]
23 [\fB-z\fR aslr[=\fIstate\fR]] [\fB-z\fR assert-deflib] [ \fB-z\fR assert-deflib=
24 [\fB-z\fR combrelloc | nocombrelloc ] [\fB-z\fR defs | nodefs]
25 [\fB-z\fR direct | nodirect] [\fB-z\fR endfiltee]
26 [\fB-z\fR fatal-warnings | nofatal-warnings ] [\fB-z\fR finiarray=\fIfunction\fR
27 [\fB-z\fR globalaudit] [\fB-z\fR groupperm | nogroupperm]
28 [\fB-z\fR guidance[=\fIIid1\fR,\fIIid2\fR...]] [\fB-z\fR help ]
29 [\fB-z\fR ignore | record] [\fB-z\fR initarray=\fIfunction\fR] [\fB-z\fR initfir
30 [\fB-z\fR interpose] [\fB-z\fR lazyload | nolazyload]
31 [\fB-z\fR ld32=\fIarg1\fR,\fIarg2\fR,...] [\fB-z\fR ld64=\fIarg1\fR,\fIarg2\fR,..
32 [\fB-z\fR loadfltr] [\fB-z\fR muldefs] [\fB-z\fR nocompstrtab] [\fB-z\fR nodefau
33 [\fB-z\fR nodelete] [\fB-z\fR nodlopen] [\fB-z\fR nodump] [\fB-z\fR noldynsym]
34 [\fB-z\fR nopartial] [\fB-z\fR noversion] [\fB-z\fR now] [\fB-z\fR origin]
35 [\fB-z\fR preinitarray=\fIfunction\fR] [\fB-z\fR redlocsym] [\fB-z\fR relaxreloc
36 [\fB-z\fR rescan-now] [\fB-z\fR rescan] [\fB-z\fR rescan-start \fI\&...\fR \fB-z\
37 [\fB-z\fR target=sparc|x86] [\fB-z\fR text | textwarn | textoff]
38 [\fB-z\fR verbose] [\fB-z\fR wrap=\fIsymbol\fR] \fIfilename\fR...
39 .fi

41 .SH DESCRIPTION
42 .LP
43 The link-editor, \fBld\fR, combines relocatable object files by resolving
44 symbol references to symbol definitions, together with performing relocations.
45 \fBld\fR operates in two modes, static or dynamic, as governed by the \fB-d\fR
46 option. In all cases, the output of \fBld\fR is left in the file \fBld.out\fR by
47 default. See NOTES.
48 .sp
49 .LP
50 In dynamic mode, \fB-dy\fR, the default, relocatable object files that are
51 provided as arguments are combined to produce an executable object file. This
52 file is linked at execution with any shared object files that are provided as
53 arguments. If the \fB-G\fR option is specified, relocatable object files are
54 combined to produce a shared object. Without the \fB-G\fR option, a dynamic
55 executable is created.
56 .sp
57 .LP
58 In static mode, \fB-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 \fBStatic 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 \fILinker and Libraries Guide\fR.
69 .sp
70 .LP
71 If any argument is a library, \fBld\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 \fBar.h\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, \fIShared Objects\fR in \fILinker and Libraries Guide\fR
84 .sp
85 .LP
86 For an archive library, \fBld\fR loads only those routines that define an
87 unresolved external reference. \fBld\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 \fB-z\fR \fBrescan\fR options. See \fIArchive Processing\fR in \fILinker and
95 Libraries Guide\fR.
96 .sp
97 .LP
98 \fBld\fR is a cross link-editor, able to link 32-bit objects or 64-bit objects,
99 for Sparc or x86 targets. \fBld\fR uses the \fBELF\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 \fB-32\fR, \fB-64\fR and \fB-z target\fR options, and the \fBLD_NOEXEC_64\fR
104 environment variable.
105 .SS "Static Executables"
106 .LP
107 The creation of static executables has been discouraged for many releases. In
108 fact, 64-bit system archive libraries have never been provided. Because a
109 static executable is built against system archive libraries, the executable
110 contains system implementation details. This self-containment has a number of
111 drawbacks.
112 .RS +4
113 .TP
114 .ie t \(\bu
115 .el o
116 The executable is immune to the benefits of system patches delivered as shared
117 objects. The executable therefore, must be rebuilt to take advantage of many
118 system improvements.
119 .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 .LP
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 \fBdirect\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 \fBdefs\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 \fBinterpose\fR
223 option. The use of \fB-B\fR \fBdirect\fR enables \fB-z\fR \fBlazyload\fR for
224 all dependencies.
225 .sp
226 The \fB-B\fR \fBnodirect\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 \fBdirect\fR option. See Appendix D, \fIDirect Bindings\fR in \fILinker and
230 Libraries Guide\fR.
231 .RE

233 .sp
234 .ne 2
235 .na
236 \fB-B\fR | \fBdynamic\fR | \fBstatic\fR
237 .ad
238 .sp .6
239 .RS 4n
240 Options governing library inclusion. \fB-B\fR \fBdynamic\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-B\fR \fBstatic\fR option is given, no shared
243 objects are accepted until \fB-B\fR \fBdynamic\fR is seen. See the \fB-l\fR
244 option.
245 .RE

247 .sp
248 .ne 2
249 .na
250 \fB-B\fR | \fBeliminate\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 `\fILinker and Libraries Guide\fR`.
 263 .RE

265 .sp
 266 .ne 2
 267 .na
 268 `\fB\fB-B\fR \fBgroup\fR\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\fB-B\fR \fBlocal\fR\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 `\fILinker and Libraries Guide\fR`.
 298 .RE

300 .sp
 301 .ne 2
 302 .na
 303 `\fB\fB-B\fR \fBreduce\fR\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\fB-B\fR \fBsymbolic\fR\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\fB-c\fR \fIname\fR\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\fB-C\fR\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\fB-d\fR \fBy\fR | \fBn\fR\fR`
 365 .ad
 366 .sp .6
 367 .RS 4n
 368 When `\fB-d\fR \fBy\fR`, the default, is specified, `\fBld\fR` uses dynamic
 369 linking. When `\fB-d\fR \fBn\fR` is specified, `\fBld\fR` uses static linking. See
 370 `\fBStatic Executables\fR` under `DESCRIPTION`, and `\fB-B\fR`
 371 `\fBdynamic\fR` or `\fBstatic\fR`.
 372 .RE

374 .sp
 375 .ne 2
 376 .na
 377 `\fB\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 \fBexec\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 \fBstatic\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-L\fR \fIpath\fR
 532 .ad
 533 .br
 534 .na
 535 \fB--library-path\fR \fIpath\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 \fIDirectories
 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-m\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-M\fR \fImapfile\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-N\fR \fIstring\fR
 580 .ad
 581 .sp .6
 582 .RS 4n
 583 This option causes a \fBBDT_NEEDED\fR entry to be added to the \fB&.dynamic\fR
 584 section of the object being built. The value of the \fBBDT_NEEDED\fR string is
 585 the \fIstring\fR that is specified on the command line. This option is position
 586 dependent, and the \fBBDT_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-o\fR \fIoutfile\fR
 596 .ad
 597 .br
 598 .na
 599 \fB--output\fR \fIoutfile\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-p\fR \fIauditlib\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-P\fR \fIauditlib\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 \fBglobaudit\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-Q\fR \fIby\fR | \fBn\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\fR
673 .ad
674 .br
675 .na
676 \fB\fB-rpath\fR \fIpath\fR\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 \fIDirectories 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 Libraries 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\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 \fB$SYMTAB\fR and its associated string table section are not
715 created in the output object file. The elimination of a \fB$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 \fBfBredlocsym\fR
718 and \fB-z\fR \fBfBnoldynsym\fR options.
719 .RE

```

```

721 .sp
722 .ne 2
723 .na
724 \fB\fB-S\fR \fIsupportlib\fR\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\fR option can also be supplied using the \fB$SGS_SUPPORT\fR
731 environment variable. See \fIILink-Editor Support Interface\fR in \fIILinker and
732 Libraries Guide\fR.
733 .RE

735 .sp
736 .ne 2
737 .na
738 \fB\fB-t\fR\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\fR \fIisymname\fR\fR
750 .ad
751 .br
752 .na
753 \fB\fB--undefined\fR \fIisymname\fR\fR
754 .ad
755 .sp .6
756 .RS 4n
757 Enters \fIisymname\fR 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\fR in \fIILinker and Libraries Guide\fR.
763 .RE

765 .sp
766 .ne 2
767 .na
768 \fB\fB-V\fR\fR
769 .ad
770 .br
771 .na
772 \fB\fB--version\fR\fR
773 .ad
774 .sp .6
775 .RS 4n
776 Outputs a message giving information about the version of \fBld\fR being used.
777 .RE

779 .sp
780 .ne 2
781 .na
782 \fB\fB-Y\fR \fIbp\fR \fIidirlist\fR\fR
783 .ad
784 .sp .6
785 .RS 4n
786 Changes the default directories used for finding libraries. \fIidirlist\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\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 \fB\fB-z\fR \fBcombrelloc\fR | \fBnoccombrelloc\fR\fR
863 .ad
864 .sp .6
865 .RS 4n
866 By default, \fBld\fR combines multiple relocation sections when building
867 executables or shared objects. This section combination differs from
868 relocatable objects, in which relocation sections are maintained in a
869 one-to-one relationship with the sections to which the relocations must be
870 applied. The \fB-z\fR \fBnoccombrelloc\fR option disables this merging of
871 relocation sections, and preserves the one-to-one relationship found in the
872 original relocatable objects.
873 .sp
874 \fBld\fR sorts the entries of data relocation sections by their symbol
875 reference. This sorting reduces runtime symbol lookup. When multiple relocation
876 sections are combined, this sorting produces the least possible relocation
877 overhead when objects are loaded into memory, and speeds the runtime loading of
878 dynamic objects.
879 .sp
880 Historically, the individual relocation sections were carried over to any
881 executable or shared object, and the \fB-z\fR \fBcombrelloc\fR option was
882 required to enable the relocation section merging previously described.
883 Relocation section merging is now the default. The \fB-z\fR \fBcombrelloc\fR
884 option is still accepted for the benefit of old build environments, but the
885 option is unnecessary, and has no effect.
886 .RE

888 .sp
889 .ne 2
890 .na
891 \fB\fB-z\fR \fBassert-deflib\fR\fR
892 .ad
893 .br
894 .na
895 \fB\fB-z\fR \fBassert-deflib=\fR\fIlibname\fR\fR
896 .ad
897 .sp .6
898 .RS 4n
899 Enables warnings that check the location of where libraries passed in with
900 \fB-l\fR are found. If the link-editor finds a library on its default search
901 path it will emit a warning. This warning can be made fatal in conjunction with
902 the option \fB-z fatal-warnings\fR. Passing \fIlibname\fR white lists a library
903 from this check. The library must be the full name of the library, e.g.
904 \fIlibc.so\fR. To white list multiple libraries, the \fB-z
905 assert-deflib=\fR\fIlibname\fR option can be repeated multiple times. This
906 option is useful when trying to build self-contained objects where a referenced
907 library might exist in the default system library path and in alternate paths
908 specified by \fB-L\fR, but you only want the alternate paths to be used.
909 .RE

911 .sp
912 .ne 2
913 .na
914 \fB\fB-z\fR \fBdefs\fR | \fBnodefs\fR\fR
915 .ad
916 .br
917 .na
918 \fB\fB--no-undefined\fR\fR

```

```

919 .ad
920 .sp .6
921 .RS 4n
922 The \fB-z\fR \fBdefs\fR option and the \fB--no-undefined\fR option force a
923 fatal error if any undefined symbols remain at the end of the link. This mode
924 is the default when an executable is built. For historic reasons, this mode is
925 \fBnot\fR the default when building a shared object. Use of the \fB-z\fR
926 \fBdefs\fR option is recommended, as this mode assures the object being built
927 is self-contained. A self-contained object has all symbolic references resolved
928 internally, or to the object's immediate dependencies.
929 .sp
930 The \fB-z\fR \fBnodefs\fR option allows undefined symbols. For historic
931 reasons, this mode is the default when a shared object is built. When used with
932 executables, the behavior of references to such undefined symbols is
933 unspecified. Use of the \fB-z\fR \fBnodefs\fR option is not recommended.
934 .RE

936 .sp
937 .ne 2
938 .na
939 \fB\fB-z\fR \fBdirect\fR | \fBnodirect\fR\fR
940 .ad
941 .sp .6
942 .RS 4n
943 Enables or disables direct binding to any dependencies that follow on the
944 command line. These options allow finer control over direct binding than the
945 global counterpart \fB-B\fR \fBdirect\fR. The \fB-z\fR \fBdirect\fR option also
946 differs from the \fB-B\fR \fBdirect\fR option in the following areas. Direct
947 binding information is not established between a symbol reference and an
948 associated definition within the object being created. Lazy loading is not
949 enabled.
950 .RE

952 .sp
953 .ne 2
954 .na
955 \fB\fB-z\fR \fBendfiltee\fR\fR
956 .ad
957 .sp .6
958 .RS 4n
959 Marks a filtee so that when processed by a filter, the filtee terminates any
960 further filtee searches by the filter. See \fIReducing Filtee Searches\fR in
961 \fILinker and Libraries Guide\fR.
962 .RE

964 .sp
965 .ne 2
966 .na
967 \fB\fB-z\fR \fBfatal-warnings\fR | \fBnofatal-warnings\fR\fR
968 .ad
969 .br
970 .na
971 \fB\fB--fatal-warnings\fR | \fB--no-fatal-warnings\fR
972 .ad
973 .sp .6
974 .RS 4n
975 Controls the behavior of warnings emitted from the link-editor. Setting \fB-z
976 fatal-warnings\fR promotes warnings emitted by the link-editor to fatal errors
977 that will cause the link-editor to fail before linking. \fB-z
978 nofatal-warnings\fR instead demotes these warnings such that they will not cause
979 the link-editor to exit prematurely.
980 .RE

983 .sp
984 .ne 2

```

```

985 .na
986 \fB\fB-z\fR \fBfiniarray=\fR\fIfunction\fR\fR
987 .ad
988 .sp .6
989 .RS 4n
990 Appends an entry to the \fB&.finiarray\fR section of the object being built.
991 If no \fB&.finiarray\fR section is present, a section is created. The new
992 entry is initialized to point to \fIfunction\fR. See \fIInitialization and
993 Termination Sections\fR in \fILinker and Libraries Guide\fR.
994 .RE

996 .sp
997 .ne 2
998 .na
999 \fB\fB-z\fR \fBglobalaudit\fR\fR
1000 .ad
1001 .sp .6
1002 .RS 4n
1003 This option supplements an audit library definition that has been recorded with
1004 the \fB-P\fR option. This option is only meaningful when building a dynamic
1005 executable. Audit libraries that are defined within an object with the \fB-P\fR
1006 option typically allow for the auditing of the immediate dependencies of the
1007 object. The \fB-z\fR \fBglobalaudit\fR promotes the auditor to a global
1008 auditor, thus allowing the auditing of all dependencies. See \fIInvoking the
1009 Auditing Interface\fR in \fILinker and Libraries Guide\fR.
1010 .sp
1011 An auditor established with the \fB-P\fR option and the \fB-z\fR
1012 \fBglobalaudit\fR option, is equivalent to the auditor being established with
1013 the \fBBLD_AUDIT\fR environment variable. See \fBld.so.1\fR(1).
1014 .RE

1016 .sp
1017 .ne 2
1018 .na
1019 \fB\fB-z\fR \fBgroupperm\fR | \fBnogroupperm\fR\fR
1020 .ad
1021 .sp .6
1022 .RS 4n
1023 Assigns, or deassigns each dependency that follows to a unique group. The
1024 assignment of a dependency to a group has the same effect as if the dependency
1025 had been built using the \fB-B\fR \fBgroup\fR option.
1026 .RE

1028 .sp
1029 .ne 2
1030 .na
1031 \fB\fB-z\fR \fBguidance\fR[=\fIid1\fR,\fIid2\fR... ]
1032 .ad
1033 .sp .6
1034 .RS 4n
1035 Give messages suggesting link-editor features that could improve the resulting
1036 dynamic object.
1037 .LP
1038 Specific classes of suggestion can be silenced by specifying an optional comma s
1039 list of guidance identifiers.
1040 .LP
1041 The current classes of suggestion provided are:

1043 .sp
1044 .ne 2
1045 .na
1046 Enable use of direct binding
1047 .ad
1048 .sp .6
1049 .RS 4n
1050 Suggests that \fB-z direct\fR or \fB-B direct\fR be present prior to any

```


1051 specified dependency. This allows predictable symbol binding at runtime.

1053 Can be disabled with `\fB-z guidance=nodirect\fR`
 1054 .RE

1056 .sp
 1057 .ne 2
 1058 .na

1059 Enable lazy dependency loading
 1060 .ad
 1061 .sp .6
 1062 .RS 4n

1063 Suggests that `\fB-z lazyload\fR` be present prior to any specified dependency.
 1064 This allows the dynamic object to be loaded more quickly.

1066 Can be disabled with `\fB-z guidance=nolazyload\fR`.
 1067 .RE

1069 .sp
 1070 .ne 2
 1071 .na

1072 Shared objects should define all their dependencies.
 1073 .ad
 1074 .sp .6
 1075 .RS 4n

1076 Suggests that `\fB-z defs\fR` be specified on the link-editor command line.
 1077 Shared objects that explicitly state all their dependencies behave more
 1078 predictably when used.

1080 Can be disabled with `\fB-z guidance=nodefs\fR`
 1081 .RE

1083 .sp
 1084 .ne 2
 1085 .na

1086 Version 2 mapfile syntax
 1087 .ad
 1088 .sp .6
 1089 .RS 4n

1090 Suggests that any specified mapfiles use the more readable version 2 syntax.

1092 Can be disabled with `\fB-z guidance=nomapfile\fR`.
 1093 .RE

1095 .sp
 1096 .ne 2
 1097 .na

1098 Read-only text segment
 1099 .ad
 1100 .sp .6
 1101 .RS 4n

1102 Should any runtime relocations within the text segment exist, suggests that
 1103 the object be compiled with position independent code (PIC). Keeping large
 1104 allocatable sections read-only allows them to be shared between processes
 1105 using a given shared object.

1107 Can be disabled with `\fB-z guidance=notext\fR`
 1108 .RE

1110 .sp
 1111 .ne 2
 1112 .na

1113 No unused dependencies
 1114 .ad
 1115 .sp .6
 1116 .RS 4n

1117 Suggests that any dependency not referenced by the resulting dynamic object be
 1118 removed from the link-editor command line.

1120 Can be disabled with `\fB-z guidance=nounused\fR`.
 1121 .RE
 1122 .RE

1124 .sp
 1125 .ne 2
 1126 .na

1127 `\fB\fB-z\fR \fBhelp\fR\fR`
 1128 .ad
 1129 .br
 1130 .na

1131 `\fB\fB--help\fR\fR`
 1132 .ad
 1133 .sp .6
 1134 .RS 4n

1135 Print a summary of the command line options on the standard output and exit.
 1136 .RE

1138 .sp
 1139 .ne 2
 1140 .na

1141 `\fB\fB-z\fR \fBignore\fR | \fBrecord\fR\fR`
 1142 .ad
 1143 .sp .6
 1144 .RS 4n

1145 Ignores, or records, dynamic dependencies that are not referenced as part of
 1146 the link-edit. Ignores, or records, unreferenced `\fBELF\fR` sections from the
 1147 relocatable objects that are read as part of the link-edit. By default,
 1148 `\fB-z\fR \fBrecord\fR` is in effect.
 1149 .sp

1150 If an `\fBELF\fR` section is ignored, the section is eliminated from the output
 1151 file being generated. A section is ignored when three conditions are true. The
 1152 eliminated section must contribute to an allocatable segment. The eliminated
 1153 section must provide no global symbols. No other section from any object that
 1154 contributes to the link-edit, must reference an eliminated section.
 1155 .RE

1157 .sp
 1158 .ne 2
 1159 .na

1160 `\fB\fB-z\fR \fBinitarray=\fR\fIfunction\fR\fR`
 1161 .ad
 1162 .sp .6
 1163 .RS 4n

1164 Appends an entry to the `\fB\&.initarray\fR` section of the object being built.
 1165 If no `\fB\&.initarray\fR` section is present, a section is created. The new
 1166 entry is initialized to point to `\fIfunction\fR`. See `\fIInitialization and`
 1167 `Termination Sections\fR` in `\fILinker and Libraries Guide\fR`.
 1168 .RE

1170 .sp
 1171 .ne 2
 1172 .na

1173 `\fB\fB-z\fR \fBinitfirst\fR\fR`
 1174 .ad
 1175 .sp .6
 1176 .RS 4n

1177 Marks the object so that its runtime initialization occurs before the runtime
 1178 initialization of any other objects brought into the process at the same time.
 1179 In addition, the object runtime finalization occurs after the runtime
 1180 finalization of any other objects removed from the process at the same time.
 1181 This option is only meaningful when building a shared object.
 1182 .RE

```

1184 .sp
1185 .ne 2
1186 .na
1187 \fB\fB-z\fR \fBinterpose\fR\fR
1188 .ad
1189 .sp .6
1190 .RS 4n
1191 Marks the object as an interposer. At runtime, an object is identified as an
1192 explicit interposer if the object has been tagged using the \fB-z interpose\fR
1193 option. An explicit interposer is also established when an object is loaded
1194 using the \fBLD_PRELOAD\fR environment variable. Implicit interposition can
1195 occur because of the load order of objects, however, this implicit
1196 interposition is unknown to the runtime linker. Explicit interposition can
1197 ensure that interposition takes place regardless of the order in which objects
1198 are loaded. Explicit interposition also ensures that the runtime linker
1199 searches for symbols in any explicit interposers when direct bindings are in
1200 effect.
1201 .RE

1203 .sp
1204 .ne 2
1205 .na
1206 \fB\fB-z\fR \fBlazyload\fR | \fBnolazyload\fR\fR
1207 .ad
1208 .sp .6
1209 .RS 4n
1210 Enables or disables the marking of dynamic dependencies to be lazily loaded.
1211 Dynamic dependencies which are marked \fBlazyload\fR are not loaded at initial
1212 process start-up. These dependencies are delayed until the first binding to the
1213 object is made. \fBNote:\fR Lazy loading requires the correct declaration of
1214 dependencies, together with associated runpaths for each dynamic object used
1215 within a process. See \fILazy Loading of Dynamic Dependencies\fR in \fILinker
1216 and Libraries Guide\fR.
1217 .RE

1219 .sp
1220 .ne 2
1221 .na
1222 \fB\fB-z\fR \fBld32\fR=\fIarg1\fR,\fIarg2\fR,...\fR
1223 .ad
1224 .br
1225 .na
1226 \fB\fB-z\fR \fBld64\fR=\fIarg1\fR,\fIarg2\fR,...\fR
1227 .ad
1228 .sp .6
1229 .RS 4n
1230 The class of the link-editor is affected by the class of the output file being
1231 created and by the capabilities of the underlying operating system. The
1232 \fB-z\fR \fBld\fR[\fB32\fR|\fB64\fR] options provide a means of defining any
1233 link-editor argument. The defined argument is only interpreted, respectively,
1234 by the 32-bit class or 64-bit class of the link-editor.
1235 .sp
1236 For example, support libraries are class specific, so the correct class of
1237 support library can be ensured using:
1238 .sp
1239 .in +2
1240 .nf
1241 \fBld ... -z ld32=-Saudit32.so.1 -z ld64=-Saudit64.so.1 ... \fR
1242 .fi
1243 .in -2
1244 .sp

