```
**********************************************************
    7285 Wed Aug 21 14:46:06 2013
new/usr/src/cmd/mdb/Makefile.module
3946 ::gcore
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Matthew Ahrens <mahrens@delphix.com>
**********************************************************
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#
#
# Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#
# Copyright (c) 2013 by Delphix. All rights reserved.
#

.KEEP_STATE:
.SUFFIXES:

include $(SRC)/cmd/mdb/Makefile.tools

$(KMOD_SOURCES_DIFFERENT)KMODSRCS = $(MODSRCS)

MODOBJS = $(MODSRCS:%.c=dmod/%.o)
KMODOBJS = $(KMODSRCS:%.c=kmod/%.o)

MODNAME = $(MODULE:%.so=%)
KMODULE = $(MODNAME)

MODFILE = dmod/$(MODULE)
KMODFILE = kmod/$(KMODULE)

#
# The mess below is designed to pick the right set of objects to build and/or
# lint.  We have three flavors:
#
#  1. proc and raw modules.  Only $(MODOBJS) are built.
#  2. kvm modules for systems without kmdb.  Only $(MODOBJS) are built.
#  3. kvm modules for systems with kmdb.  $(MODOBJS) and $(KMODOBJS) are built.
#
# Complicating matters, we'd like to make the distinction between 2 and 3 before
# this Makefile is loaded.  By default, we'll assume that all kvm modules should
# be built for kmdb.  If, however, the user sets $(MODULE_BUILD_TYPE) to 'mdb',
# the kmdb variant of the module won't be built.
#

# Which flavors are to be built?
TARGETS_kvm_type_       = both  # Build both if $(MODULE_BUILD_TYPE) is unset
```



```
TARGETS_kvm_type_kmdb   = both
TARGETS_kvm_type_mdb    = mdb
TARGETS_kvm_type        = $(TARGETS_kvm_type_$(MODULE_BUILD_TYPE))

# What should we build?
TARGETS_kvm_kmdb        = $(KMODFILE)
TARGETS_kvm_mdb         = $(MODFILE)
TARGETS_kvm_both        = $(TARGETS_kvm_mdb) $(TARGETS_kvm_kmdb)
TARGETS_kvm             = $(TARGETS_kvm_$(TARGETS_kvm_type))
TARGETS_proc            = $(MODFILE)
TARGETS_raw             = $(MODFILE)
TARGETS                 = $(TARGETS_$(MDBTGT))

# Where should we install that which we've built?
ROOTTGTS_kvm_type       = $(TARGETS_kvm_type)   # overridden by mdb_ks
ROOTTGTS_kvm_kmdb       = $(ROOTKMOD)/$(KMODULE)
ROOTTGTS_kvm_mdb        = $(ROOTMOD)/$(MODULE)
ROOTTGTS_kvm_both       = $(ROOTTGTS_kvm_mdb) $(ROOTTGTS_kvm_kmdb)
ROOTTGTS_kvm            = $(ROOTTGTS_kvm_$(ROOTTGTS_kvm_type))
ROOTTGTS_proc           = $(ROOTMOD)/$(MODULE)
ROOTTGTS_raw            = $(ROOTMOD)/$(MODULE)
ROOTTGTS                = $(ROOTTGTS_$(MDBTGT))

# What should we lint?
KLINTOBJS               = $(KMODOBJS:%.o=%.ln)
LINTOBJS                = $(MODOBJS:%.o=%.ln)

LINTFILES_kvm_type      = $(TARGETS_kvm_type)
LINTFILES_kvm_both      = $(KLINTOBJS) $(LINTOBJS)
LINTFILES_kvm_mdb       = $(LINTOBJS)
LINTFILES_kvm           = $(LINTFILES_kvm_$(LINTFILES_kvm_type))
LINTFILES_proc          = $(LINTOBJS)
LINTFILES_raw           = $(LINTOBJS)
LINTFILES               = $(LINTFILES_$(MDBTGT))

kvm_TGTFLAGS            = -D_KERNEL
proc_TGTFLAGS           = -D_USER

C99MODE                 = $(C99_ENABLE)

CFLAGS                  += $(CCVERBOSE)
CFLAGS64                += $(CCVERBOSE)
CPPFLAGS                += $($(MDBTGT)_TGTFLAGS) -I../../../common
LDFLAGS                 += $(ZTEXT)
LDFLAGS64               += $(ZTEXT)

# Module type-specific compiler flags
$(MODOBJS) :=                   CFLAGS += $(C_BIGPICFLAGS) $(XREGSFLAG)
$(MODOBJS) :=                   CFLAGS64 += $(C_BIGPICFLAGS) $(XREGSFLAG)
$(KMODOBJS) $(KLINTOBJS) :=     CPPFLAGS += -D_KMDB
$(KMODOBJS) :=                  V9CODESIZE = $(CCABS32)
$(KMODOBJS) :=                  DTS_ERRNO =

# Modules aren't allowed to export symbols
MAPFILE         = $(SRC)/cmd/mdb/common/modules/conf/mapfile

# Modules typically make external references.  To provide for -zdefs use
# and clean ldd(1) processing, explicitly define all external references.
MAPFILE-EXT     = $(SRC)/cmd/mdb/common/modules/conf/mapfile-extern

#
# kmdb is a kernel module, so we'll use the kernel's build flags.
$(KMODOBJS) := CFLAGS += $(STAND_FLAGS_32)
$(KMODOBJS) := CFLAGS64 += $(STAND_FLAGS_64)

#
```

```
126 # Override this to pull source files from another directory
127 #
128 MODSRCS_DIR = ../../../common/modules/genunix

130 all: $$(TARGETS)

132 install: all $$(ROOTTGTS)

134 dmods: install

136 clean.lint:
137         $(RM) $(LINTFILES)

139 clean:
140         $(RM) $(MODOBJS) $(KMODOBJS) $(CLEANFILES)

142 clobber: clean clean.lint
143         $(RM) $(MODFILE) $(KMODFILE) $(CLOBBERFILES)

145 lint: $$(LINTFILES)

147 .NO_PARALLEL:
148 .PARALLEL: $(MODOBJS) $(KMODOBJS) mdb_tgt kmdb_tgt dmod kmod \
149         $(TARGETS) $(LINTFILES)

151 $(MODFILE): dmod .WAIT $(MODOBJS) $$(MAPFILE-EXT)
152         $(LINK.c) $(ZDEFS) $(ZIGNORE) $(MAPFILE-EXT:%=-M%) $(GSHARED) \
153             $(MODOBJS) -o $@ $(LDLIBS) -lc
154         $(CTFMERGE) -L VERSION -o $@ $(MODOBJS)
155         $(POST_PROCESS_SO)

157 #
158 # kmdb dmods must *not* stray from the module API.  To ensure that they don't,
159 # we try to link them, at build time, against an object that exports the symbols
160 # that they can legally use.  The link test object is, however, only built when
161 # kmdb itself is built.  Requiring module developers to build kmdb first would
162 # be painful, so by default, module-level builds don't do the link test (the
163 # $(POUND_SIGN) assignment below takes care of that).  Builds of the entire
164 # tree can, however, guarantee the construction of kmdb first, and as such can
165 # override the setting of $(KMDB_LINKTEST_ENABLE).  This override causes the
166 # link test to be run.
167 #
168 # Developers wanting to force a link test for a single module can use the
169 # 'linktest' target from within a module directory.
170 #
171 LINKTESTOBJ = $(KMDBDIR)/kmdb_modlinktest.o

173 KMDB_LINKTEST = \
174         $(LD) $(ZDEFS) -dy -r -o $@.linktest $(KMODOBJS) \
175         $(STANDOBJS) $(LINKTESTOBJ) && \
176         $(RM) $@.linktest

178 KMDB_LINKTEST_ENABLE=$(POUND_SIGN)
179 $(KMDB_LINKTEST_ENABLE)KMDB_LINKTEST_CMD = $(KMDB_LINKTEST)

181 #
182 # Ensure that dmods don't use floating point
183 #
184 KMDB_FPTEST_CMD = $(KMDB_FPTEST)

186 $(KMODFILE): kmod .WAIT $(KMODOBJS) $(MAPFILE)
187         $(LD) -dy -r $(MAPFILE:%=-M%) -Nmisc/kmdbmod -o $@ $(KMODOBJS) \
188         $(STANDOBJS)
189         $(KMDB_LINKTEST_CMD)
190         $(KMDB_FPTEST_CMD)
191         $(CTFMERGE) -f -L VERSION -o $@ $(KMODOBJS)
```



```
192         $(SETDYNFLAG) -f DF_1_NOKSYMS $@

194 linktest: linktest_check .WAIT kmod .WAIT $(KMODOBJS)
195         $(KMDB_LINKTEST)

197 linktest_check:
198         @if [ "$(MDBTGT)" != "kvm" ] ; then \
199                 echo "ERROR: linktest is not supported non-kvm/disasm dmods" \
200                     >&2 ; \
201                 exit 1 ; \
202         fi

204 #
205 # Dynamic rules for object construction
206 #
207 dmod/%.o kmod/%.o: %.c
208         $(COMPILE.c) -o $@ $<
209         $(CTFCONVERT_O)

211 dmod/%.o kmod/%.o: ../%.c
212         $(COMPILE.c) -o $@ $<
213         $(CTFCONVERT_O)

215 dmod/%.o kmod/%.o: ../../../common/modules/$(MODNAME)/%.c
216         $(COMPILE.c) -o $@ $<
217         $(CTFCONVERT_O)

219 dmod/%.o kmod/%.o: $$(MODSRCS_DIR)/%.c
220         $(COMPILE.c) -o $@ $<
221         $(CTFCONVERT_O)

223 #
224 # Lint
225 #
226 dmod/%.ln kmod/%.ln: %.c
227         $(LINT.c) -dirout=$(@D) -c $<

229 dmod/%.ln kmod/%.ln: ../%.c
230         $(LINT.c) -dirout=$(@D) -c $<

232 dmod/%.ln kmod/%.ln: ../../../common/modules/$(MODNAME)/%.c
233         $(LINT.c) -dirout=$(@D) -c $<

235 dmod/%.ln kmod/%.ln: $$(MODSRCS_DIR)/%.c
236         $(LINT.c) -dirout=$(@D) -c $<

238 #
239 # Installation targets
240 #

242 $(ROOT)/usr/lib/mdb/$(MDBTGT): $(ROOT)/usr/lib/mdb
243         $(INS.dir)

245 $(ROOT)/usr/lib/mdb:
246         $(INS.dir)

248 $(ROOT)/kernel/kmdb:
249         $(INS.dir)

251 $(ROOTMOD)/$(MODULE): $(ROOTMOD)

253 $(ROOTKMOD)/$(KMODULE): $(ROOTKMOD)

255 kmod dmod:
256         -@mkdir -p $@
```

```
**********************************************************
   21454 Wed Aug 21 14:46:08 2013
new/usr/src/cmd/mdb/common/mdb/mdb_modapi.c
3946 ::gcore
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Matthew Ahrens <mahrens@delphix.com>
**********************************************************
______unchanged_portion_omitted_

  73 ssize_t
  74 mdb_aread(void *buf, size_t nbytes, uintptr_t addr, void *as)
  75 {
  76         ssize_t rbytes = mdb_tgt_aread(mdb.m_target, as, buf, nbytes, addr);

  78         if (rbytes > 0 && rbytes < nbytes)
  79                 return (set_errbytes(rbytes, nbytes));

  81         return (rbytes);
  82 }

  84 ssize_t
  85 mdb_awrite(const void *buf, size_t nbytes, uintptr_t addr, void *as)
  86 {
  87         return (mdb_tgt_awrite(mdb.m_target, as, buf, nbytes, addr));
  88 }

  90 ssize_t
  91 mdb_fread(void *buf, size_t nbytes, uintptr_t addr)
  92 {
  93         ssize_t rbytes = mdb_tgt_fread(mdb.m_target, buf, nbytes, addr);

  95         if (rbytes > 0 && rbytes < nbytes)
  96                 return (set_errbytes(rbytes, nbytes));

  98         return (rbytes);
  99 }
______unchanged_portion_omitted_
```

```
**********************************************************
   15517 Wed Aug 21 14:46:09 2013
new/usr/src/cmd/mdb/common/mdb/mdb_modapi.h
3946 ::gcore
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Matthew Ahrens <mahrens@delphix.com>
**********************************************************
```

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

/*
 * Copyright (c) 1999, 2010, Oracle and/or its affiliates. All rights reserved.
 * Copyright (c) 2013 by Delphix. All rights reserved.
 * Copyright (c) 2012 by Delphix. All rights reserved.
 * Copyright (c) 2012 Joyent, Inc. All rights reserved.
 */

#ifndef _MDB_MODAPI_H
#define _MDB_MODAPI_H

/*
 * MDB Module API
 *
 * The debugger provides a set of interfaces for use in writing loadable
 * debugger modules.  Modules that call functions not listed in this header
 * file may not be compatible with future versions of the debugger.
 */

#include <sys/types.h>
#include <gelf.h>

#ifdef  __cplusplus
extern "C" {
#endif

/*
 * Make sure that NULL, TRUE, FALSE, MIN, and MAX have the usual definitions
 * so module writers can depend on these macros and defines.
 */
#ifndef NULL
#if defined(_LP64) && !defined(__cplusplus)
#define NULL    0L
#else
#define NULL    0
#endif
#endif

#ifndef TRUE
```



```
#define TRUE    1
#endif

#ifndef FALSE
#define FALSE   0
#endif

#ifndef MIN
#define MIN(x, y) ((x) < (y) ? (x) : (y))
#endif

#ifndef MAX
#define MAX(x, y) ((x) > (y) ? (x) : (y))
#endif

#define MDB_API_VERSION 4       /* Current API version number */

/*
 * Debugger command function flags:
 */
#define DCMD_ADDRSPEC   0x01    /* Dcmd invoked with explicit address */
#define DCMD_LOOP       0x02    /* Dcmd invoked in loop with ,cnt syntax */
#define DCMD_LOOPFIRST  0x04    /* Dcmd invoked as first iteration of LOOP */
#define DCMD_PIPE       0x08    /* Dcmd invoked with input from pipe */
#define DCMD_PIPE_OUT   0x10    /* Dcmd invoked with output set to pipe */

#define DCMD_HDRSPEC(fl)        (((fl) & DCMD_LOOPFIRST) || !((fl) & DCMD_LOOP))

/*
 * Debugger tab command function flags
 */
#define DCMD_TAB_SPACE  0x01    /* Tab cb invoked with trailing space */

/*
 * Debugger command function return values:
 */
#define DCMD_OK         0       /* Dcmd completed successfully */
#define DCMD_ERR        1       /* Dcmd failed due to an error */
#define DCMD_USAGE      2       /* Dcmd usage error; abort and print usage */
#define DCMD_NEXT       3       /* Invoke next dcmd in precedence list */
#define DCMD_ABORT      4       /* Dcmd failed; abort current loop or pipe */

#define OFFSETOF(s, m)          (size_t)(&(((s *)0)->m))

extern int mdb_prop_postmortem; /* Are we looking at a static dump? */
extern int mdb_prop_kernel;     /* Are we looking at a kernel? */

typedef enum {
        MDB_TYPE_STRING,        /* a_un.a_str is valid */
        MDB_TYPE_IMMEDIATE,     /* a_un.a_val is valid */
        MDB_TYPE_CHAR           /* a_un.a_char is valid */
} mdb_type_t;
______unchanged_portion_omitted_

extern int mdb_pwalk(const char *, mdb_walk_cb_t, void *, uintptr_t);
extern int mdb_walk(const char *, mdb_walk_cb_t, void *);

extern int mdb_pwalk_dcmd(const char *, const char *,
        int, const mdb_arg_t *, uintptr_t);

extern int mdb_walk_dcmd(const char *, const char *, int, const mdb_arg_t *);

extern int mdb_layered_walk(const char *, mdb_walk_state_t *);

extern int mdb_call_dcmd(const char *, uintptr_t,
        uint_t, int, const mdb_arg_t *);
```

```
204 extern int mdb_add_walker(const mdb_walker_t *);
205 extern int mdb_remove_walker(const char *);

207 extern ssize_t mdb_vread(void *, size_t, uintptr_t);
208 extern ssize_t mdb_vwrite(const void *, size_t, uintptr_t);

210 extern ssize_t mdb_aread(void *, size_t, uintptr_t, void *);
211 extern ssize_t mdb_awrite(const void *, size_t, uintptr_t, void *);

213 extern ssize_t mdb_fread(void *, size_t, uintptr_t);
214 extern ssize_t mdb_fwrite(const void *, size_t, uintptr_t);

216 extern ssize_t mdb_pread(void *, size_t, uint64_t);
217 extern ssize_t mdb_pwrite(const void *, size_t, uint64_t);

219 extern ssize_t mdb_readstr(char *, size_t, uintptr_t);
220 extern ssize_t mdb_writestr(const char *, uintptr_t);

222 extern ssize_t mdb_readsym(void *, size_t, const char *);
223 extern ssize_t mdb_writesym(const void *, size_t, const char *);

225 extern ssize_t mdb_readvar(void *, const char *);
226 extern ssize_t mdb_writevar(const void *, const char *);

228 #define MDB_SYM_NAMLEN  1024                    /* Recommended max name len */

230 #define MDB_SYM_FUZZY   0                       /* Match closest address */
231 #define MDB_SYM_EXACT   1                       /* Match exact address only */

233 #define MDB_OBJ_EXEC    ((const char *)0L)      /* Primary executable file */
234 #define MDB_OBJ_RTLD    ((const char *)1L)      /* Run-time link-editor */
235 #define MDB_OBJ_EVERY   ((const char *)-1L)     /* All known symbols */

237 extern int mdb_lookup_by_name(const char *, GElf_Sym *);
238 extern int mdb_lookup_by_obj(const char *, const char *, GElf_Sym *);
239 extern int mdb_lookup_by_addr(uintptr_t, uint_t, char *, size_t, GElf_Sym *);

241 typedef uintptr_t mdb_tid_t;
242 typedef uint64_t mdb_reg_t;

244 extern int mdb_getareg(mdb_tid_t, const char *, mdb_reg_t *);

246 #define MDB_OPT_SETBITS 1                       /* Set specified flag bits */
247 #define MDB_OPT_CLRBITS 2                       /* Clear specified flag bits */
248 #define MDB_OPT_STR     3                       /* const char * argument */
249 #define MDB_OPT_UINTPTR 4                       /* uintptr_t argument */
250 #define MDB_OPT_UINT64  5                       /* uint64_t argument */
251 #define MDB_OPT_UINTPTR_SET     6               /* boolean_t+uintptr_t args */

253 extern int mdb_getopts(int, const mdb_arg_t *, ...);

255 extern u_longlong_t mdb_strtoull(const char *);

257 #define UM_NOSLEEP      0x0     /* Do not call failure handler; may fail */
258 #define UM_SLEEP        0x1     /* Can block for memory; will always succeed */
259 #define UM_GC           0x2     /* Garbage-collect this block automatically */

261 extern void *mdb_alloc(size_t, uint_t);
262 extern void *mdb_zalloc(size_t, uint_t);
263 extern void mdb_free(void *, size_t);

265 extern size_t mdb_snprintf(char *, size_t, const char *, ...);
266 extern void mdb_printf(const char *, ...);
267 extern void mdb_warn(const char *, ...);
268 extern void mdb_flush(void);
```



```
270 extern int mdb_ffs(uintmax_t);

272 extern void mdb_nhconvert(void *, const void *, size_t);

274 #define MDB_DUMP_RELATIVE       0x0001  /* Start numbering at 0 */
275 #define MDB_DUMP_ALIGN          0x0002  /* Enforce paragraph alignment */
276 #define MDB_DUMP_PEDANT         0x0004  /* Full-width addresses */
277 #define MDB_DUMP_ASCII          0x0008  /* Display ASCII values */
278 #define MDB_DUMP_HEADER         0x0010  /* Display a header */
279 #define MDB_DUMP_TRIM           0x0020  /* Trim at boundaries */
280 #define MDB_DUMP_SQUISH         0x0040  /* Eliminate redundant lines */
281 #define MDB_DUMP_NEWDOT         0x0080  /* Update dot when done */
282 #define MDB_DUMP_ENDIAN         0x0100  /* Adjust for endianness */
283 #define MDB_DUMP_WIDTH(x)       ((((x) - 1) & 0xf) << 16) /* paragraphs/line */
284 #define MDB_DUMP_GROUP(x)       ((((x) - 1) & 0xff) << 20) /* bytes/group */

286 typedef ssize_t (*mdb_dumpptr_cb_t)(void *, size_t, uintptr_t, void *);
287 typedef ssize_t (*mdb_dump64_cb_t)(void *, size_t, uint64_t, void *);

289 extern int mdb_dumpptr(uintptr_t, size_t, uint_t, mdb_dumpptr_cb_t, void *);
290 extern int mdb_dump64(uint64_t, uint64_t, uint_t, mdb_dump64_cb_t, void *);

292 extern const char *mdb_one_bit(int, int, int);
293 extern const char *mdb_inval_bits(int, int, int);

295 extern ulong_t mdb_inc_indent(ulong_t);
296 extern ulong_t mdb_dec_indent(ulong_t);

298 extern int mdb_eval(const char *);
299 extern void mdb_set_dot(uintmax_t);
300 extern uintmax_t mdb_get_dot(void);

302 extern void mdb_get_pipe(mdb_pipe_t *);
303 extern void mdb_set_pipe(const mdb_pipe_t *);

305 extern ssize_t mdb_get_xdata(const char *, void *, size_t);

307 typedef int (*mdb_object_cb_t)(mdb_object_t *, void *);
308 extern int mdb_object_iter(mdb_object_cb_t, void *);

310 #define MDB_SYMTAB              1       /* Normal symbol table (.symtab) */
311 #define MDB_DYNSYM              2       /* Dynamic symbol table (.dynsym) */

313 #define MDB_BIND_LOCAL          0x0001  /* Local (static-scope) symbols */
314 #define MDB_BIND_GLOBAL         0x0002  /* Global symbols */
315 #define MDB_BIND_WEAK           0x0004  /* Weak binding symbols */
316 #define MDB_BIND_ANY            0x0007  /* Any of the above */

318 #define MDB_TYPE_NOTYPE         0x0100  /* Symbol has no type */
319 #define MDB_TYPE_OBJECT         0x0200  /* Symbol refers to data */
320 #define MDB_TYPE_FUNC           0x0400  /* Symbol refers to text */
321 #define MDB_TYPE_SECT           0x0800  /* Symbol refers to a section */
322 #define MDB_TYPE_FILE           0x1000  /* Symbol refers to a source file */
323 #define MDB_TYPE_COMMON         0x2000  /* Symbol refers to a common block */
324 #define MDB_TYPE_TLS            0x4000  /* Symbol refers to TLS */

326 #define MDB_TYPE_ANY            0x7f00  /* Any of the above */

328 typedef int (*mdb_symbol_cb_t)(mdb_symbol_t *, void *);
329 extern int mdb_symbol_iter(const char *, uint_t, uint_t, mdb_symbol_cb_t,
330     void *);

332 #define MDB_STATE_IDLE          0       /* Target is idle (not running yet) */
333 #define MDB_STATE_RUNNING       1       /* Target is currently executing */
334 #define MDB_STATE_STOPPED       2       /* Target is stopped */
```

```
335 #define MDB_STATE_UNDEAD        3       /* Target is undead (zombie) */
336 #define MDB_STATE_DEAD          4       /* Target is dead (core dump) */
337 #define MDB_STATE_LOST          5       /* Target lost by debugger */

339 extern int mdb_get_state(void);

341 #define MDB_CALLBACK_STCHG      1
342 #define MDB_CALLBACK_PROMPT     2

344 typedef void (*mdb_callback_f)(void *);

346 extern void *mdb_callback_add(int, mdb_callback_f, void *);
347 extern void mdb_callback_remove(void *);

349 #define MDB_TABC_ALL_TYPES      0x1     /* Include array types in type output */
350 #define MDB_TABC_MEMBERS        0x2     /* Tab comp. types with members */
351 #define MDB_TABC_NOPOINT        0x4     /* Tab comp. everything but pointers */
352 #define MDB_TABC_NOARRAY        0x8     /* Don't include array data in output */

354 /*
355  * Module's interaction path
356  */
357 extern void mdb_tab_insert(mdb_tab_cookie_t *, const char *);
358 extern void mdb_tab_setmbase(mdb_tab_cookie_t *, const char *);

360 /*
361  * Tab completion utility functions for modules.
362  */
363 extern int mdb_tab_complete_type(mdb_tab_cookie_t *, const char *, uint_t);
364 extern int mdb_tab_complete_member(mdb_tab_cookie_t *, const char *,
365     const char *);
366 extern int mdb_tab_typename(int *, const mdb_arg_t **, char *buf, size_t len);

368 /*
369  * Tab completion functions for common signatures.
370  */
371 extern int mdb_tab_complete_mt(mdb_tab_cookie_t *, uint_t, int,
372     const mdb_arg_t *);

374 extern size_t strlcat(char *, const char *, size_t);
375 extern char *strcat(char *, const char *);
376 extern char *strcpy(char *, const char *);
377 extern char *strncpy(char *, const char *, size_t);

379 /* Need to be consistent with <string.h> C++ definitions */
380 #if __cplusplus >= 199711L
381 extern const char *strchr(const char *, int);
382 #ifndef _STRCHR_INLINE
383 #define _STRCHR_INLINE
384 extern "C++" {
385         inline char *strchr(char *__s, int __c) {
386                 return (char *)strchr((const char *)__s, __c);
387         }
388 }
```

*unchanged_portion_omitted_*

```
**********************************************************
    6356 Wed Aug 21 14:46:10 2013
new/usr/src/cmd/mdb/common/modules/conf/mapfile-extern
3946 ::gcore
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Matthew Ahrens <mahrens@delphix.com>
**********************************************************
#
# Copyright (c) 2008, 2010, Oracle and/or its affiliates. All rights reserved.
# Copyright (c) 2013 by Delphix. All rights reserved.
# Copyright (c) 2012 by Delphix. All rights reserved.
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#

#
# MAPFILE HEADER START
#
# WARNING:  STOP NOW.  DO NOT MODIFY THIS FILE.
# Object versioning must comply with the rules detailed in
#
#       usr/src/lib/README.mapfiles
#
# You should not be making modifications here until you've read the most current
# copy of that file. If you need help, contact a gatekeeper for guidance.
#
# MAPFILE HEADER END
#

$mapfile_version 2

# External interface requirements
SYMBOL_SCOPE {
        global:
#               Plwp_iter                       { FLAGS = EXTERN };
#               Pmapping_iter                   { FLAGS = EXTERN };

                _mdb_ks_ncpu                    { FLAGS = EXTERN };
                _mdb_ks_pagemask                { FLAGS = EXTERN };
                _mdb_ks_pageoffset              { FLAGS = EXTERN };
                _mdb_ks_pageshift               { FLAGS = EXTERN };
                _mdb_ks_pagesize                { FLAGS = EXTERN };
                _mdb_ks_pageoffset              { FLAGS = EXTERN };

                mdb                             { FLAGS = EXTERN };
                mdb_add_walker                  { FLAGS = EXTERN };
                mdb_alloc                       { FLAGS = EXTERN };
                mdb_aread                       { FLAGS = EXTERN };
                mdb_awrite                      { FLAGS = EXTERN };
```



```
                mdb_call_dcmd                   { FLAGS = EXTERN };
                mdb_callback_add                { FLAGS = EXTERN };
                mdb_callback_remove             { FLAGS = EXTERN };
                mdb_cpuset_find                 { FLAGS = EXTERN };
                mdb_ctf_array_info              { FLAGS = EXTERN };
                mdb_ctf_enum_name               { FLAGS = EXTERN };
                mdb_ctf_lookup_by_addr          { FLAGS = EXTERN };
                mdb_ctf_lookup_by_name          { FLAGS = EXTERN };
                mdb_ctf_member_iter             { FLAGS = EXTERN };
                mdb_ctf_module_lookup           { FLAGS = EXTERN };
                mdb_ctf_offsetof                { FLAGS = EXTERN };
                mdb_ctf_offsetof_by_name        { FLAGS = EXTERN };
                mdb_ctf_readsym                 { FLAGS = EXTERN };
                mdb_ctf_type_cmp                { FLAGS = EXTERN };
                mdb_ctf_type_invalidate         { FLAGS = EXTERN };
                mdb_ctf_type_kind               { FLAGS = EXTERN };
                mdb_ctf_type_name               { FLAGS = EXTERN };
                mdb_ctf_type_reference          { FLAGS = EXTERN };
                mdb_ctf_type_resolve            { FLAGS = EXTERN };
                mdb_ctf_type_size               { FLAGS = EXTERN };
                mdb_ctf_type_valid              { FLAGS = EXTERN };
                mdb_ctf_vread                   { FLAGS = EXTERN };
                mdb_ddi_pathname                { FLAGS = EXTERN };
                mdb_dec_indent                  { FLAGS = EXTERN };
                mdb_devinfo2driver              { FLAGS = EXTERN };
                mdb_devinfo2statep              { FLAGS = EXTERN };
                mdb_dlpi_prim                   { FLAGS = EXTERN };
                mdb_dump64                      { FLAGS = EXTERN };
                mdb_dumpptr                     { FLAGS = EXTERN };
                mdb_eval                        { FLAGS = EXTERN };
                mdb_fdio_create_path            { FLAGS = EXTERN };
                mdb_fdio_fileno                 { FLAGS = EXTERN };
                mdb_ffs                         { FLAGS = EXTERN };
                mdb_flush                       { FLAGS = EXTERN };
                mdb_fread                       { FLAGS = EXTERN };
                mdb_free                        { FLAGS = EXTERN };
                mdb_fwrite                      { FLAGS = EXTERN };
                mdb_gelf_create                 { FLAGS = EXTERN };
                mdb_gelf_destroy                { FLAGS = EXTERN };
                mdb_gelf_sect_by_name           { FLAGS = EXTERN };
                mdb_gelf_sect_load              { FLAGS = EXTERN };
                mdb_getareg                     { FLAGS = EXTERN };
                mdb_get_dot                     { FLAGS = EXTERN };
                mdb_get_lbolt                   { FLAGS = EXTERN };
                mdb_get_pipe                    { FLAGS = EXTERN };
                mdb_get_soft_state_byaddr       { FLAGS = EXTERN };
                mdb_get_state                   { FLAGS = EXTERN };
                mdb_get_xdata                   { FLAGS = EXTERN };
                mdb_gethrtime                   { FLAGS = EXTERN };
                mdb_getopts                     { FLAGS = EXTERN };
                mdb_inc_indent                  { FLAGS = EXTERN };
                mdb_inval_bits                  { FLAGS = EXTERN };
                mdb_io_destroy                  { FLAGS = EXTERN };
                mdb_iob_clrflags                { FLAGS = EXTERN };
                mdb_iob_getflags                { FLAGS = EXTERN };
                mdb_iob_resize                  { FLAGS = EXTERN };
                mdb_iob_setflags                { FLAGS = EXTERN };
                mdb_layered_walk                { FLAGS = EXTERN };
                mdb_lookup_by_addr              { FLAGS = EXTERN };
                mdb_lookup_by_name              { FLAGS = EXTERN };
                mdb_lookup_by_obj               { FLAGS = EXTERN };
                mdb_mac_addr                    { FLAGS = EXTERN };
                mdb_major_to_name               { FLAGS = EXTERN };
                mdb_mblk_count                  { FLAGS = EXTERN };
                mdb_memio_create                { FLAGS = EXTERN };
                mdb_name_to_major               { FLAGS = EXTERN };
```