1246 The class of link-editor that is invoked is determined from the \fBELF\fR class
1247 of the first relocatable file that is seen on the command line. This
1248 determination is carried out \fBprior\fR to any \fB-z\fR

```

```

1249 \fBld\fR[\fB32\fR|\fB64\fR] processing.
1250 .RE

1252 .sp
1253 .ne 2
1254 .na
1255 \fB\fB-z\fR \fBloadfilter\fR\fR
1256 .ad
1257 .sp .6
1258 .RS 4n
1259 Marks a filter to indicate that filterees must be processed immediately at
1260 runtime. Normally, filter processing is delayed until a symbol reference is
1261 bound to the filter. The runtime processing of an object that contains this
1262 flag mimics that which occurs if the \fBLD_LOADFLTR\fR environment variable is
1263 in effect. See the \fBld.so.1\fR(1).
1264 .RE

1266 .sp
1267 .ne 2
1268 .na
1269 \fB\fB-z\fR \fBmuldefs\fR\fR
1270 .ad
1271 .br
1272 .na
1273 \fB\fB--allow-multiple-definition\fR\fR
1274 .ad
1275 .sp .6
1276 .RS 4n
1277 Allows multiple symbol definitions. By default, multiple symbol definitions
1278 that occur between relocatable objects result in a fatal error condition. This
1279 option, suppresses the error condition, allowing the first symbol definition to
1280 be taken.
1281 .RE

1283 .sp
1284 .ne 2
1285 .na
1286 \fB\fB-z\fR \fBnocompstrtab\fR\fR
1287 .ad
1288 .sp .6
1289 .RS 4n
1290 Disables the compression of \fBELF\fR string tables. By default, string
1291 compression is applied to \fBSHT_STRTAB\fR sections, and to \fBSHT_PROGBITS\fR
1292 sections that have their \fBSHF_MERGE\fR and \fBSHF_STRINGS\fR section flags
1293 set.
1294 .RE

1296 .sp
1297 .ne 2
1298 .na
1299 \fB\fB-z\fR \fBnodefaultlib\fR\fR
1300 .ad
1301 .sp .6
1302 .RS 4n
1303 Marks the object so that the runtime default library search path, used after
1304 any \fBLD_LIBRARY_PATH\fR or runpaths, is ignored. This option implies that all
1305 dependencies of the object can be satisfied from its runpath.
1306 .RE

1308 .sp
1309 .ne 2
1310 .na
1311 \fB\fB-z\fR \fBnodelete\fR\fR
1312 .ad
1313 .sp .6
1314 .RS 4n

```

1315 Marks the object as non-deletable at runtime. This mode is similar to adding
 1316 the object to the process by using `\fBdlopen\fR(3C)` with the
 1317 `\fBRTLD_NODELETE\fR` mode.
 1318 .RE

1320 .sp
 1321 .ne 2
 1322 .na
 1323 `\fB\fB-z\fR \fBnodlopen\fR\fR`
 1324 .ad
 1325 .sp .6
 1326 .RS 4n
 1327 Marks the object as not available to `\fBdlopen\fR(3C)`, either as the object
 1328 specified by the `\fBdlopen()\fR`, or as any form of dependency required by the
 1329 object specified by the `\fBdlopen()\fR`. This option is only meaningful when
 1330 building a shared object.
 1331 .RE

1333 .sp
 1334 .ne 2
 1335 .na
 1336 `\fB\fB-z\fR \fBnodump\fR\fR`
 1337 .ad
 1338 .sp .6
 1339 .RS 4n
 1340 Marks the object as not available to `\fBldump\fR(3C)`.
 1341 .RE

1343 .sp
 1344 .ne 2
 1345 .na
 1346 `\fB\fB-z\fR \fBnoldynsym\fR\fR`
 1347 .ad
 1348 .sp .6
 1349 .RS 4n
 1350 Prevents the inclusion of a `\fB\fB.SUNW_ldynsym\fR` section in dynamic
 1351 executables or sharable libraries. The `\fB\fB.SUNW_ldynsym\fR` section augments
 1352 the `\fB\fB.dynsym\fR` section by providing symbols for local functions. Local
 1353 function symbols allow debuggers to display local function names in stack
 1354 traces from stripped programs. Similarly, `\fBdldaddr\fR(3C)` is able to supply
 1355 more accurate results.
 1356 .sp
 1357 The `\fB-z\fR \fBnoldynsym\fR` option also prevents the inclusion of the two
 1358 symbol sort sections that are related to the `\fB\fB.SUNW_ldynsym\fR` section. The
 1359 `\fB\fB.SUNW_dynsymsort\fR` section provides sorted access to regular function and
 1360 variable symbols. The `\fB\fB.SUNW_dyntlssort\fR` section provides sorted access
 1361 to thread local storage (`\fBFTLS\fR`) variable symbols.
 1362 .sp
 1363 The `\fB\fB.SUNW_ldynsym\fR`, `\fB\fB.SUNW_dynsymsort\fR`, and
 1364 `\fB\fB.SUNW_dyntlssort\fR` sections, which becomes part of the allocable text
 1365 segment of the resulting file, cannot be removed by `\fBstrip\fR(1)`. Therefore,
 1366 the `\fB-z\fR \fBnoldynsym\fR` option is the only way to prevent their inclusion.
 1367 See the `\fB-s\fR` and `\fB-z\fR \fBbredlocs\fR` options.
 1368 .RE

1370 .sp
 1371 .ne 2
 1372 .na
 1373 `\fB\fB-z\fR \fBnopartial\fR\fR`
 1374 .ad
 1375 .sp .6
 1376 .RS 4n
 1377 Partially initialized symbols, that are defined within relocatable object
 1378 files, are expanded in the output file being generated.
 1379 .RE

1381 .sp
 1382 .ne 2
 1383 .na
 1384 `\fB\fB-z\fR \fBnoverversion\fR\fR`
 1385 .ad
 1386 .sp .6
 1387 .RS 4n
 1388 Does not record any versioning sections. Any version sections or associated
 1389 `\fB\fB.&.dynamic\fR` section entries are not generated in the output image.
 1390 .RE

1392 .sp
 1393 .ne 2
 1394 .na
 1395 `\fB\fB-z\fR \fBnow\fR\fR`
 1396 .ad
 1397 .sp .6
 1398 .RS 4n
 1399 Marks the object as requiring non-lazy runtime binding. This mode is similar to
 1400 adding the object to the process by using `\fBdlopen\fR(3C)` with the
 1401 `\fBRTLD_NOW\fR` mode. This mode is also similar to having the `\fBLD_BIND_NOW\fR`
 1402 environment variable in effect. See `\fBld.so.1\fR(1)`.
 1403 .RE

1405 .sp
 1406 .ne 2
 1407 .na
 1408 `\fB\fB-z\fR \fBorigin\fR\fR`
 1409 .ad
 1410 .sp .6
 1411 .RS 4n
 1412 Marks the object as requiring immediate `\fB$ORIGIN\fR` processing at runtime.
 1413 This option is only maintained for historic compatibility, as the runtime
 1414 analysis of objects to provide for `\fB$ORIGIN\fR` processing is now default.
 1415 .RE

1417 .sp
 1418 .ne 2
 1419 .na
 1420 `\fB\fB-z\fR \fBpreinitarray=\fR\fIfunction\fR\fR`
 1421 .ad
 1422 .sp .6
 1423 .RS 4n
 1424 Appends an entry to the `\fB\fB.preinitarray\fR` section of the object being
 1425 built. If no `\fB\fB.preinitarray\fR` section is present, a section is created.
 1426 The new entry is initialized to point to `\fIfunction\fR`. See `\fIInitialization`
 1427 and `Termination Sections\fR` in `\fILinker and Libraries Guide\fR`.
 1428 .RE

1430 .sp
 1431 .ne 2
 1432 .na
 1433 `\fB\fB-z\fR \fBbredlocs\fR\fR`
 1434 .ad
 1435 .sp .6
 1436 .RS 4n
 1437 Eliminates all local symbols except for the `\fIsect\fR` symbols from the symbol
 1438 table `\fBBSHT_SYMTAB\fR`. All relocations that refer to local symbols are updated
 1439 to refer to the corresponding `\fIsect\fR` symbol. This option allows specialized
 1440 objects to greatly reduce their symbol table sizes. Eliminated local symbols
 1441 can reduce the `\fB\fB.&.stab*\fR` debugging information that is generated using the
 1442 compiler drivers `\fB-g\fR` option. See the `\fB-s\fR` and `\fB-z\fR \fBnoldynsym\fR`
 1443 options.
 1444 .RE

1446 .sp

```

1447 .ne 2
1448 .na
1449 \fB\fB-z\fR \fBrelaxreloc\fR\fR
1450 .ad
1451 .sp .6
1452 .RS 4n
1453 \fBld\fR normally issues a fatal error upon encountering a relocation using a
1454 symbol that references an eliminated COMDAT section. If \fB-z\fR
1455 \fBrelaxreloc\fR is enabled, \fBld\fR instead redirects such relocations to the
1456 equivalent symbol in the COMDAT section that was kept. \fB-z\fR
1457 \fBrelaxreloc\fR is a specialized option, mainly of interest to compiler
1458 authors, and is not intended for general use.
1459 .RE

1461 .sp
1462 .ne 2
1463 .na
1464 \fB\fB-z\fR \fBrescan-now\fR\fR
1465 .ad
1466 .br
1467 .na
1468 \fB\fB-z\fR \fBrescan\fR\fR
1469 .ad
1470 .sp .6
1471 .RS 4n
1472 These options rescan the archive files that are provided to the link-edit. By
1473 default, archives are processed once as the archives appear on the command
1474 line. Archives are traditionally specified at the end of the command line so
1475 that their symbol definitions resolve any preceding references. However,
1476 specifying archives multiple times to satisfy their own interdependencies can
1477 be necessary.
1478 .sp
1479 \fB-z\fR \fBrescan-now\fR is a positional option, and is processed by the
1480 link-editor immediately when encountered on the command line. All archives seen
1481 on the command line up to that point are immediately reprocessed in an attempt
1482 to locate additional archive members that resolve symbol references. This
1483 archive rescanning is repeated until a pass over the archives occurs in which
1484 no new members are extracted.
1485 .sp
1486 \fB-z\fR \fBrescan\fR is a position independent option. The link-editor defers
1487 the rescan operation until after it has processed the entire command line, and
1488 then initiates a final rescan operation over all archives seen on the command
1489 line. The \fB-z\fR \fBrescan\fR operation can interact incorrectly
1490 with objects that contain initialization (.init) or finalization (.fini)
1491 sections, preventing the code in those sections from running. For this reason,
1492 \fB-z\fR \fBrescan\fR is deprecated, and use of \fB-z\fR \fBrescan-now\fR is
1493 advised.
1494 .RE

1496 .sp
1497 .ne 2
1498 .na
1499 \fB\fB-z\fR \fBrescan-start\fR ... \fB-z\fR \fBrescan-end\fR\fR
1500 .ad
1501 .br
1502 .na
1503 \fB\fB--start-group\fR ... \fB--end-group\fR\fR
1504 .ad
1505 .br
1506 .na
1507 \fB\fB-(\fR ... \fB-)\fR\fR
1508 .ad
1509 .sp .6
1510 .RS 4n
1511 Defines an archive rescan group. This is a positional construct, and is
1512 processed by the link-editor immediately upon encountering the closing

```

```

1513 delimiter option. Archives found within the group delimiter options are
1514 reprocessed as a group in an attempt to locate additional archive members that
1515 resolve symbol references. This archive rescanning is repeated until a pass
1516 over the archives occurs in which no new members are extracted.
1517 Archive rescan groups cannot be nested.
1518 .RE

1520 .sp
1521 .ne 2
1522 .na
1523 \fB\fB-z\fR \fBtarget=sparc|x86\fR \fI\fR\fR
1524 .ad
1525 .sp .6
1526 .RS 4n
1527 Specifies the machine type for the output object. Supported targets are Sparc
1528 and x86. The 32-bit machine type for the specified target is used unless the
1529 \fB-64\fR option is also present, in which case the corresponding 64-bit
1530 machine type is used. By default, the machine type of the object being
1531 generated is determined from the first \fBELF\fR object processed from the
1532 command line. If no objects are specified, the machine type is determined by
1533 the first object encountered within the first archive processed from the
1534 command line. If there are no objects or archives, the link-editor assumes the
1535 native machine. This option is useful when creating an object directly with
1536 \fBld\fR whose input is solely from a \fBmapfile\fR. See the \fB-M\fR option.
1537 It can also be useful in the rare case of linking entirely from an archive that
1538 contains objects of different machine types for which the first object is not
1539 of the desired machine type. See \fIThe 32-bit link-editor and 64-bit
1540 link-editor\fR in \fILinker and Libraries Guide\fR.
1541 .RE

1543 .sp
1544 .ne 2
1545 .na
1546 \fB\fB-z\fR \fBtext\fR\fR
1547 .ad
1548 .sp .6
1549 .RS 4n
1550 In dynamic mode only, forces a fatal error if any relocations against
1551 non-writable, allocatable sections remain. For historic reasons, this mode is
1552 not the default when building an executable or shared object. However, its use
1553 is recommended to ensure that the text segment of the dynamic object being
1554 built is shareable between multiple running processes. A shared text segment
1555 incurs the least relocation overhead when loaded into memory. See
1556 \fIPosition-Independent Code\fR in \fILinker and Libraries Guide\fR.
1557 .RE

1559 .sp
1560 .ne 2
1561 .na
1562 \fB\fB-z\fR \fBtextoff\fR\fR
1563 .ad
1564 .sp .6
1565 .RS 4n
1566 In dynamic mode only, allows relocations against all allocatable sections,
1567 including non-writable ones. This mode is the default when building a shared
1568 object.
1569 .RE

1571 .sp
1572 .ne 2
1573 .na
1574 \fB\fB-z\fR \fBtextwarn\fR\fR
1575 .ad
1576 .sp .6
1577 .RS 4n
1578 In dynamic mode only, lists a warning if any relocations against non-writable,

```

```
1579 allocatable sections remain. This mode is the default when building an
1580 executable.
1581 .RE

1583 .sp
1584 .ne 2
1585 .na
1586 \fB\fB-z\fR \fBverbose\fR\fR
1587 .ad
1588 .sp .6
1589 .RS 4n
1590 This option provides additional warning diagnostics during a link-edit.
1591 Presently, this option conveys suspicious use of displacement relocations. This
1592 option also conveys the restricted use of static \fBTLR\fR relocations when
1593 building shared objects. In future, this option might be enhanced to provide
1594 additional diagnostics that are deemed too noisy to be generated by default.
1595 .RE

1597 .sp
1598 .ne 2
1599 .na
1600 \fB\fB-z\fR\fBwrap=\fR \fIsymbol\fR\fR
1601 .ad
1602 .br
1603 .na
1604 \fB\fB-wrap=\fR \fIsymbol\fR\fR
1605 .ad
1606 .br
1607 .na
1608 \fB\fB--wrap=\fR \fIsymbol\fR\fR
1609 .ad
1610 .sp .6
1611 .RS 4n
1612 Rename undefined references to \fIsymbol\fR in order to allow wrapper code to
1613 be linked into the output object without having to modify source code. When
1614 \fB-z wrap\fR is specified, all undefined references to \fIsymbol\fR are
1615 modified to reference \fB__wrap_\fR \fIsymbol\fR, and all references to
1616 \fB__real_\fR \fIsymbol\fR are modified to reference \fIsymbol\fR. The user is
1617 expected to provide an object containing the \fB__wrap_\fR \fIsymbol\fR
1618 function. This wrapper function can call \fB__real_\fR \fIsymbol\fR in order to
1619 reference the actual function being wrapped.
1620 .sp
1621 The following is an example of a wrapper for the \fBmalloc\fR(3C) function:
1622 .sp
1623 .in +2
1624 .nf
1625 void *
1626 __wrap_malloc(size_t c)
1627 {
1628     (void) printf("malloc called with %zu\n", c);
1629     return (__real_malloc(c));
1630 }
unchanged portion omitted
```

```

*****
5791 Wed Jun 15 19:31:39 2016
new/usr/src/man/man1/psecflags.1
Code review comments from jeffpc
*****
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"
13.TH "PSECFLAGS" "1" "May 3, 2014"
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]...
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 ...
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 \fI-command\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-l\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 -l\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      U:      aslr,forbidnullmap,noexecstack
226 .fi
227 .in -2
228 .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      U:      aslr,forbidnullmap,noexecstack
243 .fi
244 .in -2
245 .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)
```

44021 Wed Jun 15 19:31:40 2016

new/usr/src/man/man1m/zonecfg.1m

Code review comments from jeffpc

```

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
10 zonecfg \- set up zone configuration
11 .SH SYNOPSIS
12 .LP
13 \fBzonecfg\fR \fB-z\fR \fIzonename\fR
14 .fi
15
16 .LP
17 .nf
18 \fBzonecfg\fR \fB-z\fR \fIzonename\fR \fIsubcommand\fR
19 .fi
20
21 .LP
22 .nf
23 \fBzonecfg\fR \fB-z\fR \fIzonename\fR \fB-f\fR \fIcommand_file\fR
24 .fi
25
26 .LP
27 .nf
28 \fBzonecfg\fR help
29 .fi
30
31 .SH DESCRIPTION
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
49
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 .LP
75 The following resource types are supported:
76 .sp
77 .ne 2
78 .na
79 \fB\battr\fR
80 .ad
81 .sp .6
82 .RS 4n
83 Generic attribute.
84 .RE
85
86 .sp
87 .ne 2
88 .na
89 \fB\bapped-cpu\fR
90 .ad
91 .sp .6
92 .RS 4n
93 Limits for CPU usage.
94 .RE
95
96 .sp
97 .ne 2
98 .na
99 \fB\bapped-memory\fR
100 .ad
101 .sp .6
102 .RS 4n
103 Limits for physical, swap, and locked memory.
104 .RE
105
106 .sp
107 .ne 2
108 .na
109 \fB\bdataset\fR
110 .ad
111 .sp .6
112 .RS 4n
113 \fBZFS\fR dataset.
114 .RE
115
116 .sp
117 .ne 2
118 .na
119 \fB\bdedicated-cpu\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
125
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 .SS "Properties"
177 .LP
178 Each resource type has one or more properties. There are also some global
179 properties, that is, properties of the configuration as a whole, rather than of
180 some particular resource.
181 .sp
182 .LP
183 The following properties are supported:
184 .sp
185 .ne 2
186 .na
187 \fB(global)\fR
188 .ad
189 .sp .6
190 .RS 4n
191 \fBzonenname\fR
192 .RE

```

```

194 .sp
195 .ne 2
196 .na
197 \fB(global)\fR
198 .ad
199 .sp .6
200 .RS 4n
201 \fBzonepath\fR
202 .RE

204 .sp
205 .ne 2
206 .na
207 \fB(global)\fR
208 .ad
209 .sp .6
210 .RS 4n
211 \fBautoboot\fR
212 .RE

214 .sp
215 .ne 2
216 .na
217 \fB(global)\fR
218 .ad
219 .sp .6
220 .RS 4n
221 \fBbootargs\fR
222 .RE

224 .sp
225 .ne 2
226 .na
227 \fB(global)\fR
228 .ad
229 .sp .6
230 .RS 4n
231 \fBpool\fR
232 .RE

234 .sp
235 .ne 2
236 .na
237 \fB(global)\fR
238 .ad
239 .sp .6
240 .RS 4n
241 \fBlimitpriv\fR
242 .RE

244 .sp
245 .ne 2
246 .na
247 \fB(global)\fR
248 .ad
249 .sp .6
250 .RS 4n
251 \fBbrand\fR
252 .RE

254 .sp
255 .ne 2
256 .na
257 \fB(global)\fR
258 .ad

```

```

259 .sp .6
260 .RS 4n
261 \fBcpu-shares\fR
262 .RE

264 .sp
265 .ne 2
266 .na
267 \fB(global)\fR
268 .ad
269 .sp .6
270 .RS 4n
271 \fBhostid\fR
272 .RE

274 .sp
275 .ne 2
276 .na
277 \fB(global)\fR
278 .ad
279 .sp .6
280 .RS 4n
281 \fBmax-lwps\fR
282 .RE

284 .sp
285 .ne 2
286 .na
287 \fB(global)\fR
288 .ad
289 .sp .6
290 .RS 4n
291 \fBmax-msg-ids\fR
292 .RE

294 .sp
295 .ne 2
296 .na
297 \fB(global)\fR
298 .ad
299 .sp .6
300 .RS 4n
301 \fBmax-sem-ids\fR
302 .RE

304 .sp
305 .ne 2
306 .na
307 \fB(global)\fR
308 .ad
309 .sp .6
310 .RS 4n
311 \fBmax-shm-ids\fR
312 .RE

314 .sp
315 .ne 2
316 .na
317 \fB(global)\fR
318 .ad
319 .sp .6
320 .RS 4n
321 \fBmax-shm-memory\fR
322 .RE

324 .sp

```

```

325 .ne 2
326 .na
327 \fB(global)\fR
328 .ad
329 .sp .6
330 .RS 4n
331 \fBscheduling-class\fR
332 .RE

334 .sp
335 .ne 2
336 .na
337 .B (global)
338 .ad
339 .sp .6
340 .RS 4n
341 .B fs-allowed
342 .RE

344 .sp
345 .ne 2
346 .na
347 \fB\fBs\fR\fR
348 .ad
349 .sp .6
350 .RS 4n
351 \fBdir\fR, \fBspecial\fR, \fBraw\fR, \fBtype\fR, \fBoptions\fR
352 .RE

354 .sp
355 .ne 2
356 .na
357 \fB\fBnet\fR\fR
358 .ad
359 .sp .6
360 .RS 4n
361 \fBaddress\fR, \fBphysical\fR, \fBdefrouter\fR
362 .RE

364 .sp
365 .ne 2
366 .na
367 \fB\fBdevice\fR\fR
368 .ad
369 .sp .6
370 .RS 4n
371 \fBmatch\fR
372 .RE

374 .sp
375 .ne 2
376 .na
377 \fB\fBrcctl\fR\fR
378 .ad
379 .sp .6
380 .RS 4n
381 \fBname\fR, \fBvalue\fR
382 .RE

384 .sp
385 .ne 2
386 .na
387 \fB\fBattr\fR\fR
388 .ad
389 .sp .6
390 .RS 4n

```

```

391 \fBname\fR, \fBtype\fR, \fBvalue\fR
392 .RE

394 .sp
395 .ne 2
396 .na
397 \fB\fBdataset\fR\fR
398 .ad
399 .sp .6
400 .RS 4n
401 \fBname\fR
402 .RE

404 .sp
405 .ne 2
406 .na
407 \fB\fBdedicated-cpu\fR\fR
408 .ad
409 .sp .6
410 .RS 4n
411 \fBncpus\fR, \fBimportance\fR
412 .RE

414 .sp
415 .ne 2
416 .na
417 \fB\fBcapped-memory\fR\fR
418 .ad
419 .sp .6
420 .RS 4n
421 \fBphysical\fR, \fBswap\fR, \fBblocked\fR
422 .RE

424 .sp
425 .ne 2
426 .na
427 \fB\fBcapped-cpu\fR\fR
428 .ad
429 .sp .6
430 .RS 4n
431 \fBncpus\fR
432 .RE

434 .sp
435 .ne 2
436 .na
437 \fB\fBsecurity-flags\fB\fB
438 .ad
439 .sp .6
440 .RS 4n
441 \fBlower\fR, \fBdefault\fR, \fBupper\fR.
442 .RE

444 .sp
445 .LP
446 As for the property values which are paired with these names, they are either
447 simple, complex, or lists. The type allowed is property-specific. Simple values
448 are strings, optionally enclosed within quotation marks. Complex values have
449 the syntax:
450 .sp
451 .in +2
452 .nf
453 (<\fIname\fR>=<\fIvalue\fR>,<\fIname\fR>=<\fIvalue\fR>,...)
454 .fi
455 .in -2
456 .sp

```

```

458 .sp
459 .LP
460 where each <\fIvalue\fR> is simple, and the <\fIname\fR> strings are unique
461 within a given property. Lists have the syntax:
462 .sp
463 .in +2
464 .nf
465 [<\fIvalue\fR>,...]
466 .fi
467 .in -2
468 .sp

470 .sp
471 .LP
472 where each <\fIvalue\fR> is either simple or complex. A list of a single value
473 (either simple or complex) is equivalent to specifying that value without the
474 list syntax. That is, "foo" is equivalent to "[foo]". A list can be empty
475 (denoted by "").
476 .sp
477 .LP
478 In interpreting property values, \fBzonecfg\fR accepts regular expressions as
479 specified in \fBfnmatch\fR(5). See \fBEXAMPLES\fR.
480 .sp
481 .LP
482 The property types are described as follows:
483 .sp
484 .ne 2
485 .na
486 \fBglobal: \fBzonename\fR\fR
487 .ad
488 .sp .6
489 .RS 4n
490 The name of the zone.
491 .RE

493 .sp
494 .ne 2
495 .na
496 \fBglobal: \fBzonepath\fR\fR
497 .ad
498 .sp .6
499 .RS 4n
500 Path to zone's file system.
501 .RE

503 .sp
504 .ne 2
505 .na
506 \fBglobal: \fBautoboot\fR\fR
507 .ad
508 .sp .6
509 .RS 4n
510 Boolean indicating that a zone should be booted automatically at system boot.
511 Note that if the zones service is disabled, the zone will not autoboot,
512 regardless of the setting of this property. You enable the zones service with a
513 \fBsvcadm\fR command, such as:
514 .sp
515 .in +2
516 .nf
517 # \fBsvcadm enable svc:/system/zones:default\fR
518 .fi
519 .in -2
520 .sp

522 Replace \fBenable\fR with \fBdisable\fR to disable the zones service. See

```

```

523 \fBsvcadm\fR(1M).
524 .RE

526 .sp
527 .ne 2
528 .na
529 \fBglobal: \fBbootargs\fR\fR
530 .ad
531 .sp .6
532 .RS 4n
533 Arguments (options) to be passed to the zone bootup, unless options are
534 supplied to the "\fBzoneadm boot\fR" command, in which case those take
535 precedence. The valid arguments are described in \fBzoneadm\fR(1M).
536 .RE

538 .sp
539 .ne 2
540 .na
541 \fBglobal: \fBpool\fR\fR
542 .ad
543 .sp .6
544 .RS 4n
545 Name of the resource pool that this zone must be bound to when booted. This
546 property is incompatible with the \fBdedicated-cpu\fR resource.
547 .RE

549 .sp
550 .ne 2
551 .na
552 \fBglobal: \fBlimitpriv\fR\fR
553 .ad
554 .sp .6
555 .RS 4n
556 The maximum set of privileges any process in this zone can obtain. The property
557 should consist of a comma-separated privilege set specification as described in
558 \fBpriv_str_to_set\fR(3C). Privileges can be excluded from the resulting set by
559 preceding their names with a dash (-) or an exclamation point (!). The special
560 privilege string "zone" is not supported in this context. If the special string
561 "default" occurs as the first token in the property, it expands into a safe set
562 of privileges that preserve the resource and security isolation described in
563 \fBzones\fR(5). A missing or empty property is equivalent to this same set of
564 safe privileges.
565 .sp
566 The system administrator must take extreme care when configuring privileges for
567 a zone. Some privileges cannot be excluded through this mechanism as they are
568 required in order to boot a zone. In addition, there are certain privileges
569 which cannot be given to a zone as doing so would allow processes inside a zone
570 to unduly affect processes in other zones. \fBzoneadm\fR(1M) indicates when an
571 invalid privilege has been added or removed from a zone's privilege set when an
572 attempt is made to either "boot" or "ready" the zone.
573 .sp
574 See \fBprivileges\fR(5) for a description of privileges. The command "\fBppriv
575 -l\fR" (see \fBppriv\fR(1)) produces a list of all Solaris privileges. You can
576 specify privileges as they are displayed by \fBppriv\fR. In
577 \fBprivileges\fR(5), privileges are listed in the form
578 PRIV_\fIprivilege_name\fR. For example, the privilege \fIisys_time\fR, as you
579 would specify it in this property, is listed in \fBprivileges\fR(5) as
580 \fBPRIV_SYS_TIME\fR.
581 .RE

583 .sp
584 .ne 2
585 .na
586 \fBglobal: \fBbrand\fR\fR
587 .ad
588 .sp .6

```

```

589 .RS 4n
590 The zone's brand type.
591 .RE

593 .sp
594 .ne 2
595 .na
596 \fBglobal: \fBip-type\fR\fR
597 .ad
598 .sp .6
599 .RS 4n
600 A zone can either share the IP instance with the global zone, which is the
601 default, or have its own exclusive instance of IP.
602 .sp
603 This property takes the values \fBshared\fR and \fBexclusive\fR.
604 .RE

606 .sp
607 .ne 2
608 .na
609 \fBglobal: \fBhostid\fR\fR
610 .ad
611 .sp .6
612 .RS 4n
613 A zone can emulate a 32-bit host identifier to ease system consolidation. A
614 zone's \fBhostid\fR property is empty by default, meaning that the zone does
615 not emulate a host identifier. Zone host identifiers must be hexadecimal values
616 between 0 and FFFFFFFE. A \fB0x\fR or \fBX\fR prefix is optional. Both
617 uppercase and lowercase hexadecimal digits are acceptable.
618 .RE

620 .sp
621 .ne 2
622 .na
623 \fBfbfs\fR: dir, special, raw, type, options\fR
624 .ad
625 .sp .6
626 .RS 4n
627 Values needed to determine how, where, and so forth to mount file systems. See
628 \fBmount\fR(1M), \fBmount(2), \fBfsck\fR(1M), and \fBfstab\fR(4).
629 .RE

631 .sp
632 .ne 2
633 .na
634 \fBfbnet\fR: address, physical, defrouter\fR
635 .ad
636 .sp .6
637 .RS 4n
638 The network address and physical interface name of the network interface. The
639 network address is one of:
640 .RS +4
641 .TP
642 .ie t \((bu
643 .el o
644 a valid IPv4 address, optionally followed by "\fB/\fR" and a prefix length;
645 .RE
646 .RS +4
647 .TP
648 .ie t \((bu
649 .el o
650 a valid IPv6 address, which must be followed by "\fB/\fR" and a prefix length;
651 .RE
652 .RS +4
653 .TP
654 .ie t \((bu

```

```

655 .el o
656 a host name which resolves to an IPv4 address.
657 .RE
658 Note that host names that resolve to IPv6 addresses are not supported.
659 .sp
660 The physical interface name is the network interface name.
661 .sp
662 The default router is specified similarly to the network address except that it
663 must not be followed by a \fB/\fR (slash) and a network prefix length.
664 .sp
665 A zone can be configured to be either exclusive-IP or shared-IP. For a
666 shared-IP zone, you must set both the physical and address properties; setting
667 the default router is optional. The interface specified in the physical
668 property must be plumbed in the global zone prior to booting the non-global
669 zone. However, if the interface is not used by the global zone, it should be
670 configured \fBdown\fR in the global zone, and the default router for the
671 interface should be specified here.
672 .sp
673 For an exclusive-IP zone, the physical property must be set and the address and
674 default router properties cannot be set.
675 .RE

677 .sp
678 .ne 2
679 .na
680 \fB\fBdevice\fR: match\fR
681 .ad
682 .sp .6
683 .RS 4n
684 Device name to match.
685 .RE

687 .sp
688 .ne 2
689 .na
690 \fB\fBrcctl\fR: name, value\fR
691 .ad
692 .sp .6
693 .RS 4n
694 The name and \fBpriv\fR/\fBlimit\fR/\fBaction\fR triple of a resource control.
695 See \fBprctl\fR(1) and \fBrcctladm\fR(1M). The preferred way to set rctl values
696 is to use the global property name associated with a specific rctl.
697 .RE

699 .sp
700 .ne 2
701 .na
702 \fB\fBattr\fR: name, type, value\fR
703 .ad
704 .sp .6
705 .RS 4n
706 The name, type and value of a generic attribute. The \fBtype\fR must be one of
707 \fBint\fR, \fBuint\fR, \fBboolean\fR or \fBstring\fR, and the value must be of
708 that type. \fBuint\fR means unsigned , that is, a non-negative integer.
709 .RE

711 .sp
712 .ne 2
713 .na
714 \fB\fBdataset\fR: name\fR
715 .ad
716 .sp .6
717 .RS 4n
718 The name of a \fBZFS\fR dataset to be accessed from within the zone. See
719 \fBzfs\fR(1M).
720 .RE

```

```

722 .sp
723 .ne 2
724 .na
725 \fB\fBglobal: \fBcpu-shares\fR\fR
726 .ad
727 .sp .6
728 .RS 4n
729 The number of Fair Share Scheduler (FSS) shares to allocate to this zone. This
730 property is incompatible with the \fBdedicated-cpu\fR resource. This property
731 is the preferred way to set the \fBzone.cpu-shares\fR rctl.
732 .RE

734 .sp
735 .ne 2
736 .na
737 \fB\fBglobal: \fBmax-lwps\fR\fR
738 .ad
739 .sp .6
740 .RS 4n
741 The maximum number of LWPs simultaneously available to this zone. This property
742 is the preferred way to set the \fBzone.max-lwps\fR rctl.
743 .RE

745 .sp
746 .ne 2
747 .na
748 \fB\fBglobal: \fBmax-msg-ids\fR\fR
749 .ad
750 .sp .6
751 .RS 4n
752 The maximum number of message queue IDs allowed for this zone. This property is
753 the preferred way to set the \fBzone.max-msg-ids\fR rctl.
754 .RE

756 .sp
757 .ne 2
758 .na
759 \fB\fBglobal: \fBmax-sem-ids\fR\fR
760 .ad
761 .sp .6
762 .RS 4n
763 The maximum number of semaphore IDs allowed for this zone. This property is the
764 preferred way to set the \fBzone.max-sem-ids\fR rctl.
765 .RE

767 .sp
768 .ne 2
769 .na
770 \fB\fBglobal: \fBmax-shm-ids\fR\fR
771 .ad
772 .sp .6
773 .RS 4n
774 The maximum number of shared memory IDs allowed for this zone. This property is
775 the preferred way to set the \fBzone.max-shm-ids\fR rctl.
776 .RE

778 .sp
779 .ne 2
780 .na
781 \fB\fBglobal: \fBmax-shm-memory\fR\fR
782 .ad
783 .sp .6
784 .RS 4n
785 The maximum amount of shared memory allowed for this zone. This property is the
786 preferred way to set the \fBzone.max-shm-memory\fR rctl. A scale (K, M, G, T)

```

787 can be applied to the value for this number (for example, 1M is one megabyte).
788 .RE

790 .sp
791 .ne 2
792 .na
793 \fBglobal: \fBscheduling-class\fR\fR
794 .ad
795 .sp .6
796 .RS 4n
797 Specifies the scheduling class used for processes running in a zone. When this
798 property is not specified, the scheduling class is established as follows:
799 .RS +4
800 .TP
801 .ie t \(\bu
802 .el o
803 If the \fBcpu-shares\fR property or equivalent rctl is set, the scheduling
804 class FSS is used.
805 .RE
806 .RS +4
807 .TP
808 .ie t \(\bu
809 .el o
810 If neither \fBcpu-shares\fR nor the equivalent rctl is set and the zone's pool
811 property references a pool that has a default scheduling class, that class is
812 used.
813 .RE
814 .RS +4
815 .TP
816 .ie t \(\bu
817 .el o
818 Under any other conditions, the system default scheduling class is used.
819 .RE
820 .RE

824 .sp
825 .ne 2
826 .na
827 \fB\fBdedicated-cpu\fR: ncpus, importance\fR
828 .ad
829 .sp .6
830 .RS 4n
831 The number of CPUs that should be assigned for this zone's exclusive use. The
832 zone will create a pool and processor set when it boots. See \fBpooladm\fR(1M)
833 and \fBpoolcfg\fR(1M) for more information on resource pools. The \fBncpu\fR
834 property can specify a single value or a range (for example, 1-4) of
835 processors. The \fBimportance\fR property is optional; if set, it will specify
836 the \fBpset.importance\fR value for use by \fBpoold\fR(1M). If this resource is
837 used, there must be enough free processors to allocate to this zone when it
838 boots or the zone will not boot. The processors assigned to this zone will not
839 be available for the use of the global zone or other zones. This resource is
840 incompatible with both the \fBpool\fR and \fBcpu-shares\fR properties. Only a
841 single instance of this resource can be added to the zone.
842 .RE

844 .sp
845 .ne 2
846 .na
847 \fB\fBcapped-memory\fR: physical, swap, locked\fR
848 .ad
849 .sp .6
850 .RS 4n
851 The caps on the memory that can be used by this zone. A scale (K, M, G, T) can
852 be applied to the value for each of these numbers (for example, 1M is one

853 megabyte). Each of these properties is optional but at least one property must
854 be set when adding this resource. Only a single instance of this resource can
855 be added to the zone. The \fBphysical\fR property sets the \fBmax-rss\fR for
856 this zone. This will be enforced by \fBrcapd\fR(1M) running in the global zone.
857 The \fBswap\fR property is the preferred way to set the \fBzone.max-swap\fR
858 rctl. The \fBblocked\fR property is the preferred way to set the
859 \fBzone.max-locked-memory\fR rctl.
860 .RE

862 .sp
863 .ne 2
864 .na
865 \fB\fBcapped-cpu\fR: ncpus\fR
866 .ad
867 .sp .6
868 .RS 4n
869 Sets a limit on the amount of CPU time that can be used by a zone. The unit
870 used translates to the percentage of a single CPU that can be used by all user
871 threads in a zone, expressed as a fraction (for example, \fB&.75\fR) or a
872 mixed number (whole number and fraction, for example, \fB1.25\fR). An
873 \fBncpu\fR value of \fB1\fR means 100% of a CPU, a value of \fB1.25\fR means
874 125%, \fB&.75\fR mean 75%, and so forth. When projects within a capped zone
875 have their own caps, the minimum value takes precedence.
876 .sp
877 The \fBcapped-cpu\fR property is an alias for \fBzone.cpu-cap\fR resource
878 control and is related to the \fBzone.cpu-cap\fR resource control. See
879 \fBresource_controls\fR(5).
880 .RE

882 .sp
883 .ne 2
884 .na
885 \fB\fBsecurity-flags\fR: lower, default, upper\fR
886 .ad
887 .sp .6
888 .RS 4n
889 Set the process security flags associated with the zone. The \fBlower\fR and
890 \fBupper\fR fields set the limits, the \fBdefault\fR field is set of flags all
891 zone processes inherit.
892 .RE

894 .sp
895 .ne 2
896 .na
897 \fBglobal: \fBfs-allowed\fR\fR
898 .ad
899 .sp .6
900 .RS 4n
901 A comma-separated list of additional filesystems that may be mounted within
902 the zone; for example "ufs,pcfs". By default, only hfs(7fs) and network
903 filesystems can be mounted. If the first entry in the list is "-" then
904 that disables all of the default filesystems. If any filesystems are listed
905 after "-" then only those filesystems can be mounted.

907 This property does not apply to filesystems mounted into the zone via "add fs"
908 or "add dataset".

910 WARNING: allowing filesystem mounts other than the default may allow the zone
911 administrator to compromise the system with a malicious filesystem image, and
912 is not supported.
913 .RE

915 .sp
916 .LP
917 The following table summarizes resources, property-names, and types:
918 .sp

```

919 .in +2
920 .nf
921 resource          property-name  type
922 (global)         zonename       simple
923 (global)         zonepath       simple
924 (global)         autoboot       simple
925 (global)         bootargs       simple
926 (global)         pool           simple
927 (global)         limitpriv      simple
928 (global)         brand           simple
929 (global)         ip-type        simple
930 (global)         hostid         simple
931 (global)         cpu-shares     simple
932 (global)         max-lwps       simple
933 (global)         max-msg-ids    simple
934 (global)         max-sem-ids    simple
935 (global)         max-shm-ids    simple
936 (global)         max-shm-memory simple
937 (global)         scheduling-class simple
938 fs               dir             simple
939                 special        simple
940                 raw            simple
941                 type           simple
942                 options        list of simple
943 net              address        simple
944                 physical       simple
945 device          match          simple
946 rctl            name           simple
947                 value          list of complex
948 attr            name           simple
949                 type           simple
950                 value          simple
951 dataset         name           simple
952 dedicated-cpu   ncpus          simple or range
953                 importance      simple

955 capped-memory   physical       simple with scale
956                 swap           simple with scale
957                 locked         simple with scale

959 capped-cpu      ncpus          simple
960 security-flags  lower          simple
961                 default        simple
962                 upper          simple
963 .fi
964 .in -2
965 .sp

967 .sp
968 .LP
969 To further specify things, the breakdown of the complex property "value" of the
970 "rctl" resource type, it consists of three name/value pairs, the names being
971 "priv", "limit" and "action", each of which takes a simple value. The "name"
972 property of an "attr" resource is syntactically restricted in a fashion similar
973 but not identical to zone names: it must begin with an alphanumeric, and can
974 contain alphanumerics plus the hyphen (\fB-\fR), underscore (\fB_\fR), and dot
975 (\fB\&.\fR) characters. Attribute names beginning with "zone" are reserved for
976 use by the system. Finally, the "autoboot" global property must have a value of
977 "true" or "false".
978 .SS "Using Kernel Statistics to Monitor CPU Caps"
979 .LP
980 Using the kernel statistics (\fBkstat\fR(3KSTAT)) module \fBcaps\fR, the system
981 maintains information for all capped projects and zones. You can access this
982 information by reading kernel statistics (\fBkstat\fR(3KSTAT)), specifying
983 \fBcaps\fR as the \fBkstat\fR module name. The following command displays
984 kernel statistics for all active CPU caps:

```

```

985 .sp
986 .in +2
987 .nf
988 # \fBkstat caps::'/cpucaps/'\fR
989 .fi
990 .in -2
991 .sp

993 .sp
994 .LP
995 A \fBkstat\fR(1M) command running in a zone displays only CPU caps relevant for
996 that zone and for projects in that zone. See \fBEXAMPLES\fR.
997 .sp
998 .LP
999 The following are cap-related arguments for use with \fBkstat\fR(1M):
1000 .sp
1001 .ne 2
1002 .na
1003 \fB\fBcaps\fR\fR
1004 .ad
1005 .sp .6
1006 .RS 4n
1007 The \fBkstat\fR module.
1008 .RE

1010 .sp
1011 .ne 2
1012 .na
1013 \fB\fBproject_caps\fR or \fB\fBzone_caps\fR\fR
1014 .ad
1015 .sp .6
1016 .RS 4n
1017 \fBkstat\fR class, for use with the \fBkstat\fR \fB-c\fR option.
1018 .RE

1020 .sp
1021 .ne 2
1022 .na
1023 \fB\fBcpucaps_project_\fR\fR or \fB\fBcpucaps_zone_\fR\fR
1024 .ad
1025 .sp .6
1026 .RS 4n
1027 \fBkstat\fR name, for use with the \fBkstat\fR \fB-n\fR option. \fR is the
1028 project or zone identifier.
1029 .RE

1031 .sp
1032 .LP
1033 The following fields are displayed in response to a \fBkstat\fR(1M) command
1034 requesting statistics for all CPU caps.
1035 .sp
1036 .ne 2
1037 .na
1038 \fB\fBmodule\fR\fR
1039 .ad
1040 .sp .6
1041 .RS 4n
1042 In this usage of \fBkstat\fR, this field will have the value \fBcaps\fR.
1043 .RE

1045 .sp
1046 .ne 2
1047 .na
1048 \fB\fBname\fR\fR
1049 .ad
1050 .sp .6

```



```

1051 .RS 4n
1052 As described above, \fBcpucaps_project_\fR\fIid\fR or
1053 \fBcpucaps_zone_\fR\fIid\fR
1054 .RE

1056 .sp
1057 .ne 2
1058 .na
1059 \fB\babove_sec\fR\fR
1060 .ad
1061 .sp .6
1062 .RS 4n
1063 Total time, in seconds, spent above the cap.
1064 .RE

1066 .sp
1067 .ne 2
1068 .na
1069 \fB\bbelow_sec\fR\fR
1070 .ad
1071 .sp .6
1072 .RS 4n
1073 Total time, in seconds, spent below the cap.
1074 .RE

1076 .sp
1077 .ne 2
1078 .na
1079 \fB\bmaxusage\fR\fR
1080 .ad
1081 .sp .6
1082 .RS 4n
1083 Maximum observed CPU usage.
1084 .RE

1086 .sp
1087 .ne 2
1088 .na
1089 \fB\bwait\fR\fR
1090 .ad
1091 .sp .6
1092 .RS 4n
1093 Number of threads on cap wait queue.
1094 .RE

1096 .sp
1097 .ne 2
1098 .na
1099 \fB\busage\fR\fR
1100 .ad
1101 .sp .6
1102 .RS 4n
1103 Current aggregated CPU usage for all threads belonging to a capped project or
1104 zone, in terms of a percentage of a single CPU.
1105 .RE

1107 .sp
1108 .ne 2
1109 .na
1110 \fB\bvalue\fR\fR
1111 .ad
1112 .sp .6
1113 .RS 4n
1114 The cap value, in terms of a percentage of a single CPU.
1115 .RE

```

```

1117 .sp
1118 .ne 2
1119 .na
1120 \fB\bzonename\fR\fR
1121 .ad
1122 .sp .6
1123 .RS 4n
1124 Name of the zone for which statistics are displayed.
1125 .RE

1127 .sp
1128 .LP
1129 See \fBEXAMPLES\fR for sample output from a \fBkstat\fR command.
1130 .SH OPTIONS
1131 .LP
1132 The following options are supported:
1133 .sp
1134 .ne 2
1135 .na
1136 \fB\b-f\fR \fIcommand_file\fR\fR
1137 .ad
1138 .sp .6
1139 .RS 4n
1140 Specify the name of \fBzonecfg\fR command file. \fIcommand_file\fR is a text
1141 file of \fBzonecfg\fR subcommands, one per line.
1142 .RE

1144 .sp
1145 .ne 2
1146 .na
1147 \fB\b-z\fR \fIzonename\fR\fR
1148 .ad
1149 .sp .6
1150 .RS 4n
1151 Specify the name of a zone. Zone names are case sensitive. Zone names must
1152 begin with an alphanumeric character and can contain alphanumeric characters,
1153 the underscore (\fB_\fR) the hyphen (\fB-\fR), and the dot (\fB&.\fR). The
1154 name \fBglobal\fR and all names beginning with \fB$SUNW\fR are reserved and
1155 cannot be used.
1156 .RE