```
 124                 mdb_nhconvert                   { FLAGS = EXTERN };
 125                 mdb_object_iter                 { FLAGS = EXTERN };
 126                 mdb_one_bit                     { FLAGS = EXTERN };
 127                 mdb_page2pfn                    { FLAGS = EXTERN };
 128                 mdb_page_lookup                 { FLAGS = EXTERN };
 129                 mdb_pfn2page                    { FLAGS = EXTERN };
 130                 mdb_pid2proc                    { FLAGS = EXTERN };
 131                 mdb_pread                       { FLAGS = EXTERN };
 132                 mdb_printf                      { FLAGS = EXTERN };
 133                 mdb_prop_kernel                 { FLAGS = EXTERN };
 134                 mdb_prop_postmortem             { FLAGS = EXTERN };
 135                 mdb_pwalk                       { FLAGS = EXTERN };
 136                 mdb_pwalk_dcmd                  { FLAGS = EXTERN };
 137                 mdb_pwrite                      { FLAGS = EXTERN };
 138                 mdb_qinfo                       { FLAGS = EXTERN };
 139                 mdb_qname                       { FLAGS = EXTERN };
 140                 mdb_qops_install                { FLAGS = EXTERN };
 141                 mdb_qops_remove                 { FLAGS = EXTERN };
 142                 mdb_qrnext_default              { FLAGS = EXTERN };
 143                 mdb_qwnext                      { FLAGS = EXTERN };
 144                 mdb_qwnext_default              { FLAGS = EXTERN };
 145                 mdb_read_refstr                 { FLAGS = EXTERN };
 146                 mdb_readstr                     { FLAGS = EXTERN };
 147                 mdb_readsym                     { FLAGS = EXTERN };
 148                 mdb_readvar                     { FLAGS = EXTERN };
 149                 mdb_remove_walker               { FLAGS = EXTERN };
 150                 mdb_set_dot                     { FLAGS = EXTERN };
 151                 mdb_set_pipe                    { FLAGS = EXTERN };
 152                 mdb_snprintf                    { FLAGS = EXTERN };
 153                 mdb_strtoull                    { FLAGS = EXTERN };
 154                 mdb_symbol_iter                 { FLAGS = EXTERN };
 155                 mdb_tgt_notsup                  { FLAGS = EXTERN };
 156                 mdb_vnode2path                  { FLAGS = EXTERN };
 157                 mdb_vread                       { FLAGS = EXTERN };
 158                 mdb_vtype2chr                   { FLAGS = EXTERN };
 159                 mdb_vwrite                      { FLAGS = EXTERN };
 160                 mdb_walk                        { FLAGS = EXTERN };
 161                 mdb_walk_dcmd                   { FLAGS = EXTERN };
 162                 mdb_warn                        { FLAGS = EXTERN };
 163                 mdb_whatis_done                 { FLAGS = EXTERN };
 164                 mdb_whatis_flags                { FLAGS = EXTERN };
 165                 mdb_whatis_match                { FLAGS = EXTERN };
 166                 mdb_whatis_overlaps             { FLAGS = EXTERN };
 167                 mdb_whatis_register             { FLAGS = EXTERN };
 168                 mdb_whatis_report_address       { FLAGS = EXTERN };
 169                 mdb_whatis_report_object        { FLAGS = EXTERN };
 170                 mdb_writestr                    { FLAGS = EXTERN };
 171                 mdb_writesym                    { FLAGS = EXTERN };
 172                 mdb_writevar                    { FLAGS = EXTERN };
 173                 mdb_zalloc                      { FLAGS = EXTERN };
 174 };
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    1832 Wed Aug 21 14:46:11 2013
new/usr/src/cmd/mdb/common/modules/genunix/Makefile.files
3946 ::gcore
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Matthew Ahrens <mahrens@delphix.com>
**********************************************************
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#
#
# Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
# Copyright (c) 1999, 2010, Oracle and/or its affiliates. All rights reserved.
# Copyright (c) 2013, Joyent, Inc.  All rights reserved.
# Copyright (c) 2013 by Delphix. All rights reserved.
#

#
# This file simply contains the list of sources files compiled together
# to create the genunix mdb module.   Having them in one place saves
# a bunch of unnecessary replication.
#
GENUNIX_SRCS =          \
        avl.c           \
        bio.c           \
        bitset.c        \
        combined.c      \
        contract.c      \
        cpupart.c       \
        cred.c          \
        ctxop.c         \
        cyclic.c        \
        damap.c         \
        ddi_periodic.c  \
        devinfo.c       \
        dist.c          \
        findstack.c     \
        findstack_subr.c \
        fm.c            \
        gcore.c         \
        genunix.c       \
        group.c         \
        hotplug.c       \
        irm.c           \
        kgrep.c         \
        kmem.c          \
        ldi.c           \
        leaky.c         \
        leaky_subr.c    \
```



```
        lgrp.c          \
        list.c          \
        log.c           \
        mdi.c           \
        memory.c        \
        mmd.c           \
        modhash.c       \
        ndievents.c     \
        net.c           \
        netstack.c      \
        nvpair.c        \
        pg.c            \
        rctl.c          \
        sobj.c          \
        streams.c       \
        sysevent.c      \
        taskq.c         \
        thread.c        \
        tsd.c           \
        tsol.c          \
        vfs.c           \
        zone.c
```

```
**********************************************************
    7791 Wed Aug 21 14:46:13 2013
new/usr/src/cmd/mdb/common/modules/genunix/avl.c
3946 ::gcore
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Matthew Ahrens <mahrens@delphix.com>
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2008 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */
/*
 * Copyright (c) 2013 by Delphix. All rights reserved.
 */

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

#include <sys/avl.h>

#include <mdb/mdb_modapi.h>

struct aw_info {
        void *aw_buff;          /* buffer to hold tree element */
        avl_tree_t aw_tree;     /* copy of avl_tree_t being walked */
        uintptr_t aw_end;       /* last node in specified range */
        const char *aw_elem_name;
        int (*aw_elem_check)(void *, uintptr_t, void *);
        void *aw_elem_check_arg;
};
______unchanged_portion_omitted_

/*
 * Release the memory allocated for the walk
 */
void
avl_walk_fini(mdb_walk_state_t *wsp)
{
        struct aw_info *aw;

        aw = (struct aw_info *)wsp->walk_data;

        if (aw == NULL)
                return;

        if (aw->aw_buff != NULL)
                mdb_free(aw->aw_buff, aw->aw_tree.avl_size);
```



```
        mdb_free(aw, sizeof (struct aw_info));
}

/*
 * This function is named avl_walk_mdb to avoid a naming conflict with the
 * existing avl_walk function.
 */
int
avl_walk_mdb(uintptr_t addr, mdb_walk_cb_t callback, void *cbdata)
{
        mdb_walk_state_t ws;
        int ret;

        ws.walk_addr = addr;
        ws.walk_callback = callback;
        ws.walk_cbdata = cbdata;

        avl_walk_init(&ws);
        while ((ret = avl_walk_step(&ws)) == WALK_NEXT)
                continue;
        avl_walk_fini(&ws);

        return (ret);
}
______unchanged_portion_omitted_
```

```
**********************************************************
    1822 Wed Aug 21 14:46:14 2013
new/usr/src/cmd/mdb/common/modules/genunix/avl.h
3946 ::gcore
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Matthew Ahrens <mahrens@delphix.com>
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2008 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */
/*
 * Copyright (c) 2013 by Delphix. All rights reserved.
 */

#ifndef _MDB_AVL_H
#define _MDB_AVL_H

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

#ifdef  __cplusplus
extern "C" {
#endif

#define AVL_WALK_NAME   "avl"
#define AVL_WALK_DESC   "given any avl_tree_t *, forward walk all " \
                        "entries in tree"

extern int avl_walk_init(mdb_walk_state_t *);
extern int avl_walk_init_named(mdb_walk_state_t *wsp,
    const char *, const char *);
extern int avl_walk_init_checked(mdb_walk_state_t *wsp,
    const char *, const char *,
    int (*)(void *, uintptr_t, void *), void *);
extern int avl_walk_init_range(mdb_walk_state_t *wsp, uintptr_t, uintptr_t,
    const char *, const char *,
    int (*)(void *, uintptr_t, void *), void *);
extern int avl_walk_step(mdb_walk_state_t *);
extern void avl_walk_fini(mdb_walk_state_t *wsp);
extern int avl_walk_mdb(uintptr_t, mdb_walk_cb_t, void *);

#ifdef  __cplusplus
}
#endif

#endif  /* _MDB_AVL_H */
```

```
**********************************************************
   52787 Wed Aug 21 14:46:15 2013
new/usr/src/cmd/mdb/common/modules/genunix/gcore.c
3946 ::gcore
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Matthew Ahrens <mahrens@delphix.com>
**********************************************************
```

```
/*
 * This file and its contents are supplied under the terms of the
 * Common Development and Distribution License ("CDDL"), version 1.0.
 * You may only use this file in accordance with the terms of version
 * 1.0 of the CDDL.
 *
 * A full copy of the text of the CDDL should have accompanied this
 * source.  A copy of the CDDL is also available via the Internet at
 * http://www.illumos.org/license/CDDL.
 */
/*
 * Copyright (c) 2013 by Delphix. All rights reserved.
 */

/*
 * This file implements the mdb ::gcore command.  The command relies on the
 * libproc Pgcore function to actually generate the core file but we provide
 * our own ops vector to populate data required by Pgcore.  The ops vector
 * function implementations simulate the functionality implemented by procfs.
 * The data provided by some of the ops vector functions is not complete
 * (missing data is documented in function headers) but there is enough
 * information to generate a core file that can be loaded into mdb.
 *
 * Currently only x86 is supported!
 */

#ifndef _KMDB

/*
 * The kernel has its own definition of exit which has a different signature
 * than the user space definition.  This seems to be the standard way to deal
 * with this.
 */
#define exit kern_exit

#include <mdb/mdb_modapi.h>
#include <mdb/mdb_param.h>
#include <mdb/mdb_ks.h>
#include <mdb/mdb_ctf.h>
#include <mdb/mdb_debug.h>

#include <sys/class.h>
#include <sys/cpuvar.h>
#include <sys/proc.h>
#include <sys/cred_impl.h>
#include <sys/lgrp.h>
#include <sys/pool.h>
#include <sys/project.h>
#include <sys/regset.h>
#include <sys/schedctl.h>
#include <sys/session.h>
#include <sys/syscall.h>
#include <sys/task.h>
#include <sys/var.h>
#include <sys/privregs.h>
#include <sys/psw.h>
#include <sys/fault.h>
#include <sys/procfs.h>
#include <sys/sysmacros.h>
```



```
#include <sys/wait.h>
#include <vm/seg.h>
#include <vm/vpage.h>
#include <fs/proc/prdata.h>

#undef exit

#include <stdio.h>
#include <stdbool.h>
#include <string.h>
#include <libproc.h>

#include "avl.h"

#ifdef _LP64
#define LSPAN(type)     (P2ROUNDUP(sizeof (type), 16))
#else
#define LSPAN(type)     (P2ROUNDUP(sizeof (type), 8))
#endif

#define vpgtob(n)       ((n) * sizeof (struct vpage))

/* Macros to invoke gcore seg operations */
#define GSOP_INIT(_gs)          (_gs)->gs_ops->gsop_init((_gs))
#define GSOP_FINI(_gs)          (_gs)->gs_ops->gsop_fini((_gs))
#define GSOP_INCORE(_gs, _addr, _eaddr) \
        (_gs)->gs_ops->gsop_incore((_gs), (_addr), (_eaddr))
#define GSOP_GETPROT(_gs, _addr)        \
        (_gs)->gs_ops->gsop_getprot((_gs), (_addr))
#define GSOP_GETOFFSET(_gs, _addr)      \
        (_gs)->gs_ops->gsop_getoffset((_gs), (_addr))
#define GSOP_GETTYPE(_gs, _addr)        \
        (_gs)->gs_ops->gsop_gettype((_gs), (_addr))
#define GSOP_NAME(_gs, _name, _size)    \
        (_gs)->gs_ops->gsop_name((_gs), (_name), (_size))
#define GSOP_NORESERVE(_gs)             \
        (_gs)->gs_ops->gsop_noreserve((_gs))

#ifdef GCORE_DEBUG
#define dprintf(...)    mdb_printf(__VA_ARGS__)
#else
#define dprintf(...)
#endif

/* mdb versions of kernel structures used for ctf read calls */
typedef struct mdb_proc {
        uintptr_t       p_as;
        uintptr_t       p_brkbase;
        size_t          p_brksize;
        uintptr_t       p_usrstack;
        size_t          p_stksize;
        user_t          p_user;
        uintptr_t       p_agenttp;
        uintptr_t       p_tlist;
        uintptr_t       p_zone;
        uintptr_t       p_ldt;
        kcondvar_t      p_holdlwps;
        int             p_lwpcnt;
        uintptr_t       p_lwpdir;
        uint_t          p_lwpdir_sz;
        uintptr_t       p_cred;
        uint_t          p_flag;
        int             p_zombcnt;
        uintptr_t       p_pidp;
        pid_t           p_ppid;
        uintptr_t       p_pgidp;
```

```
126         uintptr_t       p_sessp;
127         uintptr_t       p_task;
128         uintptr_t       p_pool;
129         model_t         p_model;
130         char            p_wcode;
131         ushort_t        p_ldtlimit;
132         uintptr_t       p_exec;
133         uint_t          p_proc_flag;
134         ushort_t        p_pidflag;
135         k_sigset_t      p_ignore;
136         k_sigset_t      p_siginfo;
137         k_sigset_t      p_sig;
138         k_sigset_t      p_sigmask;
139         k_fltset_t      p_fltmask;
140         int             p_wdata;
141 } mdb_proc_t;

143 typedef struct mdb_kthread {
144         ushort_t        t_proc_flag;
145         uint_t          t_state;
146         lwpchan_t       t_lwpchan;
147         ushort_t        t_whystop;
148         uint8_t         t_dtrace_stop;
149         uintptr_t       t_forw;
150         uintptr_t       t_lwp;
151         id_t            t_tid;
152         short           t_sysnum;
153         pri_t           t_pri;
154         time_t          t_start;
155         id_t            t_cid;
156         uintptr_t       t_cpu;
157         int             t_bind_pset;
158         short           t_bind_cpu;
159         uintptr_t       t_lpl;
160         ushort_t        t_schedflag;
161         ushort_t        t_whatstop;
162         k_sigset_t      t_sig;
163         uintptr_t       t_schedctl;
164         k_sigset_t      t_hold;
165         hrtime_t        t_stoptime;
166 } mdb_kthread_t;

168 typedef struct mdb_seg {
169         uintptr_t       s_base;
170         size_t          s_size;
171         uintptr_t       s_ops;
172         uintptr_t       s_data;
173         uintptr_t       s_as;
174 } mdb_seg_t;

176 typedef struct mdb_as {
177         uintptr_t       a_proc;
178 } mdb_as_t;

180 typedef struct mdb_segvn_data {
181         uintptr_t       vp;
182         uint64_t        offset;
183         uint16_t        flags;
184         uint8_t         pageprot;
185         uint8_t         prot;
186         uintptr_t       amp;
187         struct vpage    *vpage;
188         uint64_t        anon_index;
189         uint8_t         type;
190 } mdb_segvn_data_t;
```



```
192 typedef struct mdb_vnode {
193         enum vtype      v_type;
194         uintptr_t       v_data;
195         uintptr_t       v_op;
196         uintptr_t       v_path;
197 } mdb_vnode_t;

199 typedef struct mdb_znode {
200         uint64_t        z_size;
201 } mdb_znode_t;

203 typedef struct mdb_tmpnode {
204         vattr_t         tn_attr;
205 } mdb_tmpnode_t;

207 typedef struct mdb_vnodeops {
208         uintptr_t       vnop_name;
209 } mdb_vnodeops_t;

211 typedef struct mdb_shm_data {
212         uintptr_t       shm_sptseg;
213 } mdb_shm_data_t;

215 typedef struct mdb_watched_page {
216         uintptr_t       wp_vaddr;
217         uint8_t         wp_oprot;
218 } mdb_watched_page_t;

220 typedef struct mdb_pid {
221         pid_t           pid_id;
222 } mdb_pid_t;

224 typedef struct mdb_sess {
225         uintptr_t       s_sidp;
226 } mdb_sess_t;

228 typedef struct mdb_task {
229         taskid_t        tk_tkid;
230         uintptr_t       tk_proj;
231 } mdb_task_t;

233 typedef struct mdb_kproject {
234         projid_t        kpj_id;
235 } mdb_kproject_t;

237 typedef struct mdb_zone {
238         zoneid_t        zone_id;
239         uintptr_t       zone_name;
240 } mdb_zone_t;

242 typedef struct mdb_sc_shared {
243         char            sc_sigblock;
244 } mdb_sc_shared_t;

246 typedef struct mdb_klwp {
247         uintptr_t       lwp_regs;
248         struct pcb      lwp_pcb;
249         uchar_t         lwp_asleep;
250         uchar_t         lwp_cursig;
251         uintptr_t       lwp_curinfo;
252         k_siginfo_t     lwp_siginfo;
253         stack_t         lwp_sigaltstack;
254         uintptr_t       lwp_oldcontext;
255         short           lwp_badpriv;
256         uintptr_t       lwp_ustack;
257         char            lwp_eosys;
```