1158 .SH SUBCOMMANDS
1159 .LP
1160 You can use the \fB\badd\fR and \fB\bselect\fR subcommands to select a specific
1161 resource, at which point the scope changes to that resource. The \fB\bend\fR and
1162 \fB\bcancel\fR subcommands are used to complete the resource specification, at
1163 which time the scope is reverted back to global. Certain subcommands, such as
1164 \fB\badd\fR, \fB\bremove\fR and \fB\bset\fR, have different semantics in each scope.
1165 .sp
1166 .LP
1167 \fBzonecfg\fR supports a semicolon-separated list of subcommands. For example:
1168 .sp
1169 .in +2
1170 .nf
1171 # \fBzonecfg -z myzone "add net; set physical=myvnic; end"\fR
1172 .fi
1173 .in -2
1174 .sp

1176 .sp
1177 .LP
1178 Subcommands which can result in destructive actions or loss of work have an
1179 \fB\b-F\fR option to force the action. If input is from a terminal device, the
1180 user is prompted when appropriate if such a command is given without the
1181 \fB\b-F\fR option otherwise, if such a command is given without the \fB\b-F\fR
1182 option, the action is disallowed, with a diagnostic message written to standard

```

```

1183 error.
1184 .sp
1185 .LP
1186 The following subcommands are supported:
1187 .sp
1188 .ne 2
1189 .na
1190 \fB\badd\fR \fIresource-type\fR (global scope)\fR
1191 .ad
1192 .br
1193 .na
1194 \fB\badd\fR \fIproperty-name property-value\fR (resource scope)\fR
1195 .ad
1196 .sp .6
1197 .RS 4n
1198 In the global scope, begin the specification for a given resource type. The
1199 scope is changed to that resource type.
1200 .sp
1201 In the resource scope, add a property of the given name with the given value.
1202 The syntax for property values varies with different property types. In
1203 general, it is a simple value or a list of simple values enclosed in square
1204 brackets, separated by commas (\fB[foo,bar,baz]\fR). See \fBPROPERTIES\fR.
1205 .RE

1207 .sp
1208 .ne 2
1209 .na
1210 \fB\bcancel\fR\fR
1211 .ad
1212 .sp .6
1213 .RS 4n
1214 End the resource specification and reset scope to global. Abandons any
1215 partially specified resources. \fB\bcancel\fR is only applicable in the resource
1216 scope.
1217 .RE

1219 .sp
1220 .ne 2
1221 .na
1222 \fB\bclear\fR \fIproperty-name\fR\fR
1223 .ad
1224 .sp .6
1225 .RS 4n
1226 Clear the value for the property.
1227 .RE

1229 .sp
1230 .ne 2
1231 .na
1232 \fB\bcommit\fR\fR
1233 .ad
1234 .sp .6
1235 .RS 4n
1236 Commit the current configuration from memory to stable storage. The
1237 configuration must be committed to be used by \fB\bzoneadm\fR. Until the
1238 in-memory configuration is committed, you can remove changes with the
1239 \fB\brevert\fR subcommand. The \fB\bcommit\fR operation is attempted automatically
1240 upon completion of a \fB\bzoncfg\fR session. Since a configuration must be
1241 correct to be committed, this operation automatically does a verify.
1242 .RE

1244 .sp
1245 .ne 2
1246 .na
1247 \fB\bcreate [\fR\F\F\FB] [\fR \fB-a\fR \fIpath\fR |\fB-b\fR \fB|\fR
1248 \fB-t\fR \fItemplate\fR\FB]\fR\fR

```

```

1249 .ad
1250 .sp .6
1251 .RS 4n
1252 Create an in-memory configuration for the specified zone. Use \fB\bcreate\fR to
1253 begin to configure a new zone. See \fB\bcommit\fR for saving this to stable
1254 storage.
1255 .sp
1256 If you are overwriting an existing configuration, specify the \fB-F\fR option
1257 to force the action. Specify the \fB-t\fR \fItemplate\fR option to create a
1258 configuration identical to \fItemplate\fR, where \fItemplate\fR is the name of
1259 a configured zone.
1260 .sp
1261 Use the \fB-a\fR \fIpath\fR option to facilitate configuring a detached zone on
1262 a new host. The \fIpath\fR parameter is the zonepath location of a detached
1263 zone that has been moved on to this new host. Once the detached zone is
1264 configured, it should be installed using the "\fB\bzoneadm attach\fR" command
1265 (see \fB\bzoneadm\fR(1M)). All validation of the new zone happens during the
1266 \fB\battach\fR process, not during zone configuration.
1267 .sp
1268 Use the \fB-b\fR option to create a blank configuration. Without arguments,
1269 \fB\bcreate\fR applies the Sun default settings.
1270 .RE

1272 .sp
1273 .ne 2
1274 .na
1275 \fB\bdelete [\fR\F\F\FB]\fR\fR
1276 .ad
1277 .sp .6
1278 .RS 4n
1279 Delete the specified configuration from memory and stable storage. This action
1280 is instantaneous, no commit is necessary. A deleted configuration cannot be
1281 reverted.
1282 .sp
1283 Specify the \fB-F\fR option to force the action.
1284 .RE

1286 .sp
1287 .ne 2
1288 .na
1289 \fB\bend\fR\fR
1290 .ad
1291 .sp .6
1292 .RS 4n
1293 End the resource specification. This subcommand is only applicable in the
1294 resource scope. \fB\bzoncfg\fR checks to make sure the current resource is
1295 completely specified. If so, it is added to the in-memory configuration (see
1296 \fB\bcommit\fR for saving this to stable storage) and the scope reverts to
1297 global. If the specification is incomplete, it issues an appropriate error
1298 message.
1299 .RE

1301 .sp
1302 .ne 2
1303 .na
1304 \fB\bexport [\fR\F\F\FB \fIoutput-file\fR\FB]\fR\fR
1305 .ad
1306 .sp .6
1307 .RS 4n
1308 Print configuration to standard output. Use the \fB-f\fR option to print the
1309 configuration to \fIoutput-file\fR. This option produces output in a form
1310 suitable for use in a command file.
1311 .RE

1313 .sp
1314 .ne 2

```

```

1315 .na
1316 \fB\bhelp [usage] [\fIsubcommand\fR] [syntax] [\fR\fIcommand-name\fR\fB]\fR\fR
1317 .ad
1318 .sp .6
1319 .RS 4n
1320 Print general help or help about given topic.
1321 .RE

```

```

1323 .sp
1324 .ne 2
1325 .na
1326 \fB\binfo zonename | zonepath | autoboot | brand | pool | limitpriv\fR\fR
1327 .ad
1328 .br
1329 .na
1330 \fB\binfo [\fR\fIresource-type\fR
1331 \fB[\fR\fIproperty-name\fR\fB=\fR\fIproperty-value\fR\fB]*]\fR\fR
1332 .ad
1333 .sp .6
1334 .RS 4n
1335 Display information about the current configuration. If \fIresource-type\fR is
1336 specified, displays only information about resources of the relevant type. If
1337 any \fIproperty-name\fR value pairs are specified, displays only information
1338 about resources meeting the given criteria. In the resource scope, any
1339 arguments are ignored, and \fBinfo\fR displays information about the resource
1340 which is currently being added or modified.
1341 .RE

```

```

1343 .sp
1344 .ne 2
1345 .na
1346 \fB\bremove\fR \fIresource-type\fR\fB{\fR\fIproperty-name\fR\fB=\fR\fIproperty
1347 -value\fR\fB}\fR(global scope)\fR
1348 .ad
1349 .sp .6
1350 .RS 4n
1351 In the global scope, removes the specified resource. The \fB[]\fR syntax means
1352 0 or more of whatever is inside the square braces. If you want only to remove a
1353 single instance of the resource, you must specify enough property name-value
1354 pairs for the resource to be uniquely identified. If no property name-value
1355 pairs are specified, all instances will be removed. If there is more than one
1356 pair is specified, a confirmation is required, unless you use the \fB-F\fR
1357 option.
1358 .RE

```

```

1360 .sp
1361 .ne 2
1362 .na
1363 \fB\bselect\fR \fIresource-type\fR
1364 \fB[\fR\fIproperty-name\fR\fB=\fR\fIproperty-value\fR\fB]\fR\fR
1365 .ad
1366 .sp .6
1367 .RS 4n
1368 Select the resource of the given type which matches the given
1369 \fIproperty-name\fR \fIproperty-value\fR pair criteria, for modification. This
1370 subcommand is applicable only in the global scope. The scope is changed to that
1371 resource type. The \fB{\}\fR syntax means 1 or more of whatever is inside the
1372 curly braces. You must specify enough \fIproperty -name property-value\fR pairs
1373 for the resource to be uniquely identified.
1374 .RE

```

```

1376 .sp
1377 .ne 2
1378 .na
1379 \fB\bset\fR \fIproperty-name\fR\fB=\fR\fIproperty\fR\fB-\fR\fIvalue\fR\fR
1380 .ad

```

```

1381 .sp .6
1382 .RS 4n
1383 Set a given property name to the given value. Some properties (for example,
1384 \fBzonename\fR and \fBzonepath\fR) are global while others are
1385 resource-specific. This subcommand is applicable in both the global and
1386 resource scopes.
1387 .RE

```

```

1389 .sp
1390 .ne 2
1391 .na
1392 \fB\bverify\fR\fR
1393 .ad
1394 .sp .6
1395 .RS 4n
1396 Verify the current configuration for correctness:
1397 .RS +4
1398 .TP
1399 .ie t \(\bu
1400 .el o
1401 All resources have all of their required properties specified.
1402 .RE
1403 .RS +4
1404 .TP
1405 .ie t \(\bu
1406 .el o
1407 A \fBzonepath\fR is specified.
1408 .RE
1409 .RE

```

```

1411 .sp
1412 .ne 2
1413 .na
1414 \fB\bvert\fR \fB[\fR\fB-F\fR\fB]\fR\fR
1415 .ad
1416 .sp .6
1417 .RS 4n
1418 Revert the configuration back to the last committed state. The \fB-F\fR option
1419 can be used to force the action.
1420 .RE

```

```

1422 .sp
1423 .ne 2
1424 .na
1425 \fB\bexit [\fR\fB-F\fR\fB]\fR\fR
1426 .ad
1427 .sp .6
1428 .RS 4n
1429 Exit the \fBzonecfg\fR session. A commit is automatically attempted if needed.
1430 You can also use an \fBEOF\fR character to exit \fBzonecfg\fR. The \fB-F\fR
1431 option can be used to force the action.
1432 .RE

```

```

1434 .SH EXAMPLES
1435 .LP
1436 \fBExample 1 \fRCreating the Environment for a New Zone
1437 .sp
1438 .LP
1439 In the following example, \fBzonecfg\fR creates the environment for a new zone.
1440 \fB/usr/local\fR is loopback mounted from the global zone into
1441 \fB/opt/local\fR. \fB/opt/sfw\fR is loopback mounted from the global zone,
1442 three logical network interfaces are added, and a limit on the number of
1443 fair-share scheduler (FSS) CPU shares for a zone is set using the \fBBrctl\fR
1444 resource type. The example also shows how to select a given resource for
1445 modification.

```

```

1447 .sp
1448 .in +2
1449 .nf
1450 example# \fBzonecfg -z myzone3\fR
1451 my-zone3: No such zone configured
1452 Use 'create' to begin configuring a new zone.
1453 zonecfg:myzone3> \fBcreate\fR
1454 zonecfg:myzone3> \fBset zonepath=/export/home/my-zone3\fR
1455 zonecfg:myzone3> \fBset autoboot=true\fR
1456 zonecfg:myzone3> \fBadd fs\fR
1457 zonecfg:myzone3:fs> \fBset dir=/usr/local\fR
1458 zonecfg:myzone3:fs> \fBset special=/opt/local\fR
1459 zonecfg:myzone3:fs> \fBset type=lofs\fR
1460 zonecfg:myzone3:fs> \fBadd options [ro,nodevices]\fR
1461 zonecfg:myzone3:fs> \fBend\fR
1462 zonecfg:myzone3> \fBadd fs\fR
1463 zonecfg:myzone3:fs> \fBset dir=/mnt\fR
1464 zonecfg:myzone3:fs> \fBset special=/dev/dsk/c0t0d0s7\fR
1465 zonecfg:myzone3:fs> \fBset raw=/dev/rdisk/c0t0d0s7\fR
1466 zonecfg:myzone3:fs> \fBset type=ufs\fR
1467 zonecfg:myzone3:fs> \fBend\fR
1468 zonecfg:myzone3> \fBadd net\fR
1469 zonecfg:myzone3:net> \fBset address=192.168.0.1/24\fR
1470 zonecfg:myzone3:net> \fBset physical=eri0\fR
1471 zonecfg:myzone3:net> \fBend\fR
1472 zonecfg:myzone3> \fBadd net\fR
1473 zonecfg:myzone3:net> \fBset address=192.168.1.2/24\fR
1474 zonecfg:myzone3:net> \fBset physical=eri0\fR
1475 zonecfg:myzone3:net> \fBend\fR
1476 zonecfg:myzone3> \fBadd net\fR
1477 zonecfg:myzone3:net> \fBset address=192.168.2.3/24\fR
1478 zonecfg:myzone3:net> \fBset physical=eri0\fR
1479 zonecfg:myzone3:net> \fBend\fR
1480 zonecfg:my-zone3> \fBset cpu-shares=5\fR
1481 zonecfg:my-zone3> \fBadd capped-memory\fR
1482 zonecfg:my-zone3:capped-memory> \fBset physical=50m\fR
1483 zonecfg:my-zone3:capped-memory> \fBset swap=100m\fR
1484 zonecfg:my-zone3:capped-memory> \fBend\fR
1485 zonecfg:myzone3> \fBexit\fR
1486 .fi
1487 .in -2
1488 .sp

1490 .LP
1491 \fBExample 2 \fRCreating a Non-Native Zone
1492 .sp
1493 .LP
1494 The following example creates a new Linux zone:

1496 .sp
1497 .in +2
1498 .nf
1499 example# \fBzonecfg -z lxzone\fR
1500 lxzone: No such zone configured
1501 Use 'create' to begin configuring a new zone
1502 zonecfg:lxzone> \fBcreate -t SUNWlx\fR
1503 zonecfg:lxzone> \fBset zonepath=/export/zones/lxzone\fR
1504 zonecfg:lxzone> \fBset autoboot=true\fR
1505 zonecfg:lxzone> \fBexit\fR
1506 .fi
1507 .in -2
1508 .sp

1510 .LP
1511 \fBExample 3 \fRCreating an Exclusive-IP Zone
1512 .sp

```

```

1513 .LP
1514 The following example creates a zone that is granted exclusive access to
1515 \fBbge1\fR and \fBbge33000\fR and that is isolated at the IP layer from the
1516 other zones configured on the system.

1518 .sp
1519 .LP
1520 The IP addresses and routing is configured inside the new zone using
1521 \fBsysidtool\fR(1M).

1523 .sp
1524 .in +2
1525 .nf
1526 example# \fBzonecfg -z excl\fR
1527 excl: No such zone configured
1528 Use 'create' to begin configuring a new zone
1529 zonecfg:excl> \fBcreate\fR
1530 zonecfg:excl> \fBset zonepath=/export/zones/excl\fR
1531 zonecfg:excl> \fBset ip-type=exclusive\fR
1532 zonecfg:excl> \fBadd net\fR
1533 zonecfg:excl:net> \fBset physical=bge1\fR
1534 zonecfg:excl:net> \fBend\fR
1535 zonecfg:excl> \fBadd net\fR
1536 zonecfg:excl:net> \fBset physical=bge33000\fR
1537 zonecfg:excl:net> \fBend\fR
1538 zonecfg:excl> \fBexit\fR
1539 .fi
1540 .in -2
1541 .sp

1543 .LP
1544 \fBExample 4 \fRAssociating a Zone with a Resource Pool
1545 .sp
1546 .LP
1547 The following example shows how to associate an existing zone with an existing
1548 resource pool:

1550 .sp
1551 .in +2
1552 .nf
1553 example# \fBzonecfg -z myzone\fR
1554 zonecfg:myzone> \fBset pool=mypool\fR
1555 zonecfg:myzone> \fBexit\fR
1556 .fi
1557 .in -2
1558 .sp

1560 .sp
1561 .LP
1562 For more information about resource pools, see \fBpooladm\fR(1M) and
1563 \fBpoolcfg\fR(1M).

1565 .LP
1566 \fBExample 5 \fRChanging the Name of a Zone
1567 .sp
1568 .LP
1569 The following example shows how to change the name of an existing zone:

1571 .sp
1572 .in +2
1573 .nf
1574 example# \fBzonecfg -z myzone\fR
1575 zonecfg:myzone> \fBset zonename=myzone2\fR
1576 zonecfg:myzone2> \fBexit\fR
1577 .fi
1578 .in -2

```

```

1579 .sp
1581 .LP
1582 \fBExample 6 \fRChanging the Privilege Set of a Zone
1583 .sp
1584 .LP
1585 The following example shows how to change the set of privileges an existing
1586 zone's processes will be limited to the next time the zone is booted. In this
1587 particular case, the privilege set will be the standard safe set of privileges
1588 a zone normally has along with the privilege to change the system date and
1589 time:
1591 .sp
1592 .in +2
1593 .nf
1594 example# \fBzonecfg -z myzone\fR
1595 zonecfg:myzone> \fBset limitpriv="default,sys_time"\fR
1596 zonecfg:myzone2> \fBexit\fR
1597 .fi
1598 .in -2
1599 .sp
1601 .LP
1602 \fBExample 7 \fRSetting the \fBzone.cpu-shares\fR Property for the Global Zone
1603 .sp
1604 .LP
1605 The following command sets the \fBzone.cpu-shares\fR property for the global
1606 zone:
1608 .sp
1609 .in +2
1610 .nf
1611 example# \fBzonecfg -z global\fR
1612 zonecfg:global> \fBset cpu-shares=5\fR
1613 zonecfg:global> \fBexit\fR
1614 .fi
1615 .in -2
1616 .sp
1618 .LP
1619 \fBExample 8 \fRUsing Pattern Matching
1620 .sp
1621 .LP
1622 The following commands illustrate \fBzonecfg\fR support for pattern matching.
1623 In the zone \fBflexlm\fR, enter:
1625 .sp
1626 .in +2
1627 .nf
1628 zonecfg:flexlm> \fBadd device\fR
1629 zonecfg:flexlm:device> \fBset match="/dev/cua/a00[2-5]"\fR
1630 zonecfg:flexlm:device> \fBend\fR
1631 .fi
1632 .in -2
1633 .sp
1635 .sp
1636 .LP
1637 In the global zone, enter:
1639 .sp
1640 .in +2
1641 .nf
1642 global# \fBls /dev/cua\fR
1643 a      a000 a001 a002 a003 a004 a005 a006 a007 b
1644 .fi

```

```

1645 .in -2
1646 .sp
1648 .sp
1649 .LP
1650 In the zone \fBflexlm\fR, enter:
1652 .sp
1653 .in +2
1654 .nf
1655 flexlm# \fBls /dev/cua\fR
1656 a002 a003 a004 a005
1657 .fi
1658 .in -2
1659 .sp
1661 .LP
1662 \fBExample 9 \fRSetting a Cap for a Zone to Three CPUs
1663 .sp
1664 .LP
1665 The following sequence uses the \fBzonecfg\fR command to set the CPU cap for a
1666 zone to three CPUs.
1668 .sp
1669 .in +2
1670 .nf
1671 zonecfg:myzone> \fBadd capped-cpu\fR
1672 zonecfg:myzone>capped-cpu> \fBset ncpus=3\fR
1673 zonecfg:myzone>capped-cpu>capped-cpu> \fBend\fR
1674 .fi
1675 .in -2
1676 .sp
1678 .sp
1679 .LP
1680 The preceding sequence, which uses the capped-cpu property, is equivalent to
1681 the following sequence, which makes use of the \fBzone.cpu-cap\fR resource
1682 control.
1684 .sp
1685 .in +2
1686 .nf
1687 zonecfg:myzone> \fBadd rctl\fR
1688 zonecfg:myzone:rctl> \fBset name=zone.cpu-cap\fR
1689 zonecfg:myzone:rctl> \fBadd value (priv=privileged,limit=300,action=none)\fR
1690 zonecfg:myzone:rctl> \fBend\fR
1691 .fi
1692 .in -2
1693 .sp
1695 .LP
1696 \fBExample 10 \fRUsing \fBkstat\fR to Monitor CPU Caps
1697 .sp
1698 .LP
1699 The following command displays information about all CPU caps.
1701 .sp
1702 .in +2
1703 .nf
1704 # \fBkstat -n /cpucaps/\fR
1705 module: caps                               instance: 0
1706 name:   cpucaps_project_0                  class:   project_caps
1707        above_sec                            0
1708        below_sec                             2157
1709        crtime                                821.048183159
1710        maxusage                               2

```

```

1711         nwait                0
1712         snaptime              235885.637253027
1713         usage                 0
1714         value                 18446743151372347932
1715         zonename             global

1717 module: caps                instance: 0
1718 name:      cpucaps_project_1 class:   project_caps
1719         above_sec            0
1720         below_sec           0
1721         crtime              225339.192787265
1722         maxusage            5
1723         nwait                0
1724         snaptime            235885.637591677
1725         usage                 5
1726         value                 18446743151372347932
1727         zonename             global

1729 module: caps                instance: 0
1730 name:      cpucaps_project_201 class:  project_caps
1731         above_sec            0
1732         below_sec           235105
1733         crtime              780.37961782
1734         maxusage            100
1735         nwait                0
1736         snaptime            235885.637789687
1737         usage                 43
1738         value                 100
1739         zonename             global

1741 module: caps                instance: 0
1742 name:      cpucaps_project_202 class:  project_caps
1743         above_sec            0
1744         below_sec           235094
1745         crtime              791.72983782
1746         maxusage            100
1747         nwait                0
1748         snaptime            235885.637967512
1749         usage                 48
1750         value                 100
1751         zonename             global

1753 module: caps                instance: 0
1754 name:      cpucaps_project_203 class:  project_caps
1755         above_sec            0
1756         below_sec           235034
1757         crtime              852.104401481
1758         maxusage            75
1759         nwait                0
1760         snaptime            235885.638144304
1761         usage                 47
1762         value                 100
1763         zonename             global

1765 module: caps                instance: 0
1766 name:      cpucaps_project_86710 class: project_caps
1767         above_sec            22
1768         below_sec           235166
1769         crtime              698.441717859
1770         maxusage            101
1771         nwait                0
1772         snaptime            235885.638319871
1773         usage                 54
1774         value                 100
1775         zonename             global

```

```

1777 module: caps                instance: 0
1778 name:      cpucaps_zone_0    class:   zone_caps
1779         above_sec            100733
1780         below_sec           134332
1781         crtime              821.048177123
1782         maxusage            207
1783         nwait                2
1784         snaptime            235885.638497731
1785         usage                 199
1786         value                 200
1787         zonename             global

1789 module: caps                instance: 1
1790 name:      cpucaps_project_0  class:  project_caps
1791         above_sec            0
1792         below_sec           0
1793         crtime              225360.256448422
1794         maxusage            7
1795         nwait                0
1796         snaptime            235885.638714404
1797         usage                 7
1798         value                 18446743151372347932
1799         zonename             test_001

1801 module: caps                instance: 1
1802 name:      cpucaps_zone_1    class:  zone_caps
1803         above_sec            2
1804         below_sec           10524
1805         crtime              225360.256440278
1806         maxusage            106
1807         nwait                0
1808         snaptime            235885.638896443
1809         usage                 7
1810         value                 100
1811         zonename             test_001
1812         .fi
1813         .in -2
1814         .sp

1816 .LP
1817 \fBExample 11 \fRDisplaying CPU Caps for a Specific Zone or Project
1818 .sp
1819 .LP
1820 Using the \fBkstat\fR \fB-c\fR and \fB-i\fR options, you can display CPU caps
1821 for a specific zone or project, as below. The first command produces a display
1822 for a specific project, the second for the same project within zone 1.

1824 .sp
1825 .in +2
1826 .nf
1827 # \fBkstat -c project_caps\fR

1829 # \fBkstat -c project_caps -i 1\fR
1830 .fi
1831 .in -2
1832 .sp

1834 .SH EXIT STATUS
1835 .LP
1836 The following exit values are returned:
1837 .sp
1838 .ne 2
1839 .na
1840 \fB0\fR
1841 .ad
1842 .sp .6

```

```
1843 .RS 4n
1844 Successful completion.
1845 .RE

1847 .sp
1848 .ne 2
1849 .na
1850 \fB\fB1\fR\fR
1851 .ad
1852 .sp .6
1853 .RS 4n
1854 An error occurred.
1855 .RE

1857 .sp
1858 .ne 2
1859 .na
1860 \fB\fB2\fR\fR
1861 .ad
1862 .sp .6
1863 .RS 4n
1864 Invalid usage.
1865 .RE

1867 .SH ATTRIBUTES
1868 .LP
1869 See \fBattributes\fR(5) for descriptions of the following attributes:
1870 .sp

1872 .sp
1873 .TS
1874 box;
1875 c | c
1876 l | l .
1877 ATTRIBUTE TYPE ATTRIBUTE VALUE
1878 _
1879 Interface Stability Volatile
1880 .TE

1882 .SH SEE ALSO
1883 .LP
1884 \fBbpriv\fR(1), \fBbprctl\fR(1), \fBbzlogin\fR(1), \fBkstat\fR(1M),
1885 \fBmount\fR(1M), \fBpooladm\fR(1M), \fBpoolcfg\fR(1M), \fBpoold\fR(1M),
1886 \fBbrcapd\fR(1M), \fBbrctladm\fR(1M), \fBbsvcadm\fR(1M), \fBsysidtool\fR(1M),
1887 \fBzfs\fR(1M), \fBzoneadm\fR(1M), \fBpriv_str_to_set\fR(3C),
1888 \fBkstat\fR(3KSTAT), \fBbvfstab\fR(4), \fBattributes\fR(5), \fBbrands\fR(5),
1889 \fBfnmatch\fR(5), \fBlx\fR(5), \fBprivileges\fR(5), \fBresource_controls\fR(5),
1890 \fBsecurity-flags\fR(5), \fBzones\fR(5)
1891 .sp
1892 .LP
1893 \fISystem Administration Guide: Solaris Containers-Resource Management, and
1894 Solaris Zones\fR
1895 .SH NOTES
1896 .LP
1897 All character data used by \fBzonecfg\fR must be in US-ASCII encoding.
```

34589 Wed Jun 15 19:31:42 2016

new/usr/src/man/man3lib/libproc.3lib

Code review comments from jeffpc

```

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 Pctld
205 .It Sy Pdelbkpt Ta Sy Pdelwapt
206 .It Sy Pdstop Ta Sy Pexecname
207 .It Sy Pfault Ta Sy Pfgcore
208 .It Sy Pgcure Ta Sy Pgetareg
209 .It Sy Pgetauxval Ta Sy Pgetauxvec
210 .It Sy Pgetenv Ta Sy Pisprocdir
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_getfpregs
215 .It Sy Plwp_getpsinfo Ta Sy Plwp_getregs
216 .It Sy Plwp_getpymaster Ta Sy Plwp_main_stack
217 .It Sy Plwp_setfpregs 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 Psecflags 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 Psetsysexit 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 .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 Lctld
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_fcctl
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 procfltset2str 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 Pgcov 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;
unchanged portion omitted
```

1637 Wed Jun 15 19:31:44 2016

new/usr/src/man/man3proc/Psecflags.3proc

Code review comments from jeffpc

```

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
14 .Dd May 14, 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
25 .Fo Ppriv
26 .Fa "struct ps_prochandle *p"
27 .Fa "prsecflags_t **psf"
28 .Fc
29 .Ft void
30 .Fo Psecflags_free
30 .Fo Ppriv_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,
62 is updated with a pointer to the allocated privilege set. 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

```

16868 Wed Jun 15 19:31:45 2016

new/usr/src/man/man4/core.4

Code review comments from jeffpc

```

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"
10 .SH NAME
11 core \- process core file
12 .SH DESCRIPTION
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 \fBgcrcore\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 \fBbproc\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 \fBbcoreadm\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 \fBbcore\fR. Therefore, by
36 default, the operating system attempts to create a core file named \fBbcore\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 \fBbcoreadm\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 \fBba.out\fR(4)), the core file generated is
121 also an ELF file, containing ELF program and file headers. The \fBBe_type\fR
122 field in the file header has type \fBET_CORE\fR. 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 \fBbcoreadm\fR(1M) are also part of the core image. Each program header has its
126 \fBp_memsz\fR 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`\fR field set the same as `\fBp_memsz`\fR, otherwise `\fBp_filesz`\fR is
 129 `\fBzero`\fR.
 130 .sp
 131 .LP
 132 A mapping's data can be excluded due to the core file content settings (see
 133 `\fBcoreadm`\fR(1M)), 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`\fR flag
 137 set in its `\fBp_flags`\fR field; if the data is excluded because of a signal,
 138 the segment's `\fBp_flags`\fR field will have the `\fBPF_SUNW_KILLED`\fR
 139 flag set.
 140 .sp
 141 .LP
 142 The program headers of an `\fBELF`\fR core file also contain entries for two
 143 `\fBNOTE`\fR segments, each containing several note entries as described below.
 144 The note entry header and core file note type (`\fBn_type`\fR) definitions are
 145 contained in `<\fBsys/elf.h`\fR>. The first `\fBNOTE`\fR segment exists for binary
 146 compatibility with old programs that deal with core files. It contains
 147 structures defined in `<\fBsys/old_procfs.h`\fR>. New programs should recognize
 148 and skip this `\fBNOTE`\fR segment, advancing instead to the new `\fBNOTE`\fR
 149 segment. The old `\fBNOTE`\fR segment is deleted from core files in a future
 150 release.
 151 .sp
 152 .LP
 153 The old `\fBNOTE`\fR segment contains the following entries. Each has entry name
 154 `\fB"CORE"`\fR and presents the contents of a system structure:
 155 .sp
 156 .ne 2
 157 .na
 158 `\fB\fBprpsinfo_t`\fR
 159 .ad
 160 .RS 16n
 161 `\fBn_type`\fR: `\fBNT_PRPSINFO`\fR. This entry contains information of interest to
 162 the `\fBps`\fR(1) command, such as process status, `\fBCPU`\fR usage, `\fBnice`\fR
 163 value, controlling terminal, user-ID, process-ID, the name of the executable,
 164 and so forth. The `\fBprpsinfo_t`\fR structure is defined in
 165 `<\fBsys/old_procfs.h`\fR>.
 166 .RE
 168 .sp
 169 .ne 2
 170 .na
 171 `\fB\fBbchar`\fR array
 172 .ad
 173 .RS 16n
 174 `\fBn_type`\fR: `\fBNT_PLATFORM`\fR. 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`\fR(2) when invoked with
 177 the command `\fBBI_PLATFORM`\fR.
 178 .RE
 180 .sp
 181 .ne 2
 182 .na
 183 `\fB\fBauxv_t`\fR array
 184 .ad
 185 .RS 16n
 186 `\fBn_type`\fR: `\fBNT_AUXV`\fR. This entry contains the array of `\fBauxv_t`\fR
 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`\fR>.
 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`\fR segment contains an entry with a
 196 `\fBprstatus_t`\fR structure, plus other optionally-present entries describing
 197 the LWP, as follows:
 198 .sp
 199 .ne 2
 200 .na
 201 `\fB\fBprstatus_t`\fR
 202 .ad
 203 .RS 16n
 204 `\fBn_type`\fR: `\fBNT_PRSTATUS`\fR. 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`\fR structure is defined in `<\fBsys/old_procfs.h`\fR>.
 208 .RE
 210 .sp
 211 .ne 2
 212 .na
 213 `\fB\fBprfpregset_t`\fR
 214 .ad
 215 .RS 16n
 216 `\fBn_type`\fR: `\fBNT_PRFPREG`\fR. This entry is present only if the `\fBBLWP`\fR
 217 used the floating-point hardware. It contains the floating-point registers. The
 218 `\fBprfpregset_t`\fR structure is defined in `<\fBsys/procfs_isa.h`\fR>.
 219 .RE
 221 .sp
 222 .ne 2
 223 .na
 224 `\fB\fBgwindows_t`\fR
 225 .ad
 226 .RS 16n
 227 `\fBn_type`\fR: `\fBNT_GWINDOWS`\fR. 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`\fR
 230 structure is defined in `<\fBsys/regset.h`\fR>.
 231 .RE
 233 .sp
 234 .ne 2
 235 .na
 236 `\fB\fBprxregset_t`\fR
 237 .ad
 238 .RS 16n
 239 `\fBn_type`\fR: `\fBNT_PRXREG`\fR. 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`\fR structure is defined in `<\fBsys/procfs_isa.h`\fR>.
 242 .RE
 244 .sp
 245 .LP
 246 The new `\fBNOTE`\fR segment contains the following entries. Each has entry name
 247 `"\fBCORE`\fR" and presents the contents of a system structure:
 248 .sp
 249 .ne 2
 250 .na
 251 `\fB\fBpsinfo_t`\fR
 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 \fBBSI_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\fBbstruct 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\fBbprcred_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 \fBgid_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_ZONE_NAME\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 \fBgetzonebynameid\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\fBbprfdinfo_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 \fBbstat\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\fBbstruct 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\fBbcore_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 \fBbcoreadm\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\fBblwpsinfo_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 \fBBLWP\fR status, \fBBCPU\fR usage,
388 \fBbnice\fR value, \fBBLWP-ID\fR, and so forth. The \fBblwpsinfo_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. 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 .LP
465 Depending on the \fBcoreadm\fR(1M) settings, the section header of an ELF core
466 file can contain entries for CTF, symbol table, and string table sections. The
467 \fBsh_addr\fR fields are set to the base address of the first mapping of the
468 load object that they came from to. This can be used to match those sections
469 with the corresponding load object.
470 .sp
471 .LP
472 The size of the core file created by a process can be controlled by the user
473 (see \fBgetrlimit\fR(2)).
474 .SH SEE ALSO
475 .LP
476 \fBelfdump\fR(1), \fBgcore\fR(1), \fBmddb\fR(1), \fBproc\fR(1), \fBps\fR(1),
477 \fBcoreadm\fR(1M), \fBgetrlimit\fR(2), \fBsetrlimit\fR(2), \fBsetuid\fR(2),
478 \fBsysinfo\fR(2), \fBuname\fR(2), \fBgetzonebyname\fR(3C),
479 \fBgetzoneid\fR(3C), \fBelf\fR(3ELF), \fBsignal.h\fR(3HEAD), \fBa.out\fR(4),
480 \fBproc\fR(4), \fBzones\fR(5), \fBsecurity-flags\fR(5)
481 .sp
482 .LP
483 \fBIANSI C Programmer's Guide\fR

```

 99687 Wed Jun 15 19:31:46 2016

new/usr/src/man/man4/proc.4

Code review comments from jeffpc

```

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"
8  .TH PROC 4 "Jul 23, 2015"
9  .SH NAME
10 proc \- /proc, the process file system
11 .SH DESCRIPTION
12 .LP
13 \fB/proc\fR is a file system that provides access to the state of each process
14 and light-weight process (lwp) in the system. The name of each entry in the
15 \fB/proc\fR directory is a decimal number corresponding to a process-ID. These
16 entries are themselves subdirectories. Access to process state is provided by
17 additional files contained within each subdirectory; the hierarchy is described
18 more completely below. In this document, "\fB/proc\fR file" refers to a
19 non-directory file within the hierarchy rooted at \fB/proc\fR. The owner of
20 each \fB/proc\fR file and subdirectory is determined by the user-ID of the
21 process.
22 .sp
23 .LP
24 \fB/proc\fR can be mounted on any mount point, in addition to the standard
25 \fB/proc\fR mount point, and can be mounted several places at once. Such
26 additional mounts are allowed in order to facilitate the confinement of
27 processes to subtrees of the file system via \fBchroot\fR(1M) and yet allow
28 such processes access to commands like \fBps\fR(1).
29 .sp
30 .LP
31 Standard system calls are used to access \fB/proc\fR files: \fBopen\fR(2),
32 \fBclose\fR(2), \fBread\fR(2), and \fBwrite\fR(2) (including \fBreadv\fR(2),
33 \fBwritev\fR(2), \fBpread\fR(2), and \fBpwrite\fR(2)). Most files describe
34 process state and can only be opened for reading. \fBbctl\fR and \fBlwpcctl\fR
35 (control) files permit manipulation of process state and can only be opened for
36 writing. \fBas\fR (address space) files contain the image of the running
37 process and can be opened for both reading and writing. An open for writing
38 allows process control; a read-only open allows inspection but not control. In
39 this document, we refer to the process as open for reading or writing if any of
40 its associated \fB/proc\fR files is open for reading or writing.
41 .sp
42 .LP
43 In general, more than one process can open the same \fB/proc\fR file at the
44 same time. \fBexclusive\fR \fBopen\fR is an advisory mechanism provided to
45 allow controlling processes to avoid collisions with each other. A process can
46 obtain exclusive control of a target process, with respect to other cooperating
47 processes, if it successfully opens any \fB/proc\fR file in the target process
48 for writing (the \fBas\fR or \fBctl\fR files, or the \fBlwpcctl\fR file of any
49 lwp) while specifying \fBEXCL\fR in the \fBopen\fR(2). Such an open will fail
50 if the target process is already open for writing (that is, if an \fBas\fR,
51 \fBctl\fR, or \fBlwpcctl\fR file is already open for writing). There can be any
52 number of concurrent read-only opens; \fBEXCL\fR is ignored on opens for
53 reading. It is recommended that the first open for writing by a controlling
54 process use the \fBEXCL\fR flag; multiple controlling processes usually
55 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 \fBEXCL\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 \fBSETFD\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\fR\fR\fR\fR\fR and
72 \fB/proc/\fR\fR\fR\fR\fR\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 \fBEIO\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, \fBfset_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 \fBlwpcctl\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 .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\fIpid\fR`
 202 .LP
 203 A given directory `\fB/proc/\fR\fIpid\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 .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 .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 .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 `\fBlwpctl\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 `\fBlwpctl\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 .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_____
```

```

*****
34007 Wed Jun 15 19:31:49 2016
new/usr/src/man/man5/privileges.5
Code review comments from jeffpc
*****
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\"
7.TH PRIVILEGES 5 \"Oct 30, 2015\"
8.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 ioctl's 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 \fbtznzonecfg\fr(\fb4\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 \fBFBPRIV_NET_MAC_IMPLICIT\fR\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 \fBFBPRIV_NET_OBSERVABILITY\fR\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 \fBFBPRIV_NET_PRIVADDR\fR\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 \fBFBPRIV_NET_RAWACCESS\fR\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 \fBFBPRIV_PROC_AUDIT\fR\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 \fBFBPRIV_PROC_CHROOT\fR\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 \fBFBPRIV_PROC_CLOCK_HIGHRES\fR\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 \fBFBPRIV_PROC_EXEC\fR\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 \fBFBPRIV_PROC_FORK\fR\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 \fBFBPRIV_PROC_INFO\fR\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 \fBFBPRIV_PROC_LOCK_MEMORY\fR\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 \fb\fbPRIV_PROC_SESSION\fr\fr
587 .ad
588 .sp .6

```

```

589 .RS 4n
590 Allow a process to send signals or trace processes outside its session.
591 .RE

593 .sp
594 .ne 2
595 .na
596 \fb\fbPRIV_PROC_SETID\fr\fr
597 .ad
598 .sp .6
599 .RS 4n
600 Allow a process to set its UIDs at will, assuming UID 0 requires all privileges
601 to be asserted.
602 .RE

604 .sp
605 .ne 2
606 .na
607 \fb\fbPRIV_PROC_TASKID\fr\fr
608 .ad
609 .sp .6
610 .RS 4n
611 Allow a process to assign a new task ID to the calling process.
612 .RE

614 .sp
615 .ne 2
616 .na
617 \fb\fbPRIV_PROC_ZONE\fr\fr
618 .ad
619 .sp .6
620 .RS 4n
621 Allow a process to trace or send signals to processes in other zones. See
622 \fbzones\fr(5).
623 .RE

625 .sp
626 .ne 2
627 .na
628 \fb\fbPRIV_SYS_ACCT\fr\fr
629 .ad
630 .sp .6
631 .RS 4n
632 Allow a process to enable and disable and manage accounting through
633 \fbacct\fr(2).
634 .RE

636 .sp
637 .ne 2
638 .na
639 \fb\fbPRIV_SYS_ADMIN\fr\fr
640 .ad
641 .sp .6
642 .RS 4n
643 Allow a process to perform system administration tasks such as setting node and
644 domain name and specifying \fbcoreadm\fr(1M) and \fbnsd\fr(1M) settings
645 .RE

647 .sp
648 .ne 2
649 .na
650 \fb\fbPRIV_SYS_AUDIT\fr\fr
651 .ad
652 .sp .6
653 .RS 4n
654 Allow a process to start the (kernel) audit daemon. Allow a process to view and

```

```

655 set audit state (audit user ID, audit terminal ID, audit sessions ID, audit
656 pre-selection mask). Allow a process to turn off and on auditing. Allow a
657 process to configure the audit parameters (cache and queue sizes, event to
658 class mappings, and policy options).
659 .RE

661 .sp
662 .ne 2
663 .na
664 \fb\fbPRIV_SYS_CONFIG\fr\fr
665 .ad
666 .sp .6
667 .RS 4n
668 Allow a process to perform various system configuration tasks. Allow
669 filesystem-specific administrative procedures, such as filesystem configuration
670 ioctls, quota calls, creation and deletion of snapshots, and manipulating the
671 PCFS bootsector.
672 .RE

674 .sp
675 .ne 2
676 .na
677 \fb\fbPRIV_SYS_DEVICES\fr\fr
678 .ad
679 .sp .6
680 .RS 4n
681 Allow a process to create device special files. Allow a process to successfully
682 call a kernel module that calls the kernel \fbdrv_priv\fr(9F) function to check
683 for allowed access. Allow a process to open the real console device directly.
684 Allow a process to open devices that have been exclusively opened.
685 .RE

687 .sp
688 .ne 2
689 .na
690 \fb\fbPRIV_SYS_DL_CONFIG\fr\fr
691 .ad
692 .sp .6
693 .RS 4n
694 Allow a process to configure a system's datalink interfaces.
695 .RE

697 .sp
698 .ne 2
699 .na
700 \fb\fbPRIV_SYS_IP_CONFIG\fr\fr
701 .ad
702 .sp .6
703 .RS 4n
704 Allow a process to configure a system's IP interfaces and routes. Allow a
705 process to configure network parameters for \fbTCP/IP\fr using \fbndd\fr. Allow
706 a process access to otherwise restricted \fbTCP/IP\fr information using
707 \fbndd\fr. Allow a process to configure \fbIPsec\fr. Allow a process to pop
708 anchored \fbSTREAM\frs modules with matching \fbzoneid\fr.
709 .RE

711 .sp
712 .ne 2
713 .na
714 \fb\fbPRIV_SYS_IPC_CONFIG\fr\fr
715 .ad
716 .sp .6
717 .RS 4n
718 Allow a process to increase the size of a System V IPC Message Queue buffer.
719 .RE

```

```

721 .sp
722 .ne 2
723 .na
724 \fb\fbPRIV_SYS_IPTUN_CONFIG\fr\fr
725 .ad
726 .sp .6
727 .RS 4n
728 Allow a process to configure IP tunnel links.
729 .RE

731 .sp
732 .ne 2
733 .na
734 \fb\fbPRIV_SYS_LINKDIR\fr\fr
735 .ad
736 .sp .6
737 .RS 4n
738 Allow a process to unlink and link directories.
739 .RE

741 .sp
742 .ne 2
743 .na
744 \fb\fbPRIV_SYS_MOUNT\fr\fr
745 .ad
746 .sp .6
747 .RS 4n
748 Allow a process to mount and unmount filesystems that would otherwise be
749 restricted (that is, most filesystems except \fbnamefs\fr). Allow a process to
750 add and remove swap devices.
751 .RE

753 .sp
754 .ne 2
755 .na
756 \fb\fbPRIV_SYS_NET_CONFIG\fr\fr
757 .ad
758 .sp .6
759 .RS 4n
760 Allow a process to do all that \fbPRIV_SYS_IP_CONFIG\fr,
761 \fbPRIV_SYS_DL_CONFIG\fr, and \fbPRIV_SYS_PPP_CONFIG\fr allow, plus the
762 following: use the \fbbrpcmod\fr STREAMS module and insert/remove STREAMS
763 modules on locations other than the top of the module stack.
764 .RE

766 .sp
767 .ne 2
768 .na
769 \fb\fbPRIV_SYS_NFS\fr\fr
770 .ad
771 .sp .6
772 .RS 4n
773 Allow a process to provide NFS service: start NFS kernel threads, perform NFS
774 locking operations, bind to NFS reserved ports: ports 2049 (\fbnfs\fr) and port
775 4045 (\fblockd\fr).
776 .RE

778 .sp
779 .ne 2
780 .na
781 \fb\fbPRIV_SYS_PPP_CONFIG\fr\fr
782 .ad
783 .sp .6
784 .RS 4n
785 Allow a process to create, configure, and destroy PPP instances with pppd(1M)
786 \fbpppd\fr(1M) and control PPPoE plumbing with \fbppptun\fr(1M)ppptun(1M).

```

```

787 This privilege is granted by default to exclusive IP stack instance zones.
788 .RE

790 .sp
791 .ne 2
792 .na
793 \fB\fBPRIV_SYS_RES_BIND\fR\fR
794 .ad
795 .sp .6
796 .RS 4n
797 Allows a process to bind processes to processor sets.
798 .RE

800 .sp
801 .ne 2
802 .na
803 \fB\fBPRIV_SYS_RES_CONFIG\fR\fR
804 .ad
805 .sp .6
806 .RS 4n
807 Allows all that PRIV_SYS_RES_BIND allows.
808 Allow a process to create and delete processor sets, assign CPUs to processor
809 sets and override the \fBSET_NOESCAPE\fR property. Allow a process to change
810 the operational status of CPUs in the system using \fBp_online\fR(2). Allow a
811 process to configure filesystem quotas. Allow a process to configure resource
812 pools and bind processes to pools.
813 .RE

815 .sp
816 .ne 2
817 .na
818 \fB\fBPRIV_SYS_RESOURCE\fR\fR
819 .ad
820 .sp .6
821 .RS 4n
822 Allow a process to exceed the resource limits imposed on it by
823 \fBsetrlimit\fR(2) and \fBsetrctl\fR(2).
824 .RE