```
} mdb_klwp_t;

typedef struct mdb_cpu {
        processorid_t   cpu_id;
} mdb_cpu_t;

typedef struct mdb_lpl {
        lgrp_id_t       lpl_lgrpid;
} mdb_lpl_t;

typedef struct mdb_sigqueue {
        k_siginfo_t     sq_info;
} mdb_sigqueue_t;

typedef struct mdb_pool {
        poolid_t        pool_id;
} mdb_pool_t;

typedef struct mdb_amp {
        uintptr_t       ahp;
} mdb_amp_t;

typedef struct mdb_anon_hdr {
        pgcnt_t         size;
        uintptr_t       array_chunk;
        int             flags;
} mdb_anon_hdr_t;

typedef struct mdb_anon {
        uintptr_t       an_vp;
        anoff_t         an_off;
} mdb_anon_t;

/* Used to construct a linked list of prmap_ts */
typedef struct prmap_node {
        struct prmap_node *next;
        prmap_t         m;
} prmap_node_t;

/* Fields common to psinfo_t and pstatus_t */
typedef struct pcommon {
        int             pc_nlwp;
        int             pc_nzomb;
        pid_t           pc_pid;
        pid_t           pc_ppid;
        pid_t           pc_pgid;
        pid_t           pc_sid;
        taskid_t        pc_taskid;
        projid_t        pc_projid;
        zoneid_t        pc_zoneid;
        char            pc_dmodel;
} pcommon_t;

/* AVL walk callback structures */
typedef struct read_maps_cbarg {
        mdb_proc_t      *p;
        uintptr_t       brkseg;
        uintptr_t       stkseg;
        prmap_node_t    *map_head;
        prmap_node_t    *map_tail;
        int             map_len;
} read_maps_cbarg_t;

typedef struct as_segat_cbarg {
        uintptr_t       addr;
        uintptr_t       res;
```



```
} as_segat_cbarg_t;

typedef struct getwatchprot_cbarg {
        uintptr_t       wp_vaddr;
        mdb_watched_page_t wp;
        boolean_t       found;
} getwatchprot_cbarg_t;

struct gcore_segops;
typedef struct gcore_seg {
        mdb_seg_t       *gs_seg;
        void            *gs_data;
        struct gcore_segops *gs_ops;
} gcore_seg_t;

/* Callback function type for processing lwp entries */
typedef int (*lwp_callback_t)(mdb_proc_t *, lwpent_t *, void *);

/* Private data */
static uintptr_t gcore_segvn_ops;
static priv_impl_info_t prinfo;
static sclass_t *gcore_sclass;
static uintptr_t gcore_kas;
static boolean_t gcore_initialized = B_FALSE;

typedef int (*gsop_init_t)(gcore_seg_t *);
typedef void (*gsop_fini_t)(gcore_seg_t *);
typedef u_offset_t (*gsop_incore_t)(gcore_seg_t *, u_offset_t, u_offset_t);
typedef uint_t (*gsop_getprot_t)(gcore_seg_t *, u_offset_t);
typedef int (*gsop_getoffset_t)(gcore_seg_t *, u_offset_t);
typedef void (*gsop_name_t)(gcore_seg_t *, char *name, size_t size);
typedef int (*gsop_gettype_t)(gcore_seg_t *, u_offset_t);
typedef boolean_t (*gsop_noreserve_t)(gcore_seg_t *);

typedef struct gcore_segops {
        gsop_init_t             gsop_init;
        gsop_fini_t             gsop_fini;
        gsop_incore_t           gsop_incore;
        gsop_getprot_t          gsop_getprot;
        gsop_getoffset_t        gsop_getoffset;
        gsop_name_t             gsop_name;
        gsop_gettype_t          gsop_gettype;
        gsop_noreserve_t        gsop_noreserve;
} gcore_segops_t;

static void map_list_free(prmap_node_t *);
static uintptr_t gcore_prchoose(mdb_proc_t *);

/*
 * Segvn ops
 */
static int gsvn_init(gcore_seg_t *);
static void gsvn_fini(gcore_seg_t *);
static u_offset_t gsvn_incore(gcore_seg_t *, u_offset_t, u_offset_t);
static uint_t gsvn_getprot(gcore_seg_t *, u_offset_t);
static int gsvn_getoffset(gcore_seg_t *, u_offset_t);
static void gsvn_name(gcore_seg_t *, char *, size_t);
static int gsvn_gettype(gcore_seg_t *, u_offset_t);
static boolean_t gsvn_noreserve(gcore_seg_t *);

static gcore_segops_t gsvn_ops = {
        .gsop_init              = gsvn_init,
        .gsop_fini              = gsvn_fini,
        .gsop_incore            = gsvn_incore,
        .gsop_getprot           = gsvn_getprot,
        .gsop_getoffset         = gsvn_getoffset,
```

```
390         .gsop_name              = gsvn_name,
391         .gsop_gettype           = gsvn_gettype,
392         .gsop_noreserve         = gsvn_noreserve
393 };

395 static int
396 gsvn_init(gcore_seg_t *gs)
397 {
398         mdb_seg_t               *seg = gs->gs_seg;
399         mdb_segvn_data_t        *svd = NULL;
400         struct vpage            *vpage = NULL;
401         size_t                  nvpage = 0;

403         if (seg->s_data != NULL) {
404                 svd = mdb_alloc(sizeof (*svd), UM_SLEEP);
405                 if (mdb_ctf_vread(svd, "segvn_data_t", "mdb_segvn_data_t",
406                     seg->s_data, 0) == -1) {
407                         goto error;
408                 }

410                 if (svd->pageprot != 0) {
411                         nvpage = seg_pages(seg);
412                         dprintf("vpage count: %d\n", nvpage);

414                         vpage = mdb_alloc(vpgtob(nvpage), UM_SLEEP);
415                         if (mdb_vread(vpage, vpgtob(nvpage),
416                             (uintptr_t)svd->vpage) != vpgtob(nvpage)) {
417                                 mdb_warn("Failed to read vpages from %p\n",
418                                     svd->vpage);
419                                 goto error;
420                         }

422                         svd->vpage = vpage;
423                 } else {
424                         svd->vpage = NULL;
425                 }
426                 gs->gs_data = svd;
427         } else {
428                 gs->gs_data = NULL;
429         }

431         return (0);

433 error:
434         mdb_free(vpage, vpgtob(nvpage));
435         mdb_free(svd, sizeof (*svd));
436         return (-1);
437 }

439 /*ARGSUSED*/
440 static int
441 gsvn_getoffset(gcore_seg_t *gs, u_offset_t addr)
442 {
443         mdb_segvn_data_t        *svd = gs->gs_data;
444         mdb_seg_t               *seg = gs->gs_seg;

446         return (svd->offset + (uintptr_t)(addr - seg->s_base));
447 }

449 static void
450 gsvn_name(gcore_seg_t *gs, char *name, size_t size)
451 {
452         mdb_segvn_data_t        *svd = gs->gs_data;

454         name[0] = '\0';
455         if (svd->vp != 0) {
```



```
456                 mdb_seg_t       *seg = gs->gs_seg;
457                 mdb_as_t        as;
458                 mdb_proc_t      p;
459                 mdb_vnode_t     vn;

461                 if (mdb_ctf_vread(&vn, "vnode_t", "mdb_vnode_t", svd->vp, 0)
462                     == -1) {
463                         return;
464                 }

466                 if (mdb_ctf_vread(&as, "struct as", "mdb_as_t", seg->s_as, 0)
467                     == -1) {
468                         return;
469                 }

471                 if (mdb_ctf_vread(&p, "proc_t", "mdb_proc_t", as.a_proc, 0)
472                     == -1) {
473                         return;
474                 }

476                 if (vn.v_type == VREG && svd->vp == p.p_exec) {
477                         (void) strncpy(name, "a.out", size);
478                 }

480                 /*
481                  * procfs has more logic here to construct a name using
482                  * vfs/vnode identifiers but didn't seem worthwhile to add
483                  * here.
484                  */
485         }
486 }

488 /*ARGSUSED*/
489 static int
490 gsvn_gettype(gcore_seg_t *gs, u_offset_t addr)
491 {
492         return (0);
493 }

495 static void
496 gsvn_fini(gcore_seg_t *gs)
497 {
498         mdb_segvn_data_t        *svd = gs->gs_data;

500         if (svd != NULL) {
501                 if (svd->vpage != NULL) {
502                         size_t nvpage = seg_pages(gs->gs_seg);

504                         mdb_free(svd->vpage, vpgtob(nvpage));
505                 }
506                 mdb_free(svd, sizeof (*svd));
507         }
508 }

510 static boolean_t
511 gsvn_noreserve(gcore_seg_t *gs)
512 {
513         mdb_segvn_data_t        *svd = gs->gs_data;

515         if (svd == NULL) {
516                 return (B_FALSE);
517         }

519         if (svd->flags & MAP_NORESERVE) {
520                 mdb_vnode_t vn;
```

```
522                 if (svd->vp == 0) {
523                         return (B_TRUE);
524                 }

526                 if (mdb_ctf_vread(&vn, "vnode_t", "mdb_vnode_t",
527                     svd->vp, 0) == -1) {
528                         return (B_FALSE);
529                 }

531                 if (vn.v_type != VREG) {
532                         return (B_TRUE);
533                 }
534         }

536         return (B_FALSE);
537 }

539 static uintptr_t
540 gcore_anon_get_ptr(uintptr_t ah_addr, ulong_t an_idx)
541 {
542         mdb_anon_hdr_t  ah;
543         uintptr_t       anon_addr;
544         uintptr_t       anon_ptr;

546         if (mdb_ctf_vread(&ah, "struct anon_hdr", "mdb_anon_hdr_t", ah_addr,
547             0) == -1) {
548                 return (0);
549         }

551         /*
552          * Single level case.
553          */
554         if ((ah.size <= ANON_CHUNK_SIZE) || (ah.flags & ANON_ALLOC_FORCE)) {
555                 anon_addr = ah.array_chunk + (sizeof (anon_ptr) * an_idx);
556                 if (mdb_vread(&anon_ptr, sizeof (anon_ptr), anon_addr) !=
557                     sizeof (anon_ptr)) {
558                         mdb_warn("Failed to read anon_ptr from %p (1 level)\n",
559                             anon_addr);
560                         return (0);
561                 }

563                 return (anon_ptr & ANON_PTRMASK);
564         }

566         /*
567          * 2 level case.
568          */
569         anon_addr = ah.array_chunk + (sizeof (anon_ptr) *
570             (an_idx >> ANON_CHUNK_SHIFT));

572         if (mdb_vread(&anon_ptr, sizeof (anon_ptr), anon_addr) !=
573             sizeof (anon_ptr)) {
574                 mdb_warn("Failed to read anon_ptr from %p (2a level)\n",
575                     anon_addr);
576                 return (0);
577         }

579         if (anon_ptr == 0) {
580                 return (0);
581         }

583         anon_addr = anon_ptr + (sizeof (anon_ptr) *
584             (an_idx & ANON_CHUNK_OFF));
585         if (mdb_vread(&anon_ptr, sizeof (anon_ptr), anon_addr) !=
586             sizeof (anon_ptr)) {
587                 mdb_warn("Failed to read anon_ptr from %p (2b level)\n",
```



```
588                     anon_addr);
589                 return (0);
590         }

592         return (anon_ptr & ANON_PTRMASK);
593 }

595 static void
596 gcore_anon_get(uintptr_t ahp, ulong_t an_index, uintptr_t *vp, u_offset_t *off)
597 {
598         mdb_anon_t      anon;
599         uintptr_t       ap;

601         ap = gcore_anon_get_ptr(ahp, an_index);
602         if (ap != 0) {
603                 if (mdb_ctf_vread(&anon, "struct anon", "mdb_anon_t", ap, 0) ==
604                     -1) {
605                         return;
606                 }

608                 *vp = anon.an_vp;
609                 *off = anon.an_off;
610         } else {
611                 *vp = 0;
612                 *off = 0;
613         }
614 }

616 static u_offset_t
617 gsvn_incore(gcore_seg_t *gs, u_offset_t addr, u_offset_t eaddr)
618 {
619         mdb_segvn_data_t        *svd = gs->gs_data;
620         mdb_seg_t               *seg = gs->gs_seg;
621         mdb_amp_t               amp;
622         u_offset_t              offset;
623         uintptr_t               vp;
624         size_t                  p, ep;

626         if (svd->amp != 0 && mdb_ctf_vread(&amp, "amp_t", "mdb_amp_t", svd->amp,
627             0) == -1) {
628                 return (eaddr);
629         }

631         p = seg_page(seg, addr);
632         ep = seg_page(seg, eaddr);
633         for (; p < ep; p++, addr += PAGESIZE) {
634                 /* First check the anon map */
635                 if (svd->amp != 0) {
636                         gcore_anon_get(amp.ahp, svd->anon_index + p, &vp,
637                             &offset);
638                         if (vp != 0 && mdb_page_lookup(vp, offset) != 0) {
639                                 break;
640                         }
641                 }

643                 /* Now check the segment's vnode */
644                 vp = svd->vp;
645                 offset = svd->offset + (addr - gs->gs_seg->s_base);
646                 if (mdb_page_lookup(vp, offset) != 0) {
647                         break;
648                 }

650                 dprintf("amp: %p vp: %p addr: %p offset: %p not in core!\n",
651                     svd->amp, svd->vp, addr, offset);
652         }
```

```
654         return (addr);
655 }

657 static uint_t
658 gsvn_getprot(gcore_seg_t *gs, u_offset_t addr)
659 {
660         mdb_segvn_data_t        *svd = gs->gs_data;
661         mdb_seg_t               *seg = gs->gs_seg;

663         if (svd->pageprot == 0) {
664                 return (svd->prot);
665         }

667         dprintf("addr: %p pgno: %p\n", addr, seg_page(seg, addr));
668         return (VPP_PROT(&svd->vpage[seg_page(seg, addr)]));
669 }

671 /*
672  * Helper functions for constructing the process address space maps.
673  */
674 /*ARGSUSED*/
675 static int
676 as_segat_cb(uintptr_t seg_addr, const void *aw_buff, void *arg)
677 {
678         as_segat_cbarg_t *as_segat_arg = arg;
679         mdb_seg_t       seg;

681         if (mdb_ctf_vread(&seg, "struct seg", "mdb_seg_t", seg_addr, 0) == -1) {
682                 return (WALK_ERR);
683         }

685         if (as_segat_arg->addr < seg.s_base) {
686                 return (WALK_NEXT);
687         }

689         if (as_segat_arg->addr >= seg.s_base + seg.s_size) {
690                 return (WALK_NEXT);
691         }

693         as_segat_arg->res = seg_addr;
694         return (WALK_DONE);
695 }

697 /*
698  * Find a segment containing addr.
699  */
700 static uintptr_t
701 gcore_as_segat(uintptr_t as_addr, uintptr_t addr)
702 {
703         as_segat_cbarg_t as_segat_arg;
704         uintptr_t       segtree_addr;

706         as_segat_arg.addr = addr;
707         as_segat_arg.res = 0;

709         segtree_addr = as_addr + mdb_ctf_offsetof_by_name("struct as",
710             "a_segtree");
711         (void) avl_walk_mdb(segtree_addr, as_segat_cb, &as_segat_arg);

713         return (as_segat_arg.res);
714 }

716 static uintptr_t
717 gcore_break_seg(mdb_proc_t *p)
718 {
719         uintptr_t addr = p->p_brkbase;
```



```
721         if (p->p_brkbase != 0)
722                 addr += p->p_brksize - 1;

724         return (gcore_as_segat(p->p_as, addr));
725 }

727 /* ISA dependent function. */
728 static uintptr_t
729 gcore_prgetstackbase(mdb_proc_t *p)
730 {
731         return (p->p_usrstack - p->p_stksize);
732 }

734 static u_offset_t
735 gcore_vnode_size(uintptr_t vnode_addr)
736 {
737         mdb_vnode_t     vnode;
738         mdb_vnodeops_t  vnodeops;
739         char            vops_name[128];

741         if (mdb_ctf_vread(&vnode, "vnode_t", "mdb_vnode_t", vnode_addr, 0) ==
742             -1) {
743                 return (-1);
744         }

746         if (mdb_ctf_vread(&vnodeops, "vnodeops_t", "mdb_vnodeops_t",
747             vnode.v_op, 0) == -1) {
748                 return (-1);
749         }

751         if (mdb_readstr(vops_name, sizeof (vops_name), vnodeops.vnop_name) ==
752             -1) {
753                 mdb_warn("Failed to read vnop_name from %p\n",
754                     vnodeops.vnop_name);
755                 return (-1);
756         }

758         if (strcmp(vops_name, "zfs") == 0) {
759                 mdb_znode_t     znode;

761                 if (mdb_ctf_vread(&znode, "znode_t", "mdb_znode_t",
762                     vnode.v_data, 0) == -1) {
763                         return (-1);
764                 }
765                 return (znode.z_size);
766         }