826 .sp
827 .ne 2
828 .na
829 \fB\fBPRIV_SYS_SMB\fR\fR
830 .ad
831 .sp .6
832 .RS 4n
833 Allow a process to provide NetBIOS or SMB services: start SMB kernel threads or
834 bind to NetBIOS or SMB reserved ports: ports 137, 138, 139 (NetBIOS) and 445
835 (SMB).
836 .RE

838 .sp
839 .ne 2
840 .na
841 \fB\fBPRIV_SYS_SUSER_COMPAT\fR\fR
842 .ad
843 .sp .6
844 .RS 4n
845 Allow a process to successfully call a third party loadable module that calls
846 the kernel \fBsbuser()\fR function to check for allowed access. This privilege
847 exists only for third party loadable module compatibility and is not used by
848 Solaris proper.
849 .RE

851 .sp
852 .ne 2

```

```

853 .na
854 \fB\fBPRIV_SYS_TIME\fR\fR
855 .ad
856 .sp .6
857 .RS 4n
858 Allow a process to manipulate system time using any of the appropriate system
859 calls: \fBstime\fR(2), \fBbadjtime\fR(2), and \fBntp_adjtime\fR(2).
860 .RE

862 .sp
863 .ne 2
864 .na
865 \fB\fBPRIV_SYS_TRANS_LABEL\fR\fR
866 .ad
867 .sp .6
868 .RS 4n
869 Allow a process to translate labels that are not dominated by the process's
870 sensitivity label to and from an external string form.
871 .sp
872 This privilege is interpreted only if the system is configured with Trusted
873 Extensions.
874 .RE

876 .sp
877 .ne 2
878 .na
879 \fB\fBPRIV_VIRT_MANAGE\fR\fR
880 .ad
881 .sp .6
882 .RS 4n
883 Allows a process to manage virtualized environments such as \fBxvm\fR(5).
884 .RE

886 .sp
887 .ne 2
888 .na
889 \fB\fBPRIV_WIN_COLORMAP\fR\fR
890 .ad
891 .sp .6
892 .RS 4n
893 Allow a process to override colormap restrictions.
894 .sp
895 Allow a process to install or remove colormaps.
896 .sp
897 Allow a process to retrieve colormap cell entries allocated by other processes.
898 .sp
899 This privilege is interpreted only if the system is configured with Trusted
900 Extensions.
901 .RE

903 .sp
904 .ne 2
905 .na
906 \fB\fBPRIV_WIN_CONFIG\fR\fR
907 .ad
908 .sp .6
909 .RS 4n
910 Allow a process to configure or destroy resources that are permanently retained
911 by the X server.
912 .sp
913 Allow a process to use SetScreenSaver to set the screen saver timeout value
914 .sp
915 Allow a process to use ChangeHosts to modify the display access control list.
916 .sp
917 Allow a process to use GrabServer.
918 .sp

```

```

919 Allow a process to use the SetCloseDownMode request that can retain window,
920 pixmap, colormap, property, cursor, font, or graphic context resources.
921 .sp
922 This privilege is interpreted only if the system is configured with Trusted
923 Extensions.
924 .RE

926 .sp
927 .ne 2
928 .na
929 \fb\fbPRIV_WIN_DAC_READ\fr\fr
930 .ad
931 .sp .6
932 .RS 4n
933 Allow a process to read from a window resource that it does not own (has a
934 different user ID).
935 .sp
936 This privilege is interpreted only if the system is configured with Trusted
937 Extensions.
938 .RE

940 .sp
941 .ne 2
942 .na
943 \fb\fbPRIV_WIN_DAC_WRITE\fr\fr
944 .ad
945 .sp .6
946 .RS 4n
947 Allow a process to write to or create a window resource that it does not own
948 (has a different user ID). A newly created window property is created with the
949 window's user ID.
950 .sp
951 This privilege is interpreted only if the system is configured with Trusted
952 Extensions.
953 .RE

955 .sp
956 .ne 2
957 .na
958 \fb\fbPRIV_WIN_DEVICES\fr\fr
959 .ad
960 .sp .6
961 .RS 4n
962 Allow a process to perform operations on window input devices.
963 .sp
964 Allow a process to get and set keyboard and pointer controls.
965 .sp
966 Allow a process to modify pointer button and key mappings.
967 .sp
968 This privilege is interpreted only if the system is configured with Trusted
969 Extensions.
970 .RE

972 .sp
973 .ne 2
974 .na
975 \fb\fbPRIV_WIN_DGA\fr\fr
976 .ad
977 .sp .6
978 .RS 4n
979 Allow a process to use the direct graphics access (DGA) X protocol extensions.
980 Direct process access to the frame buffer is still required. Thus the process
981 must have MAC and DAC privileges that allow access to the frame buffer, or the
982 frame buffer must be allocated to the process.
983 .sp
984 This privilege is interpreted only if the system is configured with Trusted

```

```

985 Extensions.
986 .RE

988 .sp
989 .ne 2
990 .na
991 \fb\fbPRIV_WIN_DOWNGRADE_SL\fr\fr
992 .ad
993 .sp .6
994 .RS 4n
995 Allow a process to set the sensitivity label of a window resource to a
996 sensitivity label that does not dominate the existing sensitivity label.
997 .sp
998 This privilege is interpreted only if the system is configured with Trusted
999 Extensions.
1000 .RE

1002 .sp
1003 .ne 2
1004 .na
1005 \fb\fbPRIV_WIN_FONTPATH\fr\fr
1006 .ad
1007 .sp .6
1008 .RS 4n
1009 Allow a process to set a font path.
1010 .sp
1011 This privilege is interpreted only if the system is configured with Trusted
1012 Extensions.
1013 .RE

1015 .sp
1016 .ne 2
1017 .na
1018 \fb\fbPRIV_WIN_MAC_READ\fr\fr
1019 .ad
1020 .sp .6
1021 .RS 4n
1022 Allow a process to read from a window resource whose sensitivity label is not
1023 equal to the process sensitivity label.
1024 .sp
1025 This privilege is interpreted only if the system is configured with Trusted
1026 Extensions.
1027 .RE

1029 .sp
1030 .ne 2
1031 .na
1032 \fb\fbPRIV_WIN_MAC_WRITE\fr\fr
1033 .ad
1034 .sp .6
1035 .RS 4n
1036 Allow a process to create a window resource whose sensitivity label is not
1037 equal to the process sensitivity label. A newly created window property is
1038 created with the window's sensitivity label.
1039 .sp
1040 This privilege is interpreted only if the system is configured with Trusted
1041 Extensions.
1042 .RE

1044 .sp
1045 .ne 2
1046 .na
1047 \fb\fbPRIV_WIN_SELECTION\fr\fr
1048 .ad
1049 .sp .6
1050 .RS 4n

```

```

1051 Allow a process to request inter-window data moves without the intervention of
1052 the selection confirmer.
1053 .sp
1054 This privilege is interpreted only if the system is configured with Trusted
1055 Extensions.
1056 .RE

1058 .sp
1059 .ne 2
1060 .na
1061 \fB\FBPRIV_WIN_UPGRADE_SL\fR\fR
1062 .ad
1063 .sp .6
1064 .RS 4n
1065 Allow a process to set the sensitivity label of a window resource to a
1066 sensitivity label that dominates the existing sensitivity label.
1067 .sp
1068 This privilege is interpreted only if the system is configured with Trusted
1069 Extensions.
1070 .RE

1072 .sp
1073 .ne 2
1074 .na
1075 \fB\FBPRIV_XVM_CONTROL\fR\fR
1076 .ad
1077 .sp .6
1078 .RS 4n
1079 Allows a process access to the \fBxVM\fR(5) control devices for managing guest
1080 domains and the hypervisor. This privilege is used only if booted into xVM on
1081 x86 platforms.
1082 .RE

1084 .sp
1085 .LP
1086 Of the privileges listed above, the privileges \fBPRIV_FILE_LINK_ANY\fR,
1087 \fBPRIV_PROC_INFO\fR, \fBPRIV_PROC_SESSION\fR, \fBPRIV_PROC_FORK\fR,
1088 \fBPRIV_FILE_READ\fR, \fBPRIV_FILE_WRITE\fR, \fBPRIV_NET_ACCESS\fR and
1089 \fBPRIV_PROC_EXEC\fR are considered "basic" privileges. These are privileges
1090 that used to be always available to unprivileged processes. By default,
1091 processes still have the basic privileges.
1092 .sp
1093 .LP
1094 The privileges \fBPRIV_PROC_SETID\fR and \fBPRIV_PROC_AUDIT\fR must be present
1095 in the Limit set (see below) of a process in order for set-uid root \fBExec\fRs
1096 to be successful, that is, get an effective UID of 0 and additional privileges.
1097 .sp
1098 .LP
1099 The privilege implementation in Solaris extends the process credential with
1100 four privilege sets:
1101 .sp
1102 .ne 2
1103 .na
1104 \fBFI, the inheritable set\fR
1105 .ad
1106 .RS 26n
1107 The privileges inherited on \fBExec\fR.
1108 .RE

1110 .sp
1111 .ne 2
1112 .na
1113 \fBFBP, the permitted set\fR
1114 .ad
1115 .RS 26n
1116 The maximum set of privileges for the process.

```

```

1117 .RE

1119 .sp
1120 .ne 2
1121 .na
1122 \fBFE, the effective set\fR
1123 .ad
1124 .RS 26n
1125 The privileges currently in effect.
1126 .RE

1128 .sp
1129 .ne 2
1130 .na
1131 \fBBL, the limit set\fR
1132 .ad
1133 .RS 26n
1134 The upper bound of the privileges a process and its offspring can obtain.
1135 Changes to L take effect on the next \fBExec\fR.
1136 .RE

1138 .sp
1139 .LP
1140 The sets I, P and E are typically identical to the basic set of privileges for
1141 unprivileged processes. The limit set is typically the full set of privileges.
1142 .sp
1143 .LP
1144 Each process has a Privilege Awareness State (PAS) that can take the value PA
1145 (privilege-aware) and NPA (not-PA). PAS is a transitional mechanism that allows
1146 a choice between full compatibility with the old superuser model and completely
1147 ignoring the effective UID.
1148 .sp
1149 .LP
1150 To facilitate the discussion, we introduce the notion of "observed effective
1151 set" (oE) and "observed permitted set" (oP) and the implementation sets iE and
1152 iP.
1153 .sp
1154 .LP
1155 A process becomes privilege-aware either by manipulating the effective,
1156 permitted, or limit privilege sets through \fBsetppriv\fR(2) or by using
1157 \fBsetpflags\fR(2). In all cases, oE and oP are invariant in the process of
1158 becoming privilege-aware. In the process of becoming privilege-aware, the
1159 following assignments take place:
1160 .sp
1161 .in +2
1162 .nf
1163 iE = oE
1164 iP = oP
1165 .fi
1166 .in -2

1168 .sp
1169 .LP
1170 When a process is privilege-aware, oE and oP are invariant under UID changes.
1171 When a process is not privilege-aware, oE and oP are observed as follows:
1172 .sp
1173 .in +2
1174 .nf
1175 oE =  $\text{euid} == 0 ? L : iE$ 
1176 oP =  $(\text{euid} == 0 \ || \ \text{ruid} == 0 \ || \ \text{suid} == 0) ? L : iP$ 
1177 .fi
1178 .in -2

1180 .sp
1181 .LP
1182 When a non-privilege-aware process has an effective UID of 0, it can exercise

```

1183 the privileges contained in its limit set, the upper bound of its privileges.
 1184 If a non-privilege-aware process has any of the UIDs 0, it appears to be
 1185 capable of potentially exercising all privileges in L.
 1186 .sp
 1187 .LP
 1188 It is possible for a process to return to the non-privilege aware state using
 1189 `\fbsetpflags()\fR`. The kernel always attempts this on `\fBexec\fR(2)`. This
 1190 operation is permitted only if the following conditions are met:
 1191 .RS +4
 1192 .TP
 1193 .ie t \(\bu
 1194 .el o
 1195 If any of the UIDs is equal to 0, P must be equal to L.
 1196 .RE
 1197 .RS +4
 1198 .TP
 1199 .ie t \(\bu
 1200 .el o
 1201 If the effective UID is equal to 0, E must be equal to L.
 1202 .RE
 1203 .sp
 1204 .LP
 1205 When a process gives up privilege awareness, the following assignments take
 1206 place:
 1207 .sp
 1208 .in +2
 1209 .nf
 1210 if (euid == 0) iE = L & I
 1211 if (any uid == 0) iP = L & I
 1212 .fi
 1213 .in -2
 1215 .sp
 1216 .LP
 1217 The privileges obtained when not having a UID of `\fB0\fR` are the inheritable
 1218 set of the process restricted by the limit set.
 1219 .sp
 1220 .LP
 1221 Only privileges in the process's (observed) effective privilege set allow the
 1222 process to perform restricted operations. A process can use any of the
 1223 privilege manipulation functions to add or remove privileges from the privilege
 1224 sets. Privileges can be removed always. Only privileges found in the permitted
 1225 set can be added to the effective and inheritable set. The limit set cannot
 1226 grow. The inheritable set can be larger than the permitted set.
 1227 .sp
 1228 .LP
 1229 When a process performs an `\fBexec\fR(2)`, the kernel first tries to relinquish
 1230 privilege awareness before making the following privilege set modifications:
 1231 .sp
 1232 .in +2
 1233 .nf
 1234 E' = P' = I' = L & I
 1235 L is unchanged
 1236 .fi
 1237 .in -2
 1239 .sp
 1240 .LP
 1241 If a process has not manipulated its privileges, the privilege sets effectively
 1242 remain the same, as E, P and I are already identical.
 1243 .sp
 1244 .LP
 1245 The limit set is enforced at `\fBexec\fR` time.
 1246 .sp
 1247 .LP
 1248 To run a non-privilege-aware application in a backward-compatible manner, a

1249 privilege-aware application should start the non-privilege-aware application
 1250 with I=basic.
 1251 .sp
 1252 .LP
 1253 For most privileges, absence of the privilege simply results in a failure. In
 1254 some instances, the absence of a privilege can cause system calls to behave
 1255 differently. In other instances, the removal of a privilege can force a set-uid
 1256 application to seriously malfunction. Privileges of this type are considered
 1257 "unsafe". When a process is lacking any of the unsafe privileges from its limit
 1258 set, the system does not honor the set-uid bit of set-uid root applications.
 1259 The following unsafe privileges have been identified: `\fBproc_setid\fR`,
 1260 `\fBsys_resource\fR` and `\fBproc_audit\fR`.
 1261 .SS "Privilege Escalation"
 1262 .LP
 1263 In certain circumstances, a single privilege could lead to a process gaining
 1264 one or more additional privileges that were not explicitly granted to that
 1265 process. To prevent such an escalation of privileges, the security policy
 1266 requires explicit permission for those additional privileges.
 1267 .sp
 1268 .LP
 1269 Common examples of escalation are those mechanisms that allow modification of
 1270 system resources through "raw" interfaces; for example, changing kernel data
 1271 structures through `\fB/dev/kmem\fR` or changing files through `\fB/dev/dsk/*\fR`.
 1272 Escalation also occurs when a process controls processes with more privileges
 1273 than the controlling process. A special case of this is manipulating or
 1274 creating objects owned by UID 0 or trying to obtain UID 0 using
 1275 `\fBsetuid\fR(2)`. The special treatment of UID 0 is needed because the UID 0
 1276 owns all system configuration files and ordinary file protection mechanisms
 1277 allow processes with UID 0 to modify the system configuration. With appropriate
 1278 file modifications, a given process running with an effective UID of 0 can gain
 1279 all privileges.
 1280 .sp
 1281 .LP
 1282 In situations where a process might obtain UID 0, the security policy requires
 1283 additional privileges, up to the full set of privileges. Such restrictions
 1284 could be relaxed or removed at such time as additional mechanisms for
 1285 protection of system files became available. There are no such mechanisms in
 1286 the current Solaris release.
 1287 .sp
 1288 .LP
 1289 The use of UID 0 processes should be limited as much as possible. They should
 1290 be replaced with programs running under a different UID but with exactly the
 1291 privileges they need.
 1292 .sp
 1293 .LP
 1294 Daemons that never need to `\fBexec\fR` subprocesses should remove the
 1295 `\fBPRIV_PROC_EXEC\fR` privilege from their permitted and limit sets.
 1296 .SS "Assigned Privileges and Safeguards"
 1297 .LP
 1298 When privileges are assigned to a user, the system administrator could give
 1299 that user more powers than intended. The administrator should consider whether
 1300 safeguards are needed. For example, if the `\fBPRIV_PROC_LOCK_MEMORY\fR`
 1301 privilege is given to a user, the administrator should consider setting the
 1302 `\fBproject.max-locked-memory\fR` resource control as well, to prevent that user
 1303 from locking all memory.
 1304 .SS "Privilege Debugging"
 1305 .LP
 1306 When a system call fails with a permission error, it is not always immediately
 1307 obvious what caused the problem. To debug such a problem, you can use a tool
 1308 called `\fBprivilege debugging\fR`. When privilege debugging is enabled for a
 1309 process, the kernel reports missing privileges on the controlling terminal of
 1310 the process. (Enable debugging for a process with the `\fB-D\fR` option of
 1311 `\fBppriv\fR(1)`.) Additionally, the administrator can enable system-wide
 1312 privilege debugging by setting the `\fBsystem\fR(4)` variable `\fBpriv_debug\fR`
 1313 using:
 1314 .sp

```
1315 .in +2
1316 .nf
1317 set priv_debug = 1
1318 .fi
1319 .in -2

1321 .sp
1322 .LP
1323 On a running system, you can use \fBmdb\fR(1) to change this variable.
1324 .SS "Privilege Administration"
1325 .LP
1326 The Solaris Management Console (see \fBsmc\fR(1M)) is the preferred method of
1327 modifying privileges for a command. Use \fBusermod\fR(1M) or \fBsmrole\fR(1M)
1328 to assign privileges to or modify privileges for, respectively, a user or a
1329 role. Use \fBppriv\fR(1) to enumerate the privileges supported on a system and
1330 \fBtruss\fR(1) to determine which privileges a program requires.
1331 .SH SEE ALSO
1332 .LP
1333 \fBmdb\fR(1), \fBppriv\fR(1), \fBadd_drv\fR(1M), \fBifconfig\fR(1M),
1334 \fBblockd\fR(1M), \fBbnfsd\fR(1M), \fBpppd\fR(1M), \fBrem_drv\fR(1M),
1335 \fBsmbd\fR(1M), \fBspptun\fR(1M), \fBupdate_drv\fR(1M), \fBintro\fR(2),
1336 \fBaccess\fR(2), \fBacct\fR(2), \fBacl\fR(2), \fBadjtime\fR(2), \fBaudit\fR(2),
1337 \fBaudition\fR(2), \fBchmod\fR(2), \fBchown\fR(2), \fBchroot\fR(2),
1338 \fBcreat\fR(2), \fBexec\fR(2), \fBfcntl\fR(2), \fBfork\fR(2),
1339 \fBfpathconf\fR(2), \fBgetacct\fR(2), \fBgetpflags\fR(2), \fBgetppriv\fR(2),
1340 \fBgetsid\fR(2), \fBkill\fR(2), \fBlink\fR(2), \fBmemcntl\fR(2),
1341 \fBmknod\fR(2), \fBmount\fR(2), \fBmsgctl\fR(2), \fBnice\fR(2),
1342 \fBntp_adjtime\fR(2), \fBopen\fR(2), \fBp_online\fR(2), \fBpriosctl\fR(2),
1343 \fBpriosctlset\fR(2), \fBprocessor_bind\fR(2), \fBpset_bind\fR(2),
1344 \fBpset_create\fR(2), \fBreadlink\fR(2), \fBresolvepath\fR(2), \fBrmkdir\fR(2),
1345 \fBsemctl\fR(2), \fBsetauid\fR(2), \fBsetegid\fR(2), \fBseteuid\fR(2),
1346 \fBsetgid\fR(2), \fBsetgroups\fR(2), \fBsetpflags\fR(2), \fBsetppriv\fR(2),
1347 \fBsetrctl\fR(2), \fBsetregid\fR(2), \fBsetreuid\fR(2), \fBsetrlimit\fR(2),
1348 \fBsettaskid\fR(2), \fBsetuid\fR(2), \fBshmctl\fR(2), \fBshmget\fR(2),
1349 \fBshmop\fR(2), \fBsigsend\fR(2), \fBstat\fR(2), \fBstatvfs\fR(2),
1350 \fBstime\fR(2), \fBswapctl\fR(2), \fBsysinfo\fR(2), \fBuadmin\fR(2),
1351 \fBulimit\fR(2), \fBumount\fR(2), \fBunlink\fR(2), \fButime\fR(2),
1352 \fButimes\fR(2), \fBbind\fR(3SOCKET), \fBdoor_ucred\fR(3C),
1353 \fBpriv_addset\fR(3C), \fBpriv_set\fR(3C), \fBpriv_getbyname\fR(3C),
1354 \fBpriv_getbynum\fR(3C), \fBpriv_set_to_str\fR(3C), \fBpriv_str_to_set\fR(3C),
1355 \fBsocket\fR(3SOCKET), \fBt_bind\fR(3NSL), \fBtimer_create\fR(3C),
1356 \fBucred_get\fR(3C), \fBexec_attr\fR(4), \fBproc\fR(4), \fBsystem\fR(4),
1357 \fBuser_attr\fR(4), \fBxVM\fR(5), \fBddi_cred\fR(9F), \fBdrv_priv\fR(9F),
1358 \fBpriv_getbyname\fR(9F), \fBpriv_policy\fR(9F), \fBpriv_policy_choice\fR(9F),
1359 \fBpriv_policy_only\fR(9F)
1360 .sp
1361 .LP
1362 \fISystem Administration Guide: Security Services\fR
```

```

*****
3590 Wed Jun 15 19:31:50 2016
new/usr/src/man/man5/security-flags.5
Code review comments from jeffpc
*****
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"
13 .TH "SECURITY-FLAGS" "5" "May 5, 2014"
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_SECFRAGS\fR privilege.
35 .P
36 Currently available features are:
37
38 .sp
39 .ne 2
40 .na
41 Address Space Layout Randomisation (\fBASLR\fR)
42 .ad
43 .RS 1ln
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 \fBDT_SUNW_ASRL\fR
52 dynamic tag in the \fb.dynamic\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
57
58 .sp
59 .ne 2
60 .na

```

```

61 Forbid mappings at NULL (\fbFORBIDNULLMAP\fR)
62 .ad
63 .RS 1ln
64 Mappings with an address of 0 are forbidden, and return EINVAL rather than
65 being honored.
66 .RE
67
68 .sp
69 .ne 2
70 .na
71 Make the userspace stack non-executable (\fbNOEXECSTACK\fR)
72 .ad
73 .RS 1ln
74 The stack will be mapped without executable permission, and attempts to
75 execute it will fault.
76 .RE
77
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.
96
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)

```



```

*****
14783 Wed Jun 15 19:31:51 2016
new/usr/src/man/man5/smf_method.5
Code review comments from jeffpc
*****
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.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\"
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\"
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\fbSMF_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\fbSMF_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\fbSMF_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\fbSMF_zone\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 \fBzone\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\"
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\fbexec(\fIastring\fR)\fR
110 .ad
111 .RS 27n
112 Method executable string.
113 .RE

115 .sp
116 .ne 2
117 .na
118 \fB\fbtimeout_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 .LP
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%\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%\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%\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%\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%\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%\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:\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:\fR\fR
247 .ad
248 .sp .6
249 .RS 4n
250 Always returns \fBEXIT_OK\fR. This token should be used for methods that
251 are required by the restarter but which are unnecessary for the particular
252 service implementation.
253 .RE

255 .SS "Exiting and Exit Status"
256 .LP
257 The required behavior of a start method is to delay exiting until the service
258 instance is ready to answer requests or is otherwise functional.

```

```

259 .sp
260 .LP
261 The following exit status codes are defined in \fB<libscf.h>\fR and in the
262 shell support file.
263 .sp

265 .sp
266 .TS
267 l l l
268 l l l .
269 \fB$SMF_EXIT_OK\fR          \fB0\fR T{
270 Method exited, performing its operation successfully.
271 T}
272 \fB$SMF_EXIT_ERR_FATAL\fR   \fB95\fR T{
273 Method failed fatally and is unrecoverable without administrative intervention.
274 T}
275 \fB$SMF_EXIT_ERR_CONFIG\fR   \fB96\fR T{
276 Unrecoverable configuration error. A common condition that returns this exit sta
277 T}
278 \fB$SMF_EXIT_ERR_NOSMF\fR     \fB99\fR T{
279 Method has been mistakenly invoked outside the \fB$smf\fR(5) facility. Services t
280 T}
281 \fB$SMF_EXIT_ERR_PERM\fR     \fB100\fR T{
282 Method requires a form of permission such as file access, privilege, authorizati
283 T}
284 \fB$SMF_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 restarter 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 .LP
296 Each method can have an independent timeout, given in seconds. The choice of a
297 particular timeout should be based on site expectations for detecting a method
298 failure due to non-responsiveness. Sites with replicated filesystems or other
299 failover resources can elect to lengthen method timeouts from the default.
300 Sites with no remote resources can elect to shorten the timeouts. Method
301 timeout is specified by the \fBtimeout_seconds\fR property.
302 .sp
303 .LP
304 If you specify \fBtimeout_seconds\fR for a method, it declares to the
305 restarter that there is no timeout for the service. This setting is not
306 preferred, but is available for services that absolutely require it.
307 .sp
308 .LP
309 \fB-1 timeout_seconds\fR is also accepted, but is a deprecated specification.
310 .SS "Shell Programming Support"
311 .LP
312 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 \fB$smf_present()\fR shell function is
320 provided. If the \fB$smf\fR(5) facility is not available, \fB$smf_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     ....
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 .LP
345 The service management facility offers a common mechanism set the context in
346 which the \fBfork\fR(2)-\fBexec\fR(2) model services execute.
347 .sp
348 .LP
349 The desired method context should be provided by the service developer. All
350 service instances should run with the lowest level of privileges possible to
351 limit potential security compromises.
352 .sp
353 .LP
354 A method context can contain the following properties:
355 .sp
356 .ne 2
357 .na
358 \fB$buse_profile\fR
359 .ad
360 .sp .6
361 .RS 4n
362 A boolean that specifies whether the profile should be used instead of the
363 \fB$buser\fR, \fB$bgroupp\fR, \fB$bprivileges\fR, and \fB$blimit_privileges\fR
364 properties.
365 .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$bprofile\fR
382 .ad
383 .sp .6
384 .RS 4n
385 The name of an RBAC (role-based access control) profile which, along with the
386 method executable, identifies an entry in \fB$bexec_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 \fB\fBcorefile_pattern\fR\fR
478 .ad
479 .sp .6
480 .RS 4n
481 An optional string that specifies the corefile pattern to use for the service,
482 as per \fBcoreadm\fR(1M). Most restarters supply a default. Setting this
483 property overrides local customizations to the global core pattern.
484 .RE

486 .sp
487 .ne 2
488 .na
489 \fB\fBproject\fR\fR
490 .ad
491 .sp .6
492 .RS 4n
493 The project ID in numeric or text form. \fB:default\fR can be used as a token
494 to indicate a project identified by \fBgetdefaultproj\fR(3PROJECT) for the user
495 whose \fBuid\fR is used to launch the method.
496 .RE

498 .sp
499 .ne 2
500 .na
501 \fB\fBresource_pool\fR\fR
502 .ad
503 .sp .6
504 .RS 4n
505 The resource pool name on which to launch the method. \fB:default\fR can be
506 used as a token to indicate the pool specified in the \fBproject\fR(4) entry
507 given in the \fBproject\fR attribute above.
508 .RE

510 .sp
511 .LP
512 The method context can be set for the entire service instance by specifying a
513 \fBmethod_context\fR property group for the service or instance. A method might
514 override the instance method context by providing the method context properties
515 on the method property group.
516 .sp
517 .LP
518 Invalid method context settings always lead to failure of the method, with the
519 exception of invalid environment variables that issue warnings.
520 .sp
521 .LP
522 In addition to the context defined above, many \fBfork\fR(2)-\fBexec\fR(2)

```

```
523 model restarters also use the following conventions when invoking executables
524 as methods:
525 .sp
526 .ne 2
527 .na
528 \fBArgument array\fR
529 .ad
530 .sp .6
531 .RS 4n
532 The arguments in \fBargv[]\fR are set consistently with the result \fB/bin/sh
533 -c\fR of the \fBexec\fR string.
534 .RE

536 .sp
537 .ne 2
538 .na
539 \fBFile descriptors\fR
540 .ad
541 .sp .6
542 .RS 4n
543 File descriptor \fB0\fR is \fB/dev/null\fR. File descriptors \fB1\fR and
544 \fB2\fR are recommended to be a per-service log file.
545 .RE

547 .SH FILES
548 .ne 2
549 .na
550 \fB\fB/lib/svc/share/smf_include.sh\fR\fR
551 .ad
552 .sp .6
553 .RS 4n
554 Definitions of exit status values.
555 .RE

557 .sp
558 .ne 2
559 .na
560 \fB\fB/usr/include/libscf.h\fR\fR
561 .ad
562 .sp .6
563 .RS 4n
564 Definitions of exit status codes.
565 .RE

567 .SH SEE ALSO
568 .LP
569 \fBzone\fR(1), \fBcoreadm\fR(1M), \fBbinetd\fR(1M), \fBsvccfg\fR(1M),
570 \fBsvcs.startd\fR(1M), \fBexec\fR(2), \fBfork\fR(2),
571 \fBgetdefaultproj\fR(3PROJECT), \fBexec_attr\fR(4), \fBproject\fR(4),
572 \fBservice_bundle\fR(4), \fBattributes\fR(5), \fBprivileges\fR(5),
573 \fBrbac\fR(5), \fBsmf\fR(5), \fBsmf_bootstrap\fR(5), \fBzones\fR(5),
574 \fBsecurity-flags\fR(5)
575 .SH NOTES
576 .LP
577 The present version of \fBsmf\fR(5) does not support multiple repositories.
578 .sp
579 .LP
580 When a service is configured to be started as root but with privileges
581 different from \fBlimit_privileges\fR, the resulting process is privilege
582 aware. This can be surprising to developers who expect \fBsetuid(<non-zero
583 UID>)\fR to reduce privileges to basic or less.
```

new/usr/src/test/os-tests/tests/secflags/Makefile

1

1454 Wed Jun 15 19:31:53 2016

new/usr/src/test/os-tests/tests/secflags/Makefile

Code review comments from jeffpc

```
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     $(LINK.c) -m32 stacky.o -o $@ $(LDLIBS)
51     $(POST_PROCESS)
```

```
52 secflags_syscall: secflags_syscall.c
53     $(LINK.c) secflags_syscall.c -o $@ $(LDLIBS)
54     $(LINK.c) -m32 secflags_syscall.c -o $@ $(LDLIBS)
55     $(POST_PROCESS)
```

```
56 all: $(PROG)
```

```
58 install: all $(CMDS)
```

new/usr/src/test/os-tests/tests/secflags/Makefile

2

```
60 lint:
```

```
62 clobber: clean
63     -$(RM) $(PROG)
```

```
65 clean:
```

```
67 $(CMDS): $(TESTDIR) $(PROG)
```

```
69 $(TESTDIR):
70     $(INS.dir)
```

```
72 $(TESTDIR)/%: %
73     $(INS.file)
```

new/usr/src/test/os-tests/tests/secflags/secflags_aslr.sh

1

1381 Wed Jun 15 19:31:54 2016

new/usr/src/test/os-tests/tests/secflags/secflags_aslr.sh

Code review comments from jeffpc

```
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.
```

```
14 # Copyright 2015, Richard Lowe.
```

```
16 # Verify that aslr messes things up, by comparing the mappings of 2 identical
```

```
17 # processes
```

```
16 # Verify that aslr musses things up, by comparing the mappings of 2 identical pr
```

```
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 }
```

```
_____unchanged_portion_omitted_
```

```
new/usr/src/test/os-tests/tests/secflags/secflags_elfdump.sh
```

1

```
*****
```

```
1754 Wed Jun 15 19:31:54 2016
```

```
new/usr/src/test/os-tests/tests/secflags/secflags_elfdump.sh
```

```
Code review comments from jeffpc
```

```
*****
```

```
unchanged portion omitted
```

```
31 trap cleanup EXIT
```

```
33 ## gcore-produced core
```

```
34 gcore $pid >/dev/null
```

```
36 cat > gcore-expected.$$ <<EOF
```

```
37 namesz: 0x5
```

```
38 descsz: 0x28
```

```
38 descsz: 0x14
```

```
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
```

```
50 /usr/bin/elfdump -n core.${pid} | grep -B5 -A5 prsecflags_t > gcore-output.$$
```

```
52 if ! diff -u gcore-expected.$$ gcore-output.$$; then
```

```
53 exit 1;
```

```
54 fi
```

```
56 ## kernel-produced core
```

```
57 kill -SEGV $pid
```

```
59 cat > core-expected.$$ <<EOF
```

```
60 namesz: 0x5
```

```
61 descsz: 0x28
```

```
61 descsz: 0x14
```

```
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
```

```
73 /usr/bin/elfdump -n core | grep -B5 -A5 prsecflags_t > core-output.$$
```

```
75 if ! diff -u core-expected.$$ core-output.$$; then
```

```
76 exit 1;
```

```
77 fi
```

```
79 exit 0
```


new/usr/src/test/os-tests/tests/secflags/secflags_psecflags.sh

1

3711 Wed Jun 15 19:31:55 2016

new/usr/src/test/os-tests/tests/secflags/secflags_psecflags.sh

Code review comments from jeffpc

unchanged portion omitted

```
49 self_add() {
50     echo "Add (self)"
51     /usr/bin/psecflags -s current,noexecstack $$
52     cat > expected <<EOF
53     I:    aslr,noexecstack
53     I:    aslr, noexecstack
54 EOF
```

```
56     /usr/bin/psecflags $$ | grep I: > output
57     diff -u expected output || exit 1
58 }
```

unchanged portion omitted

```
104 child_add() {
105     echo "Add (child)"
107     typeset pid;
109     /usr/bin/psecflags -s aslr $$
110     /usr/bin/psecflags -s current,noexecstack -e sleep 10000 &
111     pid=$!
112     cat > expected <<EOF
113     E:    aslr,noexecstack
114     I:    aslr,noexecstack
113     E:    aslr, noexecstack
114     I:    aslr, noexecstack
115 EOF
116     /usr/bin/psecflags $pid | grep '[IE]:' > output
117     kill $pid
118     /usr/bin/psecflags -s none $$
119     diff -u expected output || exit 1
120 }
```

unchanged portion omitted

new/usr/src/test/os-tests/tests/secflags/secflags_zonecfg.sh

1

3717 Wed Jun 15 19:31:56 2016

new/usr/src/test/os-tests/tests/secflags/secflags_zonecfg.sh

Code review comments from jeffpc

```
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.
```

```
14 # Copyright 2015, Richard Lowe.
```

```
16 # Verify that zones can be configured with security-flags
17 LC_ALL=C # Collation is important
```

```
19 expect_success() {
20     name=$1
```

```
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
```

```
29     r=$?
```

```
31     zonecfg -z ${name}.$$ delete -F
```

```
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 }
```

unchanged_portion_omitted

```

*****
3364 Wed Jun 15 19:31:57 2016
new/usr/src/uts/common/exec/elf/elf_impl.h
Code review comments from jeffpc
*****
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 _ELF_ELF_IMPL_H
27 #define _ELF_ELF_IMPL_H

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

31 #endif /* ! codereview */
32 #ifdef __cplusplus
33 extern "C" {
34 #endif

36 #if !defined(_LP64) || defined(_ELF32_COMPAT)

38 /*
39  * Definitions for ELF32, native 32-bit or 32-bit compatibility mode.
40  */
41 #define ELFCLASS ELFCLASS32
42 typedef unsigned int aux_val_t;
43 typedef auxv32_t aux_entry_t;

45 #define USR_LIB_RTLD "/usr/lib/ld.so.1"

47 #else /* !_LP64 || _ELF32_COMPAT */

49 /*
50  * Definitions for native 64-bit ELF
51  */
52 #define ELFCLASS ELFCLASS64
53 typedef unsigned long aux_val_t;
54 typedef auxv_t aux_entry_t;

56 /* put defines for 64-bit architectures here */
57 #if defined(__sparcv9)
58 #define USR_LIB_RTLD "/usr/lib/sparcv9/ld.so.1"
59 #endif

61 #if defined(__amd64)

```

```

62 #define USR_LIB_RTLD "/usr/lib/amd64/ld.so.1"
63 #endif

65 #endif /* !_LP64 || _ELF32_COMPAT */

67 /*
68  * Start of an ELF Note.
69  */
70 typedef struct {
71     Nhdr nhdr;
72     char name[8];
73 } Note;

75 #ifdef _ELF32_COMPAT
76 /*
77  * These are defined only for the 32-bit compatibility
78  * compilation mode of the 64-bit kernel.
79  */
80 #define elfexec elf32exec
81 #define elfnote elf32note
82 #define elfcore elf32core
83 #define mapexec_brand mapexec32_brand
84 #define setup_note_header setup_note_header32
85 #define write_elfnotes write_elfnotes32
86 #define setup_old_note_header setup_old_note_header32
87 #define write_old_elfnotes write_old_elfnotes32

89 #if defined(__sparc)
90 #define gwindows_t gwindows32_t
91 #define rwindow rwindow32
92 #endif

94 #define psinfo_t psinfo32_t
95 #define pstatus_t pstatus32_t
96 #define lwpsinfo_t lwpsinfo32_t
97 #define lwpstatus_t lwpstatus32_t

99 #define prgetpsinfo prgetpsinfo32
100 #define prgetstatus prgetstatus32
101 #define prgetlwpsinfo prgetlwpsinfo32
102 #define prgetlwpstatus prgetlwpstatus32
103 #define prgetwindows prgetwindows32

105 #define prpsinfo_t prpsinfo32_t
106 #define prstatus_t prstatus32_t
107 #if defined(prfpregset_t)
108 #undef prfpregset_t
109 #endif
110 #define prfpregset_t prfpregset32_t

112 #define oprgetstatus oprgetstatus32
113 #define oprgetpsinfo oprgetpsinfo32
114 #define prgetprfpregs prgetprfpregs32

116 #endif /* _ELF32_COMPAT */

118 extern int elfnote(vnode_t *, offset_t *, int, int, void *, rlim64_t, cred_t *);
119 extern void setup_old_note_header(Phdr *, proc_t *);
120 extern void setup_note_header(Phdr *, proc_t *);

122 extern int write_old_elfnotes(proc_t *, int, vnode_t *, offset_t,
123     rlim64_t, cred_t *);

125 extern int write_elfnotes(proc_t *, int, vnode_t *, offset_t,
126     rlim64_t, cred_t *, core_content_t);

```

new/usr/src/uts/common/exec/elf/elf_impl.h

3

```
128 #ifdef __cplusplus
129 }
130 #endif
132 #endif /* _ELF_ELF_IMPL_H */
```

```

*****
37198 Wed Jun 15 19:31:58 2016
new/usr/src/uts/common/os/fork.c
Code review comments from jeffpc
*****
_____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_splock, 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 prfind_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 = NULLVP;
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         zcomm_err(zoneid, CE_NOTE,
1053             "out of per-user processes for uid %d", ruid);
1054         goto bad;
1055     }
1056
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);
1064
1065     cp->p_next = practive;
1066     practive->p_prev = cp;
1067     practive = cp;
1068
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;
1094     bcopy(&pp->p_secflags, &cp->p_secflags, sizeof (psecflags_t));
1095
1096     /*
1097     * Initialize watchpoint structures
1098     */
1099     avl_create(&cp->p_warea, wa_compare, sizeof (struct watched_area),
1100         offsetof(struct watched_area, wa_link));
1101
1102     /*
1103     * Initialize immediate resource control values.
1104     */
1105     cp->p_stk_ctl = pp->p_stk_ctl;
1106     cp->p_fsz_ctl = pp->p_fsz_ctl;
1107     cp->p_vmem_ctl = pp->p_vmem_ctl;
1108     cp->p_fno_ctl = pp->p_fno_ctl;
1109
1110     /*
1111     * Link up to parent-child-sibling chain. No need to lock
1112     * in general since only a call to freeproc() (done by the
1113     * same parent as newproc()) diddles with the child chain.
1114     */

```

```

1115     cp->p_sibling = pp->p_child;
1116     if (pp->p_child)
1117         pp->p_child->p_sibling = cp;
1118
1119     cp->p_parent = pp;
1120     pp->p_child = cp;
1121
1122     cp->p_child_ns = NULL;
1123     cp->p_sibling_ns = NULL;
1124
1125     cp->p_nextorph = pp->p_orphan;
1126     cp->p_nextofkin = pp;
1127     pp->p_orphan = cp;
1128
1129     /*
1130     * Inherit profiling state; do not inherit REALPROF profiling state.
1131     */
1132     cp->p_prof = pp->p_prof;
1133     cp->p_rprof_cyclic = CYCLIC_NONE;
1134
1135     /*
1136     * Inherit pool pointer from the parent. Kernel processes are
1137     * always bound to the default pool.
1138     */
1139     mutex_enter(&pp->p_lock);
1140     if (flags & GETPROC_KERNEL) {
1141         cp->p_pool = pool_default;
1142         cp->p_flag |= SSYS;
1143     } else {
1144         cp->p_pool = pp->p_pool;
1145     }
1146     atomic_inc_32(&cp->p_pool->pool_ref);
1147     mutex_exit(&pp->p_lock);
1148
1149     /*
1150     * Add the child process to the current task. Kernel processes
1151     * are always attached to task0.
1152     */
1153     mutex_enter(&cp->p_lock);
1154     if (flags & GETPROC_KERNEL)
1155         task_attach(task0p, cp);
1156     else
1157         task_attach(pp->p_task, cp);
1158     mutex_exit(&cp->p_lock);
1159     mutex_exit(&pidlock);
1160
1161     avl_create(&cp->p_ct_held, contract_compar, sizeof (contract_t),
1162         offsetof(contract_t, ct_ctlist));
1163
1164     /*
1165     * Duplicate any audit information kept in the process table
1166     */
1167     if (audit_active) /* copy audit data to cp */
1168         audit_newproc(cp);
1169
1170     crhold(cp->p_cred = cr);
1171
1172     /*
1173     * Bump up the counts on the file structures pointed at by the
1174     * parent's file table since the child will point at them too.
1175     */
1176     fcnt_add(P_FINFO(pp), 1);
1177
1178     if (PTOU(pp)->u_cdir) {
1179         VN_HOLD(PTOU(pp)->u_cdir);
1180     } else {

```

```

1181         ASSERT(pp == &p0);
1182         /*
1183          * We must be at or before vfs_mountroot(); it will take care of
1184          * assigning our current directory.
1185          */
1186     }
1187     if (PTOU(pp)->u_rdir)
1188         VN_HOLD(PTOU(pp)->u_rdir);
1189     if (PTOU(pp)->u_cwd)
1190         refstr_hold(PTOU(pp)->u_cwd);
1191
1192     /*
1193      * copy the parent's uarea.
1194      */
1195     uarea = PTOU(cp);
1196     bcopy(PTOU(pp), uarea, sizeof (*uarea));
1197     flist_fork(P_FINFO(pp), P_FINFO(cp));
1198
1199     getthretime(&uarea->u_start);
1200     uarea->u_ticks = ddi_get_lbolt();
1201     uarea->u_mem = rm_asrss(pp->p_as);
1202     uarea->u_acflag = AFORK;
1203
1204     /*
1205      * If inherit-on-fork, copy /proc tracing flags to child.
1206      */
1207     if ((pp->p_proc_flag & P_PR_FORK) != 0) {
1208         cp->p_proc_flag |= pp->p_proc_flag & (P_PR_TRACE|P_PR_FORK);
1209         cp->p_sigmask = pp->p_sigmask;
1210         cp->p_fltmask = pp->p_fltmask;
1211     } else {
1212         sigemptyset(&cp->p_sigmask);
1213         premtypset(&cp->p_fltmask);
1214         uarea->u_systrap = 0;
1215         premtypset(&uarea->u_entrymask);
1216         premtypset(&uarea->u_exitmask);
1217     }
1218     /*
1219      * If microstate accounting is being inherited, mark child
1220      */
1221     if ((pp->p_flag & SMSFORK) != 0)
1222         cp->p_flag |= pp->p_flag & (SMSFORK|SMSACCT);
1223
1224     /*
1225      * Inherit fixalignment flag from the parent
1226      */
1227     cp->p_fixalignment = pp->p_fixalignment;
1228
1229     *cpp = cp;
1230     return (0);
1231
1232 bad:
1233     ASSERT(MUTEX_NOT_HELD(&pidlock));
1234
1235     mutex_destroy(&cp->p_crlock);
1236     mutex_destroy(&cp->p_plock);
1237 #if defined(__x86)
1238     mutex_destroy(&cp->p_ldtlock);
1239 #endif
1240     if (newpid != -1) {
1241         proc_entry_free(cp->p_pidp);
1242         (void) pid_rele(cp->p_pidp);
1243     }
1244     kmem_cache_free(process_cache, cp);
1245
1246     mutex_enter(&zone->zone_nlwps_lock);

```

```

1247         task->tk_nprocs--;
1248         proj->kpj_nprocs--;
1249         zone->zone_nprocs--;
1250         mutex_exit(&zone->zone_nlwps_lock);
1251         atomic_inc_32(&zone->zone_ffnoprocs);
1252
1253 punish:
1254     /*
1255      * We most likely got into this situation because some process is
1256      * forking out of control. As punishment, put it to sleep for a
1257      * bit so it can't eat the machine alive. Sleep interval is chosen
1258      * to allow no more than one fork failure per cpu per clock tick
1259      * on average (yes, I just made this up). This has two desirable
1260      * properties: (1) it sets a constant limit on the fork failure
1261      * rate, and (2) the busier the system is, the harsher the penalty
1262      * for abusing it becomes.
1263      */
1264     INCR_COUNT(&fork_fail_pending, &pidlock);
1265     delay(fork_fail_pending / ncpus + 1);
1266     DECR_COUNT(&fork_fail_pending, &pidlock);
1267
1268     return (-1); /* out of memory or proc slots */
1269 }

```

unchanged portion omitted

```

*****
35236 Wed Jun 15 19:31:59 2016
new/usr/src/uts/common/sys/procfs.h
Code review comments from jeffpc
*****
_____unchanged_portion_omitted_____