768         if (strcmp(vops_name, "tmpfs") == 0) {
769                 mdb_tmpnode_t   tnode;

771                 if (mdb_ctf_vread(&tnode, "struct tmpnode", "mdb_tmpnode_t",
772                     vnode.v_data, 0) == -1)  {
773                         return (-1);
774                 }
775                 return (tnode.tn_attr.va_size);
776         }

778         /* Unknown file system type. */
779         mdb_warn("Unknown fs type: %s\n", vops_name);
780         return (-1);
781 }

783 static uint64_t
784 gcore_pr_getsegsize(mdb_seg_t *seg)
785 {
```

```
786         uint64_t size = seg->s_size;

788         if (seg->s_ops == gcore_segvn_ops) {
789                 mdb_segvn_data_t svd;

791                 if (mdb_ctf_vread(&svd, "segvn_data_t", "mdb_segvn_data_t",
792                     seg->s_data, 0) == -1) {
793                         return (-1);
794                 }

796                 if (svd.vp != 0) {
797                         u_offset_t fsize;
798                         u_offset_t offset;

800                         fsize = gcore_vnode_size(svd.vp);
801                         if (fsize == -1) {
802                                 return (-1);
803                         }
804                         offset = svd.offset;

806                         if (fsize < offset) {
807                                 fsize = 0;
808                         } else {
809                                 fsize -= offset;
810                         }

812                         fsize = roundup(fsize, PAGESIZE);
813                 }

815                 return (size);
816         }

818         return (size);
819 }

821 /*ARGSUSED*/
822 static int
823 gcore_getwatchprot_cb(uintptr_t node_addr, const void *aw_buff, void *arg)
824 {
825         getwatchprot_cbarg_t    *cbarg = arg;

827         if (mdb_ctf_vread(&cbarg->wp, "struct watched_page",
828             "mdb_watched_page_t", node_addr, 0) == -1) {
829                 return (WALK_ERR);
830         }

832         if (cbarg->wp.wp_vaddr == cbarg->wp_vaddr) {
833                 cbarg->found = B_TRUE;
834                 return (WALK_DONE);
835         }

837         return (WALK_NEXT);
838 }

840 static void
841 gcore_getwatchprot(uintptr_t as_addr, u_offset_t addr, uint_t *prot)
842 {
843         getwatchprot_cbarg_t    cbarg;
844         uintptr_t               wp_addr;

846         cbarg.wp_vaddr = (uintptr_t)addr & (uintptr_t)PAGEMASK;
847         cbarg.found = B_FALSE;

849         wp_addr = as_addr + mdb_ctf_offsetof_by_name("struct as", "a_wpage");
850         (void) avl_walk_mdb(wp_addr, gcore_getwatchprot_cb, &cbarg);
```



```
852         if (cbarg.found) {
853                 *prot = cbarg.wp.wp_oprot;
854         }
855 }

857 static u_offset_t
858 gcore_pr_nextprot(gcore_seg_t *gs, u_offset_t *saddrp, u_offset_t eaddr,
859     uint_t *protp)
860 {
861         uint_t          prot, nprot;
862         u_offset_t      addr = *saddrp;
863         uintptr_t       as_addr = gs->gs_seg->s_as;
864         int             noreserve = 0;

866         noreserve = GSOP_NORESERVE(gs);
867         dprintf("addr: %p noreserve: %d\n", addr, noreserve);

869         if (noreserve) {
870                 addr = GSOP_INCORE(gs, addr, eaddr);
871                 if (addr == eaddr) {
872                         prot = 0;
873                         *saddrp = addr;
874                         goto out;
875                 }
876         }

878         prot = GSOP_GETPROT(gs, addr);
879         gcore_getwatchprot(as_addr, addr, &prot);
880         *saddrp = addr;

882         for (addr += PAGESIZE; addr < eaddr; addr += PAGESIZE) {
883                 /* Discontinuity */
884                 if (noreserve && GSOP_INCORE(gs, addr, eaddr) != addr) {
885                         goto out;
886                 }

888                 nprot = GSOP_GETPROT(gs, addr);
889                 gcore_getwatchprot(as_addr, addr, &nprot);

891                 if (nprot != prot) {
892                         break;
893                 }
894         }

896 out:
897         *protp = prot;
898         return (addr);
899 }

901 /*
902  * Get the page protection for the given start address.
903  *   - saddrp: in - start address
904  *             out - contains address of first in core page
905  *   - naddrp: out - address of next in core page that has different protection
906  *   - eaddr: in - end address
907  */
908 static uint_t
909 gcore_pr_getprot(gcore_seg_t *gs, u_offset_t *saddrp, u_offset_t *naddrp,
910     u_offset_t eaddr)
911 {
912         u_offset_t      naddr;
913         uint_t          prot;

915         dprintf("seg: %p saddr: %p eaddr: %p\n",
916             gs->gs_seg, *saddrp, eaddr);
```

```
        naddr = gcore_pr_nextprot(gs, saddrp, eaddr, &prot);

        dprintf("seg: %p saddr: %p naddr: %p eaddr: %p\n",
            gs->gs_seg, *saddrp, naddr, eaddr);

        *naddrp = naddr;
        return (prot);
}

static gcore_seg_t *
gcore_seg_create(mdb_seg_t *seg)
{
        gcore_seg_t     *gs;

        gs = mdb_alloc(sizeof (*gs), UM_SLEEP);
        gs->gs_seg = seg;
        if (seg->s_ops == gcore_segvn_ops) {
                gs->gs_ops = &gsvn_ops;
        } else {
                mdb_warn("Unhandled segment type, ops: %p\n", seg->s_ops);
                goto error;
        }

        if (GSOP_INIT(gs) != 0) {
                goto error;
        }

        return (gs);

error:
        mdb_free(gs, sizeof (*gs));
        return (NULL);
}

static void
gcore_seg_destroy(gcore_seg_t *gs)
{
        GSOP_FINI(gs);
        mdb_free(gs, sizeof (*gs));
}

/*ARGSUSED*/
static int
read_maps_cb(uintptr_t seg_addr, const void *aw_buff, void *arg)
{
        read_maps_cbarg_t       *cbarg = arg;
        mdb_segvn_data_t        svd;
        mdb_seg_t               s;
        mdb_seg_t               *seg;
        uint_t                  prot;
        gcore_seg_t             *gs;
        uintptr_t               eaddr;
        u_offset_t              saddr, baddr;
        prmap_node_t            *mnode;
        prmap_t                 *mp;

        if (mdb_ctf_vread(&s, "struct seg", "mdb_seg_t", seg_addr, 0) == -1) {
                return (WALK_ERR);
        }
        seg = &s;
        eaddr = seg->s_base + gcore_pr_getsegsize(seg);

        if ((gs = gcore_seg_create(seg)) == NULL) {
                mdb_warn("gcore_seg_create failed!\n");
                return (WALK_ERR);
        }
```



```
        /*
         * Iterate from the base of the segment to its end, allocating a new
         * prmap_node at each address boundary (baddr) between ranges that
         * have different virtual memory protections.
         */
        for (saddr = seg->s_base; saddr < eaddr; saddr = baddr) {
                prot = gcore_pr_getprot(gs, &saddr, &baddr, eaddr);
                if (saddr == eaddr) {
                        break;
                }

                mnode = mdb_alloc(sizeof (*mnode), UM_SLEEP);
                mnode->next = NULL;
                mp = &mnode->m;

                if (cbarg->map_head == NULL) {
                        cbarg->map_head = cbarg->map_tail = mnode;
                } else {
                        cbarg->map_tail->next = mnode;
                        cbarg->map_tail = mnode;
                }
                cbarg->map_len++;

                mp->pr_vaddr = (uintptr_t)saddr;
                mp->pr_size = baddr - saddr;
                mp->pr_offset = GSOP_GETOFFSET(gs, saddr);
                mp->pr_mflags = 0;
                if (prot & PROT_READ)
                        mp->pr_mflags |= MA_READ;
                if (prot & PROT_WRITE)
                        mp->pr_mflags |= MA_WRITE;
                if (prot & PROT_EXEC)
                        mp->pr_mflags |= MA_EXEC;
                if (GSOP_GETTYPE(gs, saddr) & MAP_SHARED)
                        mp->pr_mflags |= MA_SHARED;
                if (GSOP_GETTYPE(gs, saddr) & MAP_NORESERVE)
                        mp->pr_mflags |= MA_NORESERVE;
                if (seg->s_ops == gcore_segvn_ops) {
                        if (mdb_ctf_vread(&svd, "segvn_data_t",
                            "mdb_segvn_data_t", seg->s_data, 0) == 0 &&
                            svd.vp == NULL) {
                                mp->pr_mflags |= MA_ANON;
                        }
                }
                if (seg_addr == cbarg->brkseg)
                        mp->pr_mflags |= MA_BREAK;
                else if (seg_addr == cbarg->stkseg)
                        mp->pr_mflags |= MA_STACK;

                mp->pr_pagesize = PAGESIZE;

                /*
                 * Manufacture a filename for the "object" dir.
                 */
                GSOP_NAME(gs, mp->pr_mapname, sizeof (mp->pr_mapname));
        }

        gcore_seg_destroy(gs);

        return (0);
}

/*
 * Helper functions for retrieving process and lwp state.
 */
```

```
1050 static int
1051 pcommon_init(mdb_proc_t *p, pcommon_t *pc)
1052 {
1053         mdb_pid_t       pid;
1054         mdb_sess_t      sess;
1055         mdb_task_t      task;
1056         mdb_kproject_t  proj;
1057         mdb_zone_t      zone;

1059         pc->pc_nlwp = p->p_lwpcnt;
1060         pc->pc_nzomb = p->p_zombcnt;

1062         if (mdb_ctf_vread(&pid, "struct pid", "mdb_pid_t", p->p_pidp, 0) ==
1063             -1) {
1064                 return (-1);
1065         }
1066         pc->pc_pid = pid.pid_id;
1067         pc->pc_ppid = p->p_ppid;

1069         if (mdb_ctf_vread(&pid, "struct pid", "mdb_pid_t", p->p_pgidp, 0) ==
1070             -1) {
1071                 return (-1);
1072         }
1073         pc->pc_pgid = pid.pid_id;

1075         if (mdb_ctf_vread(&sess, "sess_t", "mdb_sess_t", p->p_sessp, 0) ==
1076             -1) {
1077                 return (-1);
1078         }
1079         if (mdb_ctf_vread(&pid, "struct pid", "mdb_pid_t", sess.s_sidp, 0) ==
1080             -1) {
1081                 return (-1);
1082         }
1083         pc->pc_sid = pid.pid_id;

1085         if (mdb_ctf_vread(&task, "task_t", "mdb_task_t", p->p_task, 0) == -1) {
1086                 return (-1);
1087         }
1088         pc->pc_taskid = task.tk_tkid;

1090         if (mdb_ctf_vread(&proj, "kproject_t", "mdb_kproject_t", task.tk_proj,
1091             0) == -1) {
1092                 return (-1);
1093         }
1094         pc->pc_projid = proj.kpj_id;

1096         if (mdb_ctf_vread(&zone, "zone_t", "mdb_zone_t", p->p_zone, 0) == -1) {
1097                 return (-1);
1098         }
1099         pc->pc_zoneid = zone.zone_id;

1101         switch (p->p_model) {
1102         case DATAMODEL_ILP32:
1103                 pc->pc_dmodel = PR_MODEL_ILP32;
1104                 break;
1105         case DATAMODEL_LP64:
1106                 pc->pc_dmodel = PR_MODEL_LP64;
1107                 break;
1108         }

1110         return (0);
1111 }

1113 static uintptr_t
1114 gcore_prchoose(mdb_proc_t *p)
1115 {
```



```
1116         mdb_kthread_t   kthr;
1117         mdb_kthread_t   *t = &kthr;
1118         ushort_t        t_istop_whystop = 0;
1119         ushort_t        t_istop_whatstop = 0;
1120         uintptr_t       t_addr = NULL;
1121         uintptr_t       t_onproc = NULL; // running on processor
1122         uintptr_t       t_run = NULL;    // runnable, on disp queue
1123         uintptr_t       t_sleep = NULL;  // sleeping
1124         uintptr_t       t_susp = NULL;   // suspended stop
1125         uintptr_t       t_jstop = NULL;  // jobcontrol stop, w/o directed stop
1126         uintptr_t       t_jdstop = NULL; // jobcontrol stop with directed stop
1127         uintptr_t       t_req = NULL;    // requested stop
1128         uintptr_t       t_istop = NULL;  // event-of-interest stop
1129         uintptr_t       t_dtrace = NULL; // DTrace stop

1131         /*
1132          * If the agent lwp exists, it takes precedence over all others.
1133          */
1134         if ((t_addr = p->p_agenttp) != NULL) {
1135                 return (t_addr);
1136         }

1138         if ((t_addr = p->p_tlist) == NULL) /* start at the head of the list */
1139                 return (t_addr);
1140         do {            /* for each lwp in the process */
1141                 if (mdb_ctf_vread(&kthr, "kthread_t", "mdb_kthread_t",
1142                     t_addr, 0) == -1) {
1143                         return (0);
1144                 }

1146                 if (VSTOPPED(t)) {      /* virtually stopped */
1147                         if (t_req == NULL)
1148                                 t_req = t_addr;
1149                         continue;
1150                 }

1152                 switch (t->t_state) {
1153                 default:
1154                         return (0);
1155                 case TS_SLEEP:
1156                         if (t_sleep == NULL)
1157                                 t_sleep = t_addr;
1158                         break;
1159                 case TS_RUN:
1160                 case TS_WAIT:
1161                         if (t_run == NULL)
1162                                 t_run = t_addr;
1163                         break;
1164                 case TS_ONPROC:
1165                         if (t_onproc == NULL)
1166                                 t_onproc = t_addr;
1167                         break;
1168                         /*
1169                          * Threads in the zombie state have the lowest
1170                          * priority when selecting a representative lwp.
1171                          */
1172                 case TS_ZOMB:
1173                         break;
1174                 case TS_STOPPED:
1175                         switch (t->t_whystop) {
1176                         case PR_SUSPENDED:
1177                                 if (t_susp == NULL)
1178                                         t_susp = t_addr;
1179                                 break;
1180                         case PR_JOBCONTROL:
1181                                 if (t->t_proc_flag & TP_PRSTOP) {
```

```
1182                                    if (t_jdstop == NULL)
1183                                            t_jdstop = t_addr;
1184                            } else {
1185                                    if (t_jstop == NULL)
1186                                            t_jstop = t_addr;
1187                            }
1188                            break;
1189                    case PR_REQUESTED:
1190                            if (t->t_dtrace_stop && t_dtrace == NULL)
1191                                    t_dtrace = t_addr;
1192                            else if (t_req == NULL)
1193                                    t_req = t_addr;
1194                            break;
1195                    case PR_SYSENTRY:
1196                    case PR_SYSEXIT:
1197                    case PR_SIGNALLED:
1198                    case PR_FAULTED:
1199                            /*
1200                             * Make an lwp calling exit() be the
1201                             * last lwp seen in the process.
1202                             */
1203                            if (t_istop == NULL ||
1204                                (t_istop_whystop == PR_SYSENTRY &&
1205                                t_istop_whatstop == SYS_exit)) {
1206                                    t_istop = t_addr;
1207                                    t_istop_whystop = t->t_whystop;
1208                                    t_istop_whatstop = t->t_whatstop;
1209                            }
1210                            break;
1211                    case PR_CHECKPOINT:     /* can't happen? */
1212                            break;
1213                    default:
1214                            return (0);
1215                    }
1216                    break;
1217            }
1218        } while ((t_addr = t->t_forw) != p->p_tlist);

1220        if (t_onproc)
1221                t_addr = t_onproc;
1222        else if (t_run)
1223                t_addr = t_run;
1224        else if (t_sleep)
1225                t_addr = t_sleep;
1226        else if (t_jstop)
1227                t_addr = t_jstop;
1228        else if (t_jdstop)
1229                t_addr = t_jdstop;
1230        else if (t_istop)
1231                t_addr = t_istop;
1232        else if (t_dtrace)
1233                t_addr = t_dtrace;
1234        else if (t_req)
1235                t_addr = t_req;
1236        else if (t_susp)
1237                t_addr = t_susp;
1238        else            /* TS_ZOMB */
1239                t_addr = p->p_tlist;

1241        return (t_addr);
1242 }

1244 /*
1245  * Fields not populated:
1246  *   - pr_stype
1247  *   - pr_oldpri
```



```
1248  *   - pr_nice
1249  *   - pr_time
1250  *   - pr_pctcpu
1251  *   - pr_cpu
1252  */
1253 static int
1254 gcore_prgetlwpsinfo(uintptr_t t_addr, mdb_kthread_t *t, lwpsinfo_t *psp)
1255 {
1256         char            c, state;
1257         mdb_cpu_t       cpu;
1258         mdb_lpl_t       lgrp;
1259         uintptr_t       str_addr;

1261         bzero(psp, sizeof (*psp));

1263         psp->pr_flag = 0;       /* lwpsinfo_t.pr_flag is deprecated */
1264         psp->pr_lwpid = t->t_tid;
1265         psp->pr_addr = t_addr;
1266         psp->pr_wchan = (uintptr_t)t->t_wchan;

1268         /* map the thread state enum into a process state enum */
1269         state = VSTOPPED(t) ? TS_STOPPED : t->t_state;
1270         switch (state) {
1271         case TS_SLEEP:          state = SSLEEP;         c = 'S';        break;
1272         case TS_RUN:            state = SRUN;           c = 'R';        break;
1273         case TS_ONPROC:         state = SONPROC;        c = 'O';        break;
1274         case TS_ZOMB:           state = SZOMB;          c = 'Z';        break;
1275         case TS_STOPPED:        state = SSTOP;          c = 'T';        break;
1276         case TS_WAIT:           state = SWAIT;          c = 'W';        break;
1277         default:                state = 0;              c = '?';        break;
1278         }
1279         psp->pr_state = state;
1280         psp->pr_sname = c;
1281         psp->pr_syscall = t->t_sysnum;
1282         psp->pr_pri = t->t_pri;
1283         psp->pr_start.tv_sec = t->t_start;
1284         psp->pr_start.tv_nsec = 0L;

1286         str_addr = (uintptr_t)gcore_sclass[t->t_cid].cl_name;
1287         if (mdb_readstr(psp->pr_clname, sizeof (psp->pr_clname) - 1, str_addr)
1288             == -1) {
1289                 mdb_warn("Failed to read string from %p\n", str_addr);
1290                 return (-1);
1291         }
1292         bzero(psp->pr_name, sizeof (psp->pr_name));

1294         if (mdb_ctf_vread(&cpu, "struct cpu", "mdb_cpu_t", t->t_cpu, 0) == -1) {
1295                 return (-1);
1296         }
1297         psp->pr_onpro = cpu.cpu_id;
1298         psp->pr_bindpro = t->t_bind_cpu;
1299         psp->pr_bindpset = t->t_bind_pset;

1301         if (mdb_ctf_vread(&lgrp, "lpl_t", "mdb_lpl_t", t->t_lpl, 0) == -1) {
1302                 return (-1);
1303         }
1304         psp->pr_lgrp = lgrp.lpl_lgrpid;

1306         return (0);
1307 }

1309 /*ARGSUSED*/
1310 static int
1311 gcore_lpsinfo_cb(mdb_proc_t *p, lwpent_t *lwent, void *data)
1312 {
1313         lwpsinfo_t      *lpsinfo = data;
```

```
1314         uintptr_t       t_addr = (uintptr_t)lwent->le_thread;
1315         mdb_kthread_t   kthrd;

1317         if (t_addr != 0) {
1318                 if (mdb_ctf_vread(&kthrd, "kthread_t", "mdb_kthread_t", t_addr,
1319                     0) == -1) {
1320                         return (-1);
1321                 }
1322                 return (gcore_prgetlwpsinfo(t_addr, &kthrd, lpsinfo));
1323         }

1325         bzero(lpsinfo, sizeof (*lpsinfo));
1326         lpsinfo->pr_lwpid = lwent->le_lwpid;
1327         lpsinfo->pr_state = SZOMB;
1328         lpsinfo->pr_sname = 'Z';
1329         lpsinfo->pr_start.tv_sec = lwent->le_start;
1330         lpsinfo->pr_bindpro = PBIND_NONE;
1331         lpsinfo->pr_bindpset = PS_NONE;
1332         return (0);
1333 }

1335 static void
1336 gcore_schedctl_finish_sigblock(mdb_kthread_t *t)
1337 {
1338         mdb_sc_shared_t td;
1339         mdb_sc_shared_t *tdp;

1341         if (t->t_schedctl == NULL) {
1342                 return;
1343         }

1345         if (mdb_ctf_vread(&td, "sc_shared_t", "mdb_sc_shared_t", t->t_schedctl,
1346             0) == -1) {
1347                 return;
1348         }
1349         tdp = &td;

1351         if (tdp->sc_sigblock) {
1352                 t->t_hold.__sigbits[0] = FILLSET0 & ~CANTMASK0;
1353                 t->t_hold.__sigbits[1] = FILLSET1 & ~CANTMASK1;
1354                 t->t_hold.__sigbits[2] = FILLSET2 & ~CANTMASK2;
1355                 tdp->sc_sigblock = 0;
1356         }
1357 }

1359 static void
1360 gcore_prgetaction(mdb_proc_t *p, user_t *up, uint_t sig, struct sigaction *sp)
1361 {
1362         int nsig = NSIG;

1364         bzero(sp, sizeof (*sp));

1366         if (sig != 0 && (unsigned)sig < nsig) {
1367                 sp->sa_handler = up->u_signal[sig-1];
1368                 prassignset(&sp->sa_mask, &up->u_sigmask[sig-1]);
1369                 if (sigismember(&up->u_sigonstack, sig))
1370                         sp->sa_flags |= SA_ONSTACK;
1371                 if (sigismember(&up->u_sigresethand, sig))
1372                         sp->sa_flags |= SA_RESETHAND;
1373                 if (sigismember(&up->u_sigrestart, sig))
1374                         sp->sa_flags |= SA_RESTART;
1375                 if (sigismember(&p->p_siginfo, sig))
1376                         sp->sa_flags |= SA_SIGINFO;
1377                 if (sigismember(&up->u_signodefer, sig))
1378                         sp->sa_flags |= SA_NODEFER;
1379                 if (sig == SIGCLD) {
```



```
1380                         if (p->p_flag & SNOWAIT)
1381                                 sp->sa_flags |= SA_NOCLDWAIT;
1382                         if ((p->p_flag & SJCTL) == 0)
1383                                 sp->sa_flags |= SA_NOCLDSTOP;
1384                 }
1385         }
1386 }

1388 /* ISA dependent function. */
1389 static int
1390 gcore_prfetchinstr(mdb_klwp_t *lwp, ulong_t *ip)
1391 {
1392         *ip = (ulong_t)(instr_t)lwp->lwp_pcb.pcb_instr;
1393         return (lwp->lwp_pcb.pcb_flags & INSTR_VALID);
1394 }

1396 /* ISA dependent function. */
1397 static int
1398 gcore_prisstep(mdb_klwp_t *lwp)
1399 {
1400         return ((lwp->lwp_pcb.pcb_flags &
1401             (NORMAL_STEP|WATCH_STEP|DEBUG_PENDING)) != 0);
1402 }

1404 /* ISA dependent function. */
1405 static void
1406 gcore_getgregs(mdb_klwp_t *lwp, gregset_t grp)
1407 {
1408         struct regs rgs;
1409         struct regs *rp;

1411         if (mdb_vread(&rgs, sizeof (rgs), lwp->lwp_regs) != sizeof (rgs)) {
1412                 mdb_warn("Failed to read regs from %p\n", lwp->lwp_regs);
1413                 return;
1414         }
1415         rp = &rgs;

1417 #if defined(__amd64)
1418         struct pcb *pcb = &lwp->lwp_pcb;

1420         grp[REG_RDI] = rp->r_rdi;
1421         grp[REG_RSI] = rp->r_rsi;
1422         grp[REG_RDX] = rp->r_rdx;
1423         grp[REG_RCX] = rp->r_rcx;
1424         grp[REG_R8] = rp->r_r8;
1425         grp[REG_R9] = rp->r_r9;
1426         grp[REG_RAX] = rp->r_rax;
1427         grp[REG_RBX] = rp->r_rbx;
1428         grp[REG_RBP] = rp->r_rbp;
1429         grp[REG_R10] = rp->r_r10;
1430         grp[REG_R11] = rp->r_r11;
1431         grp[REG_R12] = rp->r_r12;
1432         grp[REG_R13] = rp->r_r13;
1433         grp[REG_R14] = rp->r_r14;
1434         grp[REG_R15] = rp->r_r15;
1435         grp[REG_FSBASE] = pcb->pcb_fsbase;
1436         grp[REG_GSBASE] = pcb->pcb_gsbase;
1437         if (pcb->pcb_rupdate == 1) {
1438                 grp[REG_DS] = pcb->pcb_ds;
1439                 grp[REG_ES] = pcb->pcb_es;
1440                 grp[REG_FS] = pcb->pcb_fs;
1441                 grp[REG_GS] = pcb->pcb_gs;
1442         } else {
1443                 grp[REG_DS] = rp->r_ds;
1444                 grp[REG_ES] = rp->r_es;
1445                 grp[REG_FS] = rp->r_fs;
```

```
1446                    grp[REG_GS] = rp->r_gs;
1447            }
1448            grp[REG_TRAPNO] = rp->r_trapno;
1449            grp[REG_ERR] = rp->r_err;
1450            grp[REG_RIP] = rp->r_rip;
1451            grp[REG_CS] = rp->r_cs;
1452            grp[REG_SS] = rp->r_ss;
1453            grp[REG_RFL] = rp->r_rfl;
1454            grp[REG_RSP] = rp->r_rsp;
1455    #else
1456            bcopy(&rp->r_gs, grp, sizeof (gregset_t));
1457    #endif
1458    }

1460    /* ISA dependent functions. */
1461    static int
1462    gcore_prgetrvals(mdb_klwp_t *lwp, long *rval1, long *rval2)
1463    {
1464            struct regs *r = lwptoregs(lwp);

1466            if (r->r_ps & PS_C)
1467                    return (r->r_r0);
1468            if (lwp->lwp_eosys == JUSTRETURN) {
1469                    *rval1 = 0;
1470                    *rval2 = 0;
1471            } else {
1472                    *rval1 = r->r_r0;
1473                    *rval2 = r->r_r1;
1474            }
1475            return (0);
1476    }

1478    static void
1479    gcore_prgetprregs(mdb_klwp_t *lwp, prgregset_t prp)
1480    {
1481            gcore_getgregs(lwp, prp);
1482    }

1484    /*
1485     * Field not populated:
1486     *   - pr_tstamp
1487     *   - pr_utime
1488     *   - pr_stime
1489     *   - pr_syscall
1490     *   - pr_syarg
1491     *   - pr_nsysarg
1492     *   - pr_fpreg
1493     */
1494    /*ARGSUSED*/
1495    static int
1496    gcore_prgetlwpstatus(mdb_proc_t *p, uintptr_t t_addr, mdb_kthread_t *t,
1497        lwpstatus_t *sp, zone_t *zp)
1498    {
1499            uintptr_t       lwp_addr = ttolwp(t);
1500            mdb_klwp_t      lw;
1501            mdb_klwp_t      *lwp;
1502            ulong_t         instr;
1503            int             flags;
1504            uintptr_t       str_addr;
1505            struct pid      pid;

1507            if (mdb_ctf_vread(&lw, "klwp_t", "mdb_klwp_t", lwp_addr, 0) == -1) {
1508                    return (-1);
1509            }
1510            lwp = &lw;
```



```
1512            bzero(sp, sizeof (*sp));
1513            flags = 0L;
1514            if (t->t_state == TS_STOPPED) {
1515                    flags |= PR_STOPPED;
1516                    if ((t->t_schedflag & TS_PSTART) == 0)
1517                            flags |= PR_ISTOP;
1518            } else if (VSTOPPED(t)) {
1519                    flags |= PR_STOPPED|PR_ISTOP;
1520            }
1521            if (!(flags & PR_ISTOP) && (t->t_proc_flag & TP_PRSTOP))
1522                    flags |= PR_DSTOP;
1523            if (lwp->lwp_asleep)
1524                    flags |= PR_ASLEEP;
1525            if (t_addr == p->p_agenttp)
1526                    flags |= PR_AGENT;
1527            if (!(t->t_proc_flag & TP_TWAIT))
1528                    flags |= PR_DETACH;
1529            if (t->t_proc_flag & TP_DAEMON)
1530                    flags |= PR_DAEMON;
1531            if (p->p_proc_flag & P_PR_FORK)
1532                    flags |= PR_FORK;
1533            if (p->p_proc_flag & P_PR_RUNLCL)
1534                    flags |= PR_RLC;
1535            if (p->p_proc_flag & P_PR_KILLCL)
1536                    flags |= PR_KLC;
1537            if (p->p_proc_flag & P_PR_ASYNC)
1538                    flags |= PR_ASYNC;
1539            if (p->p_proc_flag & P_PR_BPTADJ)
1540                    flags |= PR_BPTADJ;
1541            if (p->p_proc_flag & P_PR_PTRACE)
1542                    flags |= PR_PTRACE;
1543            if (p->p_flag & SMSACCT)
1544                    flags |= PR_MSACCT;
1545            if (p->p_flag & SMSFORK)
1546                    flags |= PR_MSFORK;
1547            if (p->p_flag & SVFWAIT)
1548                    flags |= PR_VFORKP;

1550            if (mdb_vread(&pid, sizeof (struct pid), p->p_pgidp) != sizeof (pid)) {
1551                    mdb_warn("Failed to read pid from %p\n", p->p_pgidp);
1552                    return (-1);
1553            }
1554            if (pid.pid_pgorphaned)
1555                    flags |= PR_ORPHAN;
1556            if (p->p_pidflag & CLDNOSIGCHLD)
1557                    flags |= PR_NOSIGCHLD;
1558            if (p->p_pidflag & CLDWAITPID)
1559                    flags |= PR_WAITPID;
1560            sp->pr_flags = flags;
1561            if (VSTOPPED(t)) {
1562                    sp->pr_why   = PR_REQUESTED;
1563                    sp->pr_what  = 0;
1564            } else {
1565                    sp->pr_why   = t->t_whystop;
1566                    sp->pr_what  = t->t_whatstop;
1567            }
1568            sp->pr_lwpid = t->t_tid;
1569            sp->pr_cursig  = lwp->lwp_cursig;
1570            prassignset(&sp->pr_lwppend, &t->t_sig);
1571            gcore_schedctl_finish_sigblock(t);
1572            prassignset(&sp->pr_lwphold, &t->t_hold);
1573            if (t->t_whystop == PR_FAULTED) {
1574                    bcopy(&lwp->lwp_siginfo,
1575                        &sp->pr_info, sizeof (k_siginfo_t));
1576            } else if (lwp->lwp_curinfo) {
1577                    mdb_sigqueue_t  sigq;
```

```
1579                if (mdb_ctf_vread(&sigq, "sigqueue_t", "mdb_sigqueue_t",
1580                    lwp->lwp_curinfo, 0) == -1) {
1581                        return (-1);
1582                }
1583                bcopy(&sigq.sq_info, &sp->pr_info, sizeof (k_siginfo_t));
1584        }

1586        sp->pr_altstack = lwp->lwp_sigaltstack;
1587        gcore_prgetaction(p, PTOU(p), lwp->lwp_cursig, &sp->pr_action);
1588        sp->pr_oldcontext = lwp->lwp_oldcontext;
1589        sp->pr_ustack = lwp->lwp_ustack;

1591        str_addr = (uintptr_t)gcore_sclass[t->t_cid].cl_name;
1592        if (mdb_readstr(sp->pr_clname, sizeof (sp->pr_clname) - 1, str_addr) ==
1593            -1) {
1594                mdb_warn("Failed to read string from %p\n", str_addr);
1595                return (-1);
1596        }

1598        /*
1599         * Fetch the current instruction, if not a system process.
1600         * We don't attempt this unless the lwp is stopped.
1601         */
1602        if ((p->p_flag & SSYS) || p->p_as == gcore_kas)
1603                sp->pr_flags |= (PR_ISSYS|PR_PCINVAL);
1604        else if (!(flags & PR_STOPPED))
1605                sp->pr_flags |= PR_PCINVAL;
1606        else if (!gcore_prfetchinstr(lwp, &instr))
1607                sp->pr_flags |= PR_PCINVAL;
1608        else
1609                sp->pr_instr = instr;