402 #define PRSECFLAGS_VERSION_1 1
403 #define PRSECFLAGS_VERSION_CURRENT PRSECFLAGS_VERSION_1
404 typedef struct prsecflags {
405     uint32_t pr_version;
406     char pr_pad[4];
407 #endif /* ! codereview */
408     secflags_t pr_effective;
409     secflags_t pr_inherit;
410     secflags_t pr_lower;
411     secflags_t pr_upper;
412 } prsecflags_t;

414 /*
415  * Watchpoint interface. PCWATCH and /proc/<pid>/watch
416  */
417 typedef struct prwatch {
418     uintptr_t pr_vaddr; /* virtual address of watched area */
419     size_t pr_size; /* size of watched area in bytes */
420     int pr_wflags; /* watch type flags */
421     int pr_pad;
422 } prwatch_t;

424 /* pr_wflags */
425 #define WA_READ 0x04 /* trap on read access */
426 #define WA_WRITE 0x02 /* trap on write access */
427 #define WA_EXEC 0x01 /* trap on execute access */
428 #define WA_TRAPAFTER 0x08 /* trap after instruction completes */

430 /*
431  * PCREAD/PCWRITE I/O interface.
432  */
433 typedef struct priovec {
434     void *pio_base; /* buffer in controlling process */
435     size_t pio_len; /* size of read/write request */
436     off_t pio_offset; /* virtual address in target process */
437 } priovec_t;

439 /*
440  * Resource usage. /proc/<pid>/usage /proc/<pid>/lwp/<lwpid>/lwpusage
441  */
442 typedef struct prusage {
443     id_t pr_lwpid; /* lwp id. 0: process or defunct */
444     int pr_count; /* number of contributing lwps */
445     timestruc_t pr_tstamp; /* current time stamp */
446     timestruc_t pr_create; /* process/lwp creation time stamp */
447     timestruc_t pr_term; /* process/lwp termination time stamp */
448     timestruc_t pr_rtime; /* total lwp real (elapsed) time */
449     timestruc_t pr_utime; /* user level cpu time */
450     timestruc_t pr_stime; /* system call cpu time */
451     timestruc_t pr_ttime; /* other system trap cpu time */
452     timestruc_t pr_tftime; /* text page fault sleep time */
453     timestruc_t pr_dftime; /* data page fault sleep time */
454     timestruc_t pr_kftime; /* kernel page fault sleep time */
455     timestruc_t pr_ltime; /* user lock wait sleep time */
456     timestruc_t pr_slptime; /* all other sleep time */
457     timestruc_t pr_wtime; /* wait-cpu (latency) time */
458     timestruc_t pr_stoptime; /* stopped time */
459     timestruc_t filltime[6]; /* filler for future expansion */
460     ulong_t pr_minf; /* minor page faults */

```

```

461     ulong_t pr_majf; /* major page faults */
462     ulong_t pr_nswap; /* swaps */
463     ulong_t pr_inblk; /* input blocks */
464     ulong_t pr_oublk; /* output blocks */
465     ulong_t pr_msnd; /* messages sent */
466     ulong_t pr_mrcv; /* messages received */
467     ulong_t pr_sigs; /* signals received */
468     ulong_t pr_vctx; /* voluntary context switches */
469     ulong_t pr_ictx; /* involuntary context switches */
470     ulong_t pr_sysc; /* system calls */
471     ulong_t pr_ioch; /* chars read and written */
472     ulong_t filler[10]; /* filler for future expansion */
473 } prusage_t;

475 /*
476  * Page data file. /proc/<pid>/pagedata
477  */

479 /* page data file header */
480 typedef struct prpageheader {
481     timestruc_t pr_tstamp; /* real time stamp */
482     long pr_nmap; /* number of address space mappings */
483     long pr_npage; /* total number of pages */
484 } prpageheader_t;

486 /* page data mapping header */
487 typedef struct prsmmap {
488     uintptr_t pr_vaddr; /* virtual address of mapping */
489     size_t pr_npage; /* number of pages in mapping */
490     char pr_mapname[PRMAPSZ]; /* name in /proc/<pid>/object */
491     offset_t pr_offset; /* offset into mapped object, if any */
492     int pr_mflags; /* protection and attribute flags */
493     int pr_pagesize; /* pagesize (bytes) for this mapping */
494     int pr_shmid; /* SysV shmid, -1 if not SysV shared memory */
495     int pr_filler[1]; /* filler for future expansion */
496 } prsmmap_t;

498 /*
499  * pr_npage bytes (plus 0-7 null bytes to round up to an 8-byte boundary)
500  * follow each mapping header, each containing zero or more of these flags.
501  */
502 #define PG_REFERENCED 0x02 /* page referenced since last read */
503 #define PG_MODIFIED 0x01 /* page modified since last read */
504 #define PG_HWMAPPED 0x04 /* page is present and mapped */

506 /*
507  * Open files. Only in core files (for now). Note that we'd like to use
508  * the stat or stat64 structure, but both of these structures are unfortunately
509  * not consistent between 32 and 64 bit modes. To keep our lives simpler, we
510  * just define our own structure with types that are not sensitive to this
511  * difference. Also, it turns out that pfiles omits a lot of info from the
512  * struct stat (e.g. times, device sizes, etc.) so we don't bother adding those
513  * here.
514  */
515 typedef struct prfdinfo {
516     int pr_fd;
517     mode_t pr_mode;

519     uid_t pr_uid;
520     gid_t pr_gid;

522     major_t pr_major; /* think stat.st_dev */
523     minor_t pr_minor;

525     major_t pr_rmajor; /* think stat.st_rdev */
526     minor_t pr_rminor;

```



```

528     ino64_t      pr_ino;
529     off64_t      pr_offset;
530     off64_t      pr_size;

532     int          pr_fileflags; /* fcntl(F_GETXFL), etc */
533     int          pr_fdflags; /* fcntl(F_GETFD), etc. */

535     char         pr_path[MAXPATHLEN];
536 } prfdinfo_t;

538 /*
539  * Header for /proc/<pid>/lstatus /proc/<pid>/lpsinfo /proc/<pid>/lusage
540  */
541 typedef struct prheader {
542     long         pr_nent;      /* number of entries */
543     long         pr_entsize;   /* size of each entry, in bytes */
544 } prheader_t;

546 /*
547  * Macros for manipulating sets of flags.
548  * sp must be a pointer to one of sigset_t, fltset_t, or sysset_t.
549  * flag must be a member of the enumeration corresponding to *sp.
550  */

552 /* turn on all flags in set */
553 #define prfillset(sp) \
554     { register int _i = sizeof (*(sp))/sizeof (uint32_t); \
555       while (_i) ((uint32_t *) (sp))[--_i] = (uint32_t)0xFFFFFFFF; }

557 /* turn off all flags in set */
558 #define premtysset(sp) \
559     { register int _i = sizeof (*(sp))/sizeof (uint32_t); \
560       while (_i) ((uint32_t *) (sp))[--_i] = (uint32_t)0; }

562 /* turn on specified flag in set */
563 #define praddset(sp, flag) \
564     ((void)((unsigned)((flag)-1) < 32*sizeof (*(sp))/sizeof (uint32_t)) ? \
565     (((uint32_t *) (sp))[(flag)-1/32] |= (1U<<(((flag)-1)%32))) : 0))

567 /* turn off specified flag in set */
568 #define prdelset(sp, flag) \
569     ((void)((unsigned)((flag)-1) < 32*sizeof (*(sp))/sizeof (uint32_t)) ? \
570     (((uint32_t *) (sp))[(flag)-1/32] &= ~(1U<<(((flag)-1)%32))) : 0))

572 /* query: != 0 iff flag is turned on in set */
573 #define prismember(sp, flag) \
574     (((unsigned)((flag)-1) < 32*sizeof (*(sp))/sizeof (uint32_t)) && \
575     (((uint32_t *) (sp))[(flag)-1/32] & (1U<<(((flag)-1)%32))))

577 #if defined(_SYSCALL32)

579 /*
580  * dev32_t version of PRNODEV
581  */
582 #define PRNODEV32 (dev32_t)(-1)

584 /*
585  * Kernel view of /proc structures for _ILP32 programs.
586  */

588 /*
589  * _ILP32 lwp status file. /proc/<pid>/lwp/<lwpid>/lwpstatus
590  */
591 typedef struct lwpstatus32 {
592     int          pr_flags;      /* flags */

```

```

593     id32_t      pr_lwpid;      /* specific lwp identifier */
594     short       pr_why;        /* reason for lwp stop, if stopped */
595     short       pr_what;      /* more detailed reason */
596     short       pr_cursig;    /* current signal, if any */
597     short       pr_pad1;
598     siginfo32_t pr_info;      /* info associated with signal or fault */
599     sigset_t    pr_lwppend;   /* set of signals pending to the lwp */
600     sigset_t    pr_lwphold;   /* set of signals blocked by the lwp */
601     struct sigaction32 pr_action; /* signal action for current signal */
602     stack32_t  pr_altstack;   /* alternate signal stack info */
603     caddr32_t  pr_oldcontext; /* address of previous ucontext */
604     short       pr_syscall;   /* system call number (if in syscall) */
605     short       pr_nsysarg;   /* number of arguments to this syscall */
606     int         pr_errno;     /* errno for failed syscall, 0 if successful */
607     int32_t     pr_sysarg[PRSYSARGS]; /* arguments to this syscall */
608     int32_t     pr_rvall;     /* primary syscall return value */
609     int32_t     pr_rval2;     /* second syscall return value, if any */
610     char        pr_clname[PRCLSZ]; /* scheduling class name */
611     timestruc32_t pr_tstamp; /* real-time time stamp of stop */
612     timestruc32_t pr_utime; /* lwp user cpu time */
613     timestruc32_t pr_stime; /* lwp system cpu time */
614     int         pr_filler[11 - 2 * sizeof (timestruc32_t) / sizeof (int)];
615     int         pr_errpriv;   /* missing privilege */
616     caddr32_t  pr_ustack;    /* address of stack boundary data (stack32_t) */
617     uint32_t   pr_instr;     /* current instruction */
618     prgregset32_t pr_reg; /* general registers */
619     prfpregset32_t pr_fpreg; /* floating-point registers */
620 } lwpstatus32_t;

622 /*
623  * _ILP32 process status file. /proc/<pid>/status
624  */
625 typedef struct pstatus32 {
626     int         pr_flags;      /* flags */
627     int         pr_nlwp;      /* number of active lwps in the process */
628     pid32_t     pr_pid;       /* process id */
629     pid32_t     pr_ppid;      /* parent process id */
630     pid32_t     pr_pgid;      /* process group id */
631     pid32_t     pr_sid;       /* session id */
632     id32_t     pr_aslwpid;    /* historical; now always zero */
633     id32_t     pr_agentid;    /* lwp id of the /proc agent lwp, if any */
634     sigset_t    pr_sigpend;   /* set of process pending signals */
635     caddr32_t  pr_brkbase;    /* address of the process heap */
636     size32_t   pr_brksize;    /* size of the process heap, in bytes */
637     caddr32_t  pr_stkbase;    /* address of the process stack */
638     size32_t   pr_stksize;    /* size of the process stack, in bytes */
639     timestruc32_t pr_utime; /* process user cpu time */
640     timestruc32_t pr_stime; /* process system cpu time */
641     timestruc32_t pr_cutime; /* sum of children's user times */
642     timestruc32_t pr_cstime; /* sum of children's system times */
643     sigset_t    pr_sigtrace; /* set of traced signals */
644     fltset_t    pr_fltrtrace; /* set of traced faults */
645     sysset_t    pr_sysentry; /* set of system calls traced on entry */
646     sysset_t    pr_sysexit; /* set of system calls traced on exit */
647     char        pr_dmodel;    /* data model of the process */
648     char        pr_pad[3];
649     id32_t     pr_taskid;     /* task id */
650     id32_t     pr_projid;     /* project id */
651     int        pr_nzomb;     /* number of zombie lwps in the process */
652     id32_t     pr_zoneid;     /* zone id */
653     int        pr_filler[15]; /* reserved for future use */
654     lwpstatus32_t pr_lwp; /* status of the representative lwp */
655 } pstatus32_t;

657 /*
658  * _ILP32 lwp ps(1) information file. /proc/<pid>/lwp/<lwpid>/lwpsinfo

```

```

659 */
660 typedef struct lwpsinfo32 {
661     int     pr_flag;          /* lwp flags */
662     id32_t  pr_lwpid;        /* lwp id */
663     caddr32_t pr_addr;      /* internal address of lwp */
664     caddr32_t pr_wchan;     /* wait addr for sleeping lwp */
665     char    pr_stype;        /* synchronization event type */
666     char    pr_state;        /* numeric lwp state */
667     char    pr_sname;        /* printable character for pr_state */
668     char    pr_nice;         /* nice for cpu usage */
669     short   pr_syscall;     /* system call number (if in syscall) */
670     char    pr_oldpri;       /* pre-SVR4, low value is high priority */
671     char    pr_cpu;         /* pre-SVR4, cpu usage for scheduling */
672     int     pr_pri;         /* priority, high value is high priority */
673     /* The following percent number is a 16-bit binary */
674     /* fraction [0 .. 1] with the binary point to the */
675     /* right of the high-order bit (1.0 == 0x8000) */
676     ushort_t pr_pctcpu;     /* % of recent cpu time used by this lwp */
677     ushort_t pr_pad;
678     timestruc32_t pr_start; /* lwp start time, from the epoch */
679     timestruc32_t pr_time;  /* usr+sys cpu time for this lwp */
680     char    pr_clname[PRCLSZ]; /* scheduling class name */
681     char    pr_name[PRFNSZ]; /* name of system lwp */
682     processorid_t pr_onpro; /* processor which last ran this lwp */
683     processorid_t pr_bindpro; /* processor to which lwp is bound */
684     psetid_t pr_bindpset; /* processor set to which lwp is bound */
685     int     pr_lgrp;        /* lwp home lgroup */
686     int     pr_filler[4];   /* reserved for future use */
687 } lwpsinfo32_t;

689 /*
690 * _ILP32 process ps(1) information file. /proc/<pid>/psinfo
691 */
692 typedef struct psinfo32 {
693     int     pr_flag;        /* process flags */
694     int     pr_nlwp;        /* number of active lwps in the process */
695     pid32_t pr_pid;         /* unique process id */
696     pid32_t pr_ppid;        /* process id of parent */
697     pid32_t pr_pgid;        /* pid of process group leader */
698     pid32_t pr_sid;         /* session id */
699     uid32_t pr_uid;         /* real user id */
700     uid32_t pr_euid;        /* effective user id */
701     gid32_t pr_gid;         /* real group id */
702     gid32_t pr_egid;        /* effective group id */
703     caddr32_t pr_addr;      /* address of process */
704     size32_t pr_size;        /* size of process image in Kbytes */
705     size32_t pr_rssize;     /* resident set size in Kbytes */
706     size32_t pr_pad1;
707     dev32_t pr_ttydev;      /* controlling tty device (or PRNODEV) */
708     ushort_t pr_pctcpu;     /* % of recent cpu time used by all lwps */
709     ushort_t pr_pctmem;     /* % of system memory used by process */
710     timestruc32_t pr_start; /* process start time, from the epoch */
711     timestruc32_t pr_time;  /* usr+sys cpu time for this process */
712     timestruc32_t pr_ctime; /* usr+sys cpu time for reaped children */
713     char    pr_fname[PRFNSZ]; /* name of execed file */
714     char    pr_psargs[PRARGSZ]; /* initial characters of arg list */
715     int     pr_wstat;        /* if zombie, the wait() status */
716     int     pr_argc;         /* initial argument count */
717     caddr32_t pr_argv;      /* address of initial argument vector */
718     caddr32_t pr_envp;      /* address of initial environment vector */
719     char    pr_dmodel;      /* data model of the process */
720     char    pr_pad2[3];
721     id32_t  pr_taskid;       /* task id */
722     id32_t  pr_projid;       /* project id */
723     int     pr_nzomb;        /* number of zombie lwps in the process */
724     id32_t  pr_poolid;       /* pool id */

```

```

725     id32_t  pr_zoneid;      /* zone id */
726     id32_t  pr_contract;    /* process contract */
727     int     pr_filler[1];   /* reserved for future use */
728     lwpsinfo32_t pr_lwp;    /* information for representative lwp */
729 } psinfo32_t;

731 /*
732 * _ILP32 Memory-management interface. /proc/<pid>/map /proc/<pid>/rmap
733 */
734 typedef struct prmap32 {
735     caddr32_t pr_vaddr;     /* virtual address of mapping */
736     size32_t pr_size;       /* size of mapping in bytes */
737     char    pr_mapname[64]; /* name in /proc/<pid>/object */
738     offset_t pr_offset;     /* offset into mapped object, if any */
739     int     pr_mflags;      /* protection and attribute flags */
740     int     pr_pagesize;    /* pagesize (bytes) for this mapping */
741     int     pr_shmid;       /* SysV shmid, -1 if not SysV shared memory */
742     int     pr_filler[1];   /* filler for future expansion */
743 } prmap32_t;

745 /*
746 * _ILP32 HAT memory-map interface. /proc/<pid>/xmap
747 */
748 typedef struct prxmap32 {
749     caddr32_t pr_vaddr;     /* virtual address of mapping */
750     size32_t pr_size;       /* size of mapping in bytes */
751     char    pr_mapname[PRMAPSZ]; /* name in /proc/<pid>/object */
752     offset_t pr_offset;     /* offset into mapped object, if any */
753     int     pr_mflags;      /* protection and attribute flags (see below) */
754     int     pr_pagesize;    /* pagesize (bytes) for this mapping */
755     int     pr_shmid;       /* SysV shmid, -1 if not SysV shared memory */
756     dev32_t pr_dev;         /* st_dev from stat64() of mapped object, or PRNODEV */
757     uint64_t pr_ino;        /* st_ino from stat64() of mapped object, if any */
758     uint32_t pr_rss;        /* pages of resident memory */
759     uint32_t pr_anon;       /* pages of resident anonymous memory */
760     uint32_t pr_locked;     /* pages of locked memory */
761     uint32_t pr_pad;        /* currently unused */
762     uint64_t pr_hatpagesize; /* pagesize of the hat mapping */
763     uint32_t pr_filler[6]; /* filler for future expansion */
764 } prxmap32_t;

766 /*
767 * _ILP32 Process credentials. PCSCRED and /proc/<pid>/cred
768 */
769 typedef struct prcred32 {
770     uid32_t pr_euid;        /* effective user id */
771     uid32_t pr_ruid;        /* real user id */
772     uid32_t pr_suid;        /* saved user id (from exec) */
773     gid32_t pr_egid;        /* effective group id */
774     gid32_t pr_rgid;        /* real group id */
775     gid32_t pr_sgid;        /* saved group id (from exec) */
776     int     pr_ngroups;     /* number of supplementary groups */
777     gid32_t pr_groups[1];  /* array of supplementary groups */
778 } prcred32_t;

780 /*
781 * _ILP32 Watchpoint interface. PCWATCH and /proc/<pid>/watch
782 */
783 typedef struct prwatch32 {
784     caddr32_t pr_vaddr;     /* virtual address of watched area */
785     size32_t pr_size;       /* size of watched area in bytes */
786     int     pr_wflags;      /* watch type flags */
787     int     pr_pad;
788 } prwatch32_t;

790 /*

```

```

791 * _ILP32 PCREAD/PCWRITE I/O interface.
792 */
793 typedef struct priovec32 {
794     caddr32_t pio_base;    /* buffer in controlling process */
795     size32_t pio_len;     /* size of read/write request */
796     off32_t pio_offset;   /* virtual address in target process */
797 } priovec32_t;

799 /*
800 * _ILP32 Resource usage. /proc/<pid>/usage /proc/<pid>/lwp/<lwpid>/lwpusage
801 */
802 typedef struct prusage32 {
803     id32_t pr_lwpid;      /* lwp id. 0: process or defunct */
804     int32_t pr_count;     /* number of contributing lwps */
805     timestruc32_t pr_tstamp; /* current time stamp */
806     timestruc32_t pr_create; /* process/lwp creation time stamp */
807     timestruc32_t pr_term; /* process/lwp termination time stamp */
808     timestruc32_t pr_rtime; /* total lwp real (elapsed) time */
809     timestruc32_t pr_utime; /* user level cpu time */
810     timestruc32_t pr_stime; /* system call cpu time */
811     timestruc32_t pr_ttime; /* other system trap cpu time */
812     timestruc32_t pr_tftime; /* text page fault sleep time */
813     timestruc32_t pr_dftime; /* data page fault sleep time */
814     timestruc32_t pr_kftime; /* kernel page fault sleep time */
815     timestruc32_t pr_ltime; /* user lock wait sleep time */
816     timestruc32_t pr_slptime; /* all other sleep time */
817     timestruc32_t pr_wtime; /* wait-cpu (latency) time */
818     timestruc32_t pr_stoptime; /* stopped time */
819     timestruc32_t filltime[6]; /* filler for future expansion */
820     uint32_t pr_minf; /* minor page faults */
821     uint32_t pr_majf; /* major page faults */
822     uint32_t pr_nswap; /* swaps */
823     uint32_t pr_inblk; /* input blocks */
824     uint32_t pr_oublk; /* output blocks */
825     uint32_t pr_msnd; /* messages sent */
826     uint32_t pr_mrcv; /* messages received */
827     uint32_t pr_sigs; /* signals received */
828     uint32_t pr_vctx; /* voluntary context switches */
829     uint32_t pr_ictx; /* involuntary context switches */
830     uint32_t pr_sysc; /* system calls */
831     uint32_t pr_ioch; /* chars read and written */
832     uint32_t filler[10]; /* filler for future expansion */
833 } prusage32_t;

835 /*
836 * _ILP32 Page data file. /proc/<pid>/pagedata
837 */

839 /* _ILP32 page data file header */
840 typedef struct prpageheader32 {
841     timestruc32_t pr_tstamp; /* real time stamp */
842     int32_t pr_nmap; /* number of address space mappings */
843     int32_t pr_npage; /* total number of pages */
844 } prpageheader32_t;

846 /* _ILP32 page data mapping header */
847 typedef struct prasmap32 {
848     caddr32_t pr_vaddr; /* virtual address of mapping */
849     size32_t pr_npage; /* number of pages in mapping */
850     char pr_mapname[64]; /* name in /proc/<pid>/object */
851     offset_t pr_offset; /* offset into mapped object, if any */
852     int pr_mflags; /* protection and attribute flags */
853     int pr_pagesize; /* pagesize (bytes) for this mapping */
854     int pr_shmid; /* SysV shmid, -1 if not SysV shared memory */
855     int pr_filler[1]; /* filler for future expansion */
856 } prasmap32_t;

```

```

858 /*
859 * _ILP32 Header for /proc/<pid>/lstatus /proc/<pid>/lpsinfo /proc/<pid>/lusage
860 */
861 typedef struct prheader32 {
862     int32_t pr_nent; /* number of entries */
863     int32_t pr_entsize; /* size of each entry, in bytes */
864 } prheader32_t;

866 #endif /* _SYS_SYSCALL32 */

868 #endif /* !_KERNEL && _STRUCTURED_PROC == 0 */

870 #ifdef __cplusplus
871 }
872 #endif

874 #endif /* _SYS_PROCFS_H */

```

new/usr/src/uts/common/sys/secflags.h

1

3060 Wed Jun 15 19:32:00 2016

new/usr/src/uts/common/sys/secflags.h

Code review comments from jeffpc

```
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;
25 typedef uint32_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;
_____ unchanged_portion_omitted
```