1611        if (gcore_prisstep(lwp))
1612                sp->pr_flags |= PR_STEP;
1613        gcore_prgetprregs(lwp, sp->pr_reg);
1614        if ((t->t_state == TS_STOPPED && t->t_whystop == PR_SYSEXIT) ||
1615            (flags & PR_VFORKP)) {
1616                user_t *up;
1617                auxv_t *auxp;
1618                int i;

1620                sp->pr_errno = gcore_prgetrvals(lwp, &sp->pr_rval1,
1621                    &sp->pr_rval2);
1622                if (sp->pr_errno == 0)
1623                        sp->pr_errpriv = PRIV_NONE;
1624                else
1625                        sp->pr_errpriv = lwp->lwp_badpriv;

1627                if (t->t_sysnum == SYS_execve) {
1628                        up = PTOU(p);
1629                        sp->pr_sysarg[0] = 0;
1630                        sp->pr_sysarg[1] = (uintptr_t)up->u_argv;
1631                        sp->pr_sysarg[2] = (uintptr_t)up->u_envp;
1632                        for (i = 0, auxp = up->u_auxv;
1633                            i < sizeof (up->u_auxv) / sizeof (up->u_auxv[0]);
1634                            i++, auxp++) {
1635                                if (auxp->a_type == AT_SUN_EXECNAME) {
1636                                        sp->pr_sysarg[0] =
1637                                            (uintptr_t)auxp->a_un.a_ptr;
1638                                        break;
1639                                }
1640                        }
1641                }
1642        }
1643        return (0);
```



```
1644 }

1646 static int
1647 gcore_lstatus_cb(mdb_proc_t *p, lwpent_t *lwent, void *data)
1648 {
1649         lwpstatus_t     *lstatus = data;
1650         uintptr_t       t_addr = (uintptr_t)lwent->le_thread;
1651         mdb_kthread_t   kthrd;

1653         if (t_addr == NULL) {
1654                 return (1);
1655         }

1657         if (mdb_ctf_vread(&kthrd, "kthread_t", "mdb_kthread_t", t_addr, 0)
1658             == -1) {
1659                 return (-1);
1660         }

1662         return (gcore_prgetlwpstatus(p, t_addr, &kthrd, lstatus, NULL));
1663 }

1665 static prheader_t *
1666 gcore_walk_lwps(mdb_proc_t *p, lwp_callback_t callback, int nlwp,
1667     size_t ent_size)
1668 {
1669         void            *ent;
1670         prheader_t      *php;
1671         lwpdir_t        *ldp;
1672         lwpdir_t        ld;
1673         lwpent_t        lwent;
1674         int             status;
1675         int             i;

1677         php = calloc(1, sizeof (prheader_t) + nlwp * ent_size);
1678         if (php == NULL) {
1679                 return (NULL);
1680         }
1681         php->pr_nent = nlwp;
1682         php->pr_entsize = ent_size;

1684         ent = php + 1;
1685         for (ldp = (lwpdir_t *)p->p_lwpdir, i = 0; i < p->p_lwpdir_sz; i++,
1686             ldp++) {
1687                 if (mdb_vread(&ld, sizeof (ld), (uintptr_t)ldp) !=
1688                     sizeof (ld)) {
1689                         mdb_warn("Failed to read lwpdir_t from %p\n", ldp);
1690                         goto error;
1691                 }

1693                 if (ld.ld_entry == NULL) {
1694                         continue;
1695                 }

1697                 if (mdb_vread(&lwent, sizeof (lwent), (uintptr_t)ld.ld_entry) !=
1698                     sizeof (lwent)) {
1699                         mdb_warn("Failed to read lwpent_t from %p\n",
1700                             ld.ld_entry);
1701                         goto error;
1702                 }

1704                 status = callback(p, &lwent, ent);
1705                 if (status == -1) {
1706                         dprintf("lwp callback %p returned -1\n", callback);
1707                         goto error;
1708                 }
1709                 if (status == 1) {
```

```
1710                        dprintf("lwp callback %p returned 1\n", callback);
1711                        continue;
1712                }

1714                ent = (caddr_t)ent + ent_size;
1715        }

1717        return (php);

1719 error:
1720        free(php);
1721        return (NULL);
1722 }

1724 /*
1725  * Misc helper functions.
1726  */
1727 /*
1728  * convert code/data pair into old style wait status
1729  */
1730 static int
1731 gcore_wstat(int code, int data)
1732 {
1733        int stat = (data & 0377);

1735        switch (code) {
1736        case CLD_EXITED:
1737                stat <<= 8;
1738                break;
1739        case CLD_DUMPED:
1740                stat |= WCOREFLG;
1741                break;
1742        case CLD_KILLED:
1743                break;
1744        case CLD_TRAPPED:
1745        case CLD_STOPPED:
1746                stat <<= 8;
1747                stat |= WSTOPFLG;
1748                break;
1749        case CLD_CONTINUED:
1750                stat = WCONTFLG;
1751                break;
1752        default:
1753                mdb_warn("wstat: bad code %d\n", code);
1754        }
1755        return (stat);
1756 }

1758 #if defined(__i386) || defined(__amd64)
1759 static void
1760 gcore_usd_to_ssd(user_desc_t *usd, struct ssd *ssd, selector_t sel)
1761 {
1762        ssd->bo = USEGD_GETBASE(usd);
1763        ssd->ls = USEGD_GETLIMIT(usd);
1764        ssd->sel = sel;

1766        /*
1767         * set type, dpl and present bits.
1768         */
1769        ssd->acc1 = usd->usd_type;
1770        ssd->acc1 |= usd->usd_dpl << 5;
1771        ssd->acc1 |= usd->usd_p << (5 + 2);

1773        /*
1774         * set avl, DB and granularity bits.
1775         */
```



```
1776        ssd->acc2 = usd->usd_avl;

1778 #if defined(__amd64)
1779        ssd->acc2 |= usd->usd_long << 1;
1780 #else
1781        ssd->acc2 |= usd->usd_reserved << 1;
1782 #endif

1784        ssd->acc2 |= usd->usd_def32 << (1 + 1);
1785        ssd->acc2 |= usd->usd_gran << (1 + 1 + 1);
1786 }
1787 #endif

1789 static priv_set_t *
1790 gcore_priv_getset(cred_t *cr, int set)
1791 {
1792        if ((CR_FLAGS(cr) & PRIV_AWARE) == 0) {
1793                switch (set) {
1794                case PRIV_EFFECTIVE:
1795                        return (&CR_OEPRIV(cr));
1796                case PRIV_PERMITTED:
1797                        return (&CR_OPPRIV(cr));
1798                }
1799        }
1800        return (&CR_PRIVS(cr)->crprivs[set]);
1801 }

1803 static void
1804 gcore_priv_getinfo(const cred_t *cr, void *buf)
1805 {
1806        struct priv_info_uint *ii;

1808        ii = buf;
1809        ii->val = CR_FLAGS(cr);
1810        ii->info.priv_info_size = (uint32_t)sizeof (*ii);
1811        ii->info.priv_info_type = PRIV_INFO_FLAGS;
1812 }

1814 static void
1815 map_list_free(prmap_node_t *n)
1816 {
1817        prmap_node_t    *next;

1819        while (n != NULL) {
1820                next = n->next;
1821                mdb_free(n, sizeof (*n));
1822                n = next;
1823        }
1824 }

1826 /*
1827  * Ops vector functions for ::gcore.
1828  */
1829 /*ARGSUSED*/
1830 static ssize_t
1831 Pread_gcore(struct ps_prochandle *P, void *buf, size_t n, uintptr_t addr,
1832     void *data)
1833 {
1834        mdb_proc_t      *p = data;
1835        ssize_t         ret;

1837        ret = mdb_aread(buf, n, addr, (void *)p->p_as);
1838        if (ret != n) {
1839                dprintf("%s: addr: %p len: %llx\n", __func__, addr, n);
1840                (void) memset(buf, 0, n);
1841                return (n);
```

```
1842         }

1844         return (ret);
1845 }

1847 /*ARGSUSED*/
1848 static ssize_t
1849 Pwrite_gcore(struct ps_prochandle *P, const void *buf, size_t n, uintptr_t addr,
1850     void *data)
1851 {
1852         dprintf("%s: addr: %p len: %llx\n", __func__, addr, n);

1854         return (-1);
1855 }

1857 /*ARGSUSED*/
1858 static int
1859 Pread_maps_gcore(struct ps_prochandle *P, prmap_t **Pmapp, ssize_t *nmapp,
1860     void *data)
1861 {
1862         mdb_proc_t      *p = data;
1863         read_maps_cbarg_t cbarg;
1864         prmap_node_t    *n;
1865         prmap_t         *pmap;
1866         uintptr_t       segtree_addr;
1867         int             error;
1868         int             i;

1870         cbarg.p = p;
1871         cbarg.brkseg = gcore_break_seg(p);
1872         cbarg.stkseg = gcore_as_segat(p->p_as, gcore_prgetstackbase(p));

1874         (void) memset(&cbarg, 0, sizeof (cbarg));
1875         segtree_addr = p->p_as + mdb_ctf_offsetof_by_name("struct as",
1876             "a_segtree");
1877         error = avl_walk_mdb(segtree_addr, read_maps_cb, &cbarg);
1878         if (error != WALK_DONE) {
1879                 return (-1);
1880         }

1882         /* Conver the linked list into an array */
1883         pmap = malloc(cbarg.map_len * sizeof (*pmap));
1884         if (pmap == NULL) {
1885                 map_list_free(cbarg.map_head);
1886                 return (-1);
1887         }

1889         for (i = 0, n = cbarg.map_head; i < cbarg.map_len; i++, n = n->next) {
1890                 (void) memcpy(&pmap[i], &n->m, sizeof (prmap_t));
1891         }
1892         map_list_free(cbarg.map_head);

1894         for (i = 0; i < cbarg.map_len; i++) {
1895                 dprintf("pr_vaddr: %p pr_size: %llx, pr_name: %s "
1896                     "pr_offset: %p pr_mflags: 0x%x\n",
1897                     pmap[i].pr_vaddr, pmap[i].pr_size,
1898                     pmap[i].pr_mapname, pmap[i].pr_offset,
1899                     pmap[i].pr_mflags);
1900         }

1902         *Pmapp = pmap;
1903         *nmapp = cbarg.map_len;

1905         return (0);
1906 }
```



```
1908 /*ARGSUSED*/
1909 static void
1910 Pread_aux_gcore(struct ps_prochandle *P, auxv_t **auxvp, int *nauxp, void *data)
1911 {
1912         mdb_proc_t      *p = data;
1913         auxv_t          *auxv;
1914         int             naux;

1916         naux = __KERN_NAUXV_IMPL;
1917         auxv = calloc(naux + 1, sizeof (*auxv));
1918         if (auxv == NULL) {
1919                 *auxvp = NULL;
1920                 *nauxp = 0;
1921                 return;
1922         }

1924         (void) memcpy(auxv, p->p_user.u_auxv, naux * sizeof (*auxv));

1926         *auxvp = auxv;
1927         *nauxp = naux;
1928 }

1930 /*ARGSUSED*/
1931 static int
1932 Pcred_gcore(struct ps_prochandle *P, prcred_t *prcp, int ngroups, void *data)
1933 {
1934         mdb_proc_t      *p = data;
1935         cred_t          cr;
1936         credgrp_t       crgrp;
1937         int             i;

1939         if (mdb_vread(&cr, sizeof (cr), p->p_cred) != sizeof (cr)) {
1940                 mdb_warn("Failed to read cred_t from %p\n", p->p_cred);
1941                 return (-1);
1942         }

1944         prcp->pr_euid = cr.cr_uid;
1945         prcp->pr_ruid = cr.cr_ruid;
1946         prcp->pr_suid = cr.cr_suid;
1947         prcp->pr_egid = cr.cr_gid;
1948         prcp->pr_rgid = cr.cr_rgid;
1949         prcp->pr_sgid = cr.cr_sgid;

1951         if (cr.cr_grps == 0) {
1952                 prcp->pr_ngroups = 0;
1953                 return (0);
1954         }

1956         if (mdb_vread(&crgrp, sizeof (crgrp), (uintptr_t)cr.cr_grps) !=
1957             sizeof (crgrp)) {
1958                 mdb_warn("Failed to read credgrp_t from %p\n", cr.cr_grps);
1959                 return (-1);
1960         }

1962         prcp->pr_ngroups = MIN(ngroups, crgrp.crg_ngroups);
1963         for (i = 0; i < prcp->pr_ngroups; i++) {
1964                 prcp->pr_groups[i] = crgrp.crg_groups[i];
1965         }

1967         return (0);
1968 }

1970 /*ARGSUSED*/
1971 static int
1972 Ppriv_gcore(struct ps_prochandle *P, prpriv_t **pprv, void *data)
1973 {
```

```
1974         mdb_proc_t      *p = data;
1975         prpriv_t        *pp;
1976         cred_t          cr;
1977         priv_set_t      *psa;
1978         size_t          pprv_size;
1979         int             i;

1981         pprv_size = sizeof (prpriv_t) + PRIV_SETBYTES - sizeof (priv_chunk_t) +
1982             prinfo.priv_infosize;

1984         pp = malloc(pprv_size);
1985         if (pp == NULL) {
1986                 return (-1);
1987         }

1989         if (mdb_vread(&cr, sizeof (cr), p->p_cred) != sizeof (cr)) {
1990                 mdb_warn("Failed to read cred_t from %p\n", p->p_cred);
1991                 free(pp);
1992                 return (-1);
1993         }

1995         pp->pr_nsets = PRIV_NSET;
1996         pp->pr_setsize = PRIV_SETSIZE;
1997         pp->pr_infosize = prinfo.priv_infosize;

1999         psa = (priv_set_t *)pp->pr_sets;
2000         for (i = 0; i < PRIV_NSET; i++) {
2001                 psa[i] = *gcore_priv_getset(&cr, i);
2002         }

2004         gcore_priv_getinfo(&cr, (char *)pp + PRIV_PRPRIV_INFO_OFFSET(pp));

2006         *pprv = pp;
2007         return (0);
2008 }

2010 /*
2011  * Fields not filled populated:
2012  *   - pr_utime
2013  *   - pr_stkbase
2014  *   - pr_cutime
2015  *   - pr_cstime
2016  *   - pr_agentid
2017  */
2018 /*ARGSUSED*/
2019 static void
2020 Pstatus_gcore(struct ps_prochandle *P, pstatus_t *sp, void *data)
2021 {
2022         mdb_proc_t      *p = data;
2023         uintptr_t       t_addr;
2024         mdb_kthread_t   kthr;
2025         mdb_kthread_t   *t;
2026         pcommon_t       pc;

2028         t_addr = gcore_prchoose(p);
2029         if (t_addr != NULL) {
2030                 if (mdb_ctf_vread(&kthr, "kthread_t", "mdb_kthread_t", t_addr,
2031                     0) == -1) {
2032                         return;
2033                 }
2034                 t = &kthr;
2035         }

2037         /* just bzero the process part, prgetlwpstatus() does the rest */
2038         bzero(sp, sizeof (pstatus_t) - sizeof (lwpstatus_t));
```



```
2040         if (pcommon_init(p, &pc) == -1) {
2041                 return;
2042         }
2043         sp->pr_nlwp = pc.pc_nlwp;
2044         sp->pr_nzomb = pc.pc_nzomb;
2045         sp->pr_pid = pc.pc_pid;
2046         sp->pr_ppid = pc.pc_ppid;
2047         sp->pr_pgid = pc.pc_pgid;
2048         sp->pr_sid = pc.pc_sid;
2049         sp->pr_taskid = pc.pc_taskid;
2050         sp->pr_projid = pc.pc_projid;
2051         sp->pr_zoneid = pc.pc_zoneid;
2052         sp->pr_dmodel = pc.pc_dmodel;

2054         prassignset(&sp->pr_sigpend, &p->p_sig);
2055         sp->pr_brkbase = p->p_brkbase;
2056         sp->pr_brksize = p->p_brksize;
2057         sp->pr_stkbase = gcore_prgetstackbase(p);
2058         sp->pr_stksize = p->p_stksize;

2060         prassignset(&sp->pr_sigtrace, &p->p_sigmask);
2061         prassignset(&sp->pr_flttrace, &p->p_fltmask);
2062         prassignset(&sp->pr_sysentry, &PTOU(p)->u_entrymask);
2063         prassignset(&sp->pr_sysexit, &PTOU(p)->u_exitmask);

2065         /* get the chosen lwp's status */
2066         gcore_prgetlwpstatus(p, t_addr, t, &sp->pr_lwp, NULL);

2068         /* replicate the flags */
2069         sp->pr_flags = sp->pr_lwp.pr_flags;
2070 }

2072 /*
2073  * Fields not populated:
2074  *   - pr_contract
2075  *   - pr_addr
2076  *   - pr_rtime
2077  *   - pr_ctime
2078  *   - pr_ttydev
2079  *   - pr_pctcpu
2080  *   - pr_size
2081  *   - pr_rsize
2082  *   - pr_pctmem
2083  */
2084 /*ARGSUSED*/
2085 static const psinfo_t *
2086 Ppsinfo_gcore(struct ps_prochandle *P, psinfo_t *psp, void *data)
2087 {
2088         mdb_proc_t      *p = data;
2089         mdb_kthread_t   *t;
2090         mdb_pool_t      pool;
2091         cred_t          cr;
2092         uintptr_t       t_addr;
2093         pcommon_t       pc;

2095         if ((t_addr = gcore_prchoose(p)) == NULL) {
2096                 bzero(psp, sizeof (*psp));
2097         } else {
2098                 bzero(psp, sizeof (*psp) - sizeof (psp->pr_lwp));
2099         }

2101         if (pcommon_init(p, &pc) == -1) {
2102                 return (NULL);
2103         }
2104         psp->pr_nlwp = pc.pc_nlwp;
2105         psp->pr_nzomb = pc.pc_nzomb;
```

```
2106         psp->pr_pid = pc.pc_pid;
2107         psp->pr_ppid = pc.pc_ppid;
2108         psp->pr_pgid = pc.pc_pgid;
2109         psp->pr_sid = pc.pc_sid;
2110         psp->pr_taskid = pc.pc_taskid;
2111         psp->pr_projid = pc.pc_projid;
2112         psp->pr_dmodel = pc.pc_dmodel;

2114         /*
2115          * only export SSYS and SMSACCT; everything else is off-limits to
2116          * userland apps.
2117          */
2118         psp->pr_flag = p->p_flag & (SSYS | SMSACCT);

2120         if (mdb_vread(&cr, sizeof (cr), p->p_cred) != sizeof (cr)) {
2121                 mdb_warn("Failed to read cred_t from %p\n", p->p_cred);
2122                 return (NULL);
2123         }

2125         psp->pr_uid = cr.cr_ruid;
2126         psp->pr_euid = cr.cr_uid;
2127         psp->pr_gid = cr.cr_rgid;
2128         psp->pr_egid = cr.cr_gid;

2130         if (mdb_ctf_vread(&pool, "pool_t", "mdb_pool_t", p->p_pool, 0) == -1) {
2131                 return (NULL);
2132         }
2133         psp->pr_poolid = pool.pool_id;

2135         if (t_addr == 0) {
2136                 int wcode = p->p_wcode;

2138                 if (wcode)
2139                         psp->pr_wstat = gcore_wstat(wcode, p->p_wdata);
2140                 psp->pr_ttydev = PRNODEV;
2141                 psp->pr_lwp.pr_state = SZOMB;
2142                 psp->pr_lwp.pr_sname = 'Z';
2143                 psp->pr_lwp.pr_bindpro = PBIND_NONE;
2144                 psp->pr_lwp.pr_bindpset = PS_NONE;
2145         } else {
2146                 mdb_kthread_t   kthr;
2147                 user_t          *up = PTOU(p);

2149                 psp->pr_start = up->u_start;
2150                 bcopy(up->u_comm, psp->pr_fname,
2151                     MIN(sizeof (up->u_comm), sizeof (psp->pr_fname)-1));
2152                 bcopy(up->u_psargs, psp->pr_psargs,
2153                     MIN(PRARGSZ-1, PSARGSZ));

2155                 psp->pr_argc = up->u_argc;
2156                 psp->pr_argv = up->u_argv;
2157                 psp->pr_envp = up->u_envp;

2159                 /* get the chosen lwp's lwpsinfo */
2160                 if (mdb_ctf_vread(&kthr, "kthread_t", "mdb_kthread_t", t_addr,
2161                     0) == -1) {
2162                         return (NULL);
2163                 }
2164                 t = &kthr;

2166                 gcore_prgetlwpsinfo(t_addr, t, &psp->pr_lwp);
2167         }

2169         return (NULL);
2170 }
```



```
2172 /*ARGSUSED*/
2173 static prheader_t *
2174 Plstatus_gcore(struct ps_prochandle *P, void *data)
2175 {
2176         mdb_proc_t      *p = data;
2177         int             nlwp = p->p_lwpcnt;
2178         size_t          ent_size = LSPAN(lwpstatus_t);

2180         return (gcore_walk_lwps(p, gcore_lstatus_cb, nlwp, ent_size));
2181 }

2183 /*ARGSUSED*/
2184 static prheader_t *
2185 Plpsinfo_gcore(struct ps_prochandle *P, void *data)
2186 {
2187         mdb_proc_t      *p = data;
2188         int             nlwp = p->p_lwpcnt + p->p_zombcnt;
2189         size_t          ent_size = LSPAN(lwpsinfo_t);

2191         return (gcore_walk_lwps(p, gcore_lpsinfo_cb, nlwp, ent_size));
2192 }

2194 /*ARGSUSED*/
2195 static char *
2196 Pplatform_gcore(struct ps_prochandle *P, char *s, size_t n, void *data)
2197 {
2198         char    platform[SYS_NMLN];

2200         if (mdb_readvar(platform, "platform") == -1) {
2201                 mdb_warn("failed to read platform!\n");
2202                 return (NULL);
2203         }
2204         dprintf("platform: %s\n", platform);

2206         (void) strncpy(s, platform, n);
2207         return (s);
2208 }

2210 /*ARGSUSED*/
2211 static int
2212 Puname_gcore(struct ps_prochandle *P, struct utsname *u, void *data)
2213 {
2214         if (mdb_readvar(u, "utsname") != sizeof (*u)) {
2215                 return (-1);
2216         }

2218         return (0);
2219 }

2221 /*ARGSUSED*/
2222 static char *
2223 Pzonename_gcore(struct ps_prochandle *P, char *s, size_t n, void *data)
2224 {
2225         mdb_proc_t      *p = data;
2226         mdb_zone_t      zone;

2228         if (mdb_ctf_vread(&zone, "zone_t", "mdb_zone_t", p->p_zone, 0) == -1) {
2229                 return (NULL);
2230         }

2232         if (mdb_readstr(s, n, zone.zone_name) == -1) {
2233                 mdb_warn("Failed to read zone name from %p\n", zone.zone_name);
2234                 return (NULL);
2235         }

2237         return (s);
```

```
}

/*ARGSUSED*/
static char *
Pexecname_gcore(struct ps_prochandle *P, char *buf, size_t buflen, void *data)
{
        mdb_proc_t      *p = data;
        mdb_vnode_t     vn;

        if (mdb_ctf_vread(&vn, "vnode_t", "mdb_vnode_t", p->p_exec, 0) == -1) {
                return (NULL);
        }

        if (mdb_readstr(buf, buflen, vn.v_path) == -1) {
                mdb_warn("Failed to read vnode path from %p\n", vn.v_path);
                return (NULL);
        }

        dprintf("execname: %s\n", buf);

        return (buf);
}

#if defined(__i386) || defined(__amd64)
/*ARGSUSED*/
static int
Pldt_gcore(struct ps_prochandle *P, struct ssd *pldt, int nldt, void *data)
{
        mdb_proc_t      *p = data;
        user_desc_t     *udp;
        user_desc_t     *ldts;
        size_t          ldt_size;
        int             i, limit;

        if (p->p_ldt == NULL) {
                return (0);
        }

        limit = p->p_ldtlimit;

        /* Is this call just to query the size ? */
        if (pldt == NULL || nldt == 0) {
                return (limit);
        }

        ldt_size = limit * sizeof (*ldts);
        ldts = malloc(ldt_size);
        if (ldts == NULL) {
                mdb_warn("Failed to malloc ldts (size %lld)n", ldt_size);
                return (-1);
        }

        if (mdb_vread(ldts, ldt_size, p->p_ldt) != ldt_size) {
                mdb_warn("Failed to read ldts from %p\n", p->p_ldt);
                free(ldts);
                return (-1);
        }

        for (i = LDT_UDBASE, udp = &ldts[i]; i <= limit; i++, udp++) {
                if (udp->usd_type != 0 || udp->usd_dpl != 0 ||
                    udp->usd_p != 0) {
                        gcore_usd_to_ssd(udp, pldt++, SEL_LDT(i));
                }
        }

        free(ldts);
```



```
        return (limit);
}
#endif

static const ps_ops_t Pgcore_ops = {
        .pop_pread      = Pread_gcore,
        .pop_pwrite     = Pwrite_gcore,
        .pop_read_maps  = Pread_maps_gcore,
        .pop_read_aux   = Pread_aux_gcore,
        .pop_cred       = Pcred_gcore,
        .pop_priv       = Ppriv_gcore,
        .pop_psinfo     = Ppsinfo_gcore,
        .pop_status     = Pstatus_gcore,
        .pop_lstatus    = Plstatus_gcore,
        .pop_lpsinfo    = Plpsinfo_gcore,
        .pop_platform   = Pplatform_gcore,
        .pop_uname      = Puname_gcore,
        .pop_zonename   = Pzonename_gcore,
        .pop_execname   = Pexecname_gcore,
#if defined(__i386) || defined(__amd64)
        .pop_ldt        = Pldt_gcore
#endif
};

/*ARGSUSED*/
int
gcore_dcmd(uintptr_t addr, uint_t flags, int argc, const mdb_arg_t *argv)
{
        struct ps_prochandle *P;
        char            core_name[MAXNAMELEN];
        mdb_proc_t      p;
        mdb_pid_t       pid;
        int             error;

        if (!gcore_initialized) {
                mdb_warn("gcore unavailable\n");
                return (DCMD_ERR);
        }

        if (mdb_ctf_vread(&p, "proc_t", "mdb_proc_t", addr, 0) == -1) {
                return (DCMD_ERR);
        }

        if (p.p_flag & SSYS) {
                mdb_warn("'%s' is a system process\n", p.p_user.u_comm);
                return (DCMD_ERR);
        }

        if (mdb_ctf_vread(&pid, "struct pid", "mdb_pid_t", p.p_pidp, 0)
            == -1) {
                return (DCMD_ERR);
        }

        if ((P = Pgrab_ops(pid.pid_id, &p, &Pgcore_ops, PGRAB_INCORE)) ==
            NULL) {
                mdb_warn("Failed to initialize proc handle");
                return (DCMD_ERR);
        }

        (void) snprintf(core_name, sizeof (core_name), "core.%s.%d",
            p.p_user.u_comm, pid.pid_id);

        if ((error = Pgcore(P, core_name, CC_CONTENT_DEFAULT)) != 0) {
                mdb_warn("Failed to generate core file: %d", error);
                Pfree(P);
                return (DCMD_ERR);
```

```
2370         }

2372         Pfree(P);
2373         mdb_printf("Created core file: %s\n", core_name);

2375         return (0);
2376 }

2378 void
2379 gcore_init(void)
2380 {
2381         GElf_Sym        sym;
2382         uintptr_t       priv_info_addr;

2384         if (mdb_lookup_by_name("segvn_ops", &sym) == -1) {
2385                 mdb_warn("Failed to lookup symbol 'segvn_ops'\n");
2386                 return;
2387         }
2388         gcore_segvn_ops = sym.st_value;

2390         if (mdb_readvar(&priv_info_addr, "priv_info") == -1) {
2391                 mdb_warn("Failed to read variable 'priv_info'\n");
2392                 return;
2393         }

2395         if (mdb_vread(&prinfo, sizeof (prinfo), priv_info_addr) == -1) {
2396                 mdb_warn("Failed to read prinfo from %p\n", priv_info_addr);
2397                 return;
2398         }

2400         if (mdb_lookup_by_name("sclass", &sym) == -1) {
2401                 mdb_warn("Failed to lookup symbol 'segvn_ops'\n");
2402                 return;
2403         }

2405         gcore_sclass = mdb_zalloc(sym.st_size, UM_SLEEP);
2406         if (mdb_vread(gcore_sclass, sym.st_size, sym.st_value) != sym.st_size) {
2407                 mdb_warn("Failed to read sclass' from %p\n", sym.st_value);
2408                 return;
2409         }

2411         if (mdb_lookup_by_name("kas", &sym) == -1) {
2412                 mdb_warn("Failed to lookup symbol 'kas'\n");
2413                 return;
2414         }
2415         gcore_kas = sym.st_value;

2417         gcore_initialized = B_TRUE;
2418 }

2420 #endif /* _KMDB */
```

****
711 Wed Aug 21 14:46:15 2013
new/usr/src/cmd/mdb/common/modules/genunix/gcore.h
3946 ::gcore
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Matthew Ahrens <mahrens@delphix.com>
****

```
/*
 * This file and its contents are supplied under the terms of the
 * Common Development and Distribution License ("CDDL"), version 1.0.
 * You may only use this file in accordance with the terms of version
 * 1.0 of the CDDL.
 *
 * A full copy of the text of the CDDL should have accompanied this
 * source.  A copy of the CDDL is also available via the Internet at
 * http://www.illumos.org/license/CDDL.
 */
/*
 * Copyright (c) 2013 by Delphix. All rights reserved.
 */

#ifndef _MDB_GCORE_H
#define _MDB_GCORE_H

#ifdef __cplusplus
extern "C" {
#endif

extern void gcore_init(void);
extern int gcore_dcmd(uintptr_t, uint_t, int, const mdb_arg_t *);

#ifdef __cplusplus
}
#endif

#endif /* _MDB_GCORE_H */
```

```
**********************************************************
  129260 Wed Aug 21 14:46:16 2013
new/usr/src/cmd/mdb/common/modules/genunix/genunix.c
3946 ::gcore
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Matthew Ahrens <mahrens@delphix.com>
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 * Copyright (c) 1999, 2010, Oracle and/or its affiliates. All rights reserved.
 * Copyright (c) 2013 Joyent, Inc. All rights reserved.
 * Copyright (c) 2013 by Delphix. All rights reserved.
 */

#include <mdb/mdb_param.h>
#include <mdb/mdb_modapi.h>
#include <mdb/mdb_ks.h>
#include <mdb/mdb_ctf.h>

#include <sys/types.h>
#include <sys/thread.h>
#include <sys/session.h>
#include <sys/user.h>
#include <sys/proc.h>
#include <sys/var.h>
#include <sys/t_lock.h>
#include <sys/callo.h>
#include <sys/priocntl.h>
#include <sys/class.h>
#include <sys/regset.h>
#include <sys/stack.h>
#include <sys/cpuvar.h>
#include <sys/vnode.h>
#include <sys/vfs.h>
#include <sys/flock_impl.h>
#include <sys/kmem_impl.h>
#include <sys/vmem_impl.h>
#include <sys/kstat.h>
#include <sys/dditypes.h>
#include <sys/ddi_impldefs.h>
#include <sys/sysmacros.h>
#include <sys/sysconf.h>
#include <sys/task.h>
#include <sys/project.h>
#include <sys/errorq_impl.h>
#include <sys/cred_impl.h>
```



```
#include <sys/zone.h>
#include <sys/panic.h>
#include <regex.h>
#include <sys/port_impl.h>

#include "avl.h"
#include "bio.h"
#include "bitset.h"
#include "combined.h"
#include "contract.h"
#include "cpupart_mdb.h"
#include "cred.h"
#include "ctxop.h"
#include "cyclic.h"
#include "damap.h"
#include "ddi_periodic.h"
#include "devinfo.h"
#include "findstack.h"
#include "fm.h"
#include "gcore.h"
#include "group.h"
#include "irm.h"
#include "kgrep.h"
#include "kmem.h"
#include "ldi.h"
#include "leaky.h"
#include "lgrp.h"
#include "list.h"
#include "log.h"
#include "mdi.h"
#include "memory.h"
#include "mmd.h"
#include "modhash.h"
#include "ndievents.h"
#include "net.h"
#include "netstack.h"
#include "nvpair.h"
#include "pg.h"
#include "rctl.h"
#include "sobj.h"
#include "streams.h"
#include "sysevent.h"
#include "taskq.h"
#include "thread.h"
#include "tsd.h"
#include "tsol.h"
#include "typegraph.h"
#include "vfs.h"
#include "zone.h"
#include "hotplug.h"

/*
 * Surely this is defined somewhere...
 */
#define NINTR           16

#define KILOS           10
#define MEGS            20
#define GIGS            30

#ifndef STACK_BIAS
#define STACK_BIAS      0
#endif

static char
pstat2ch(uchar_t state)
```

```
 126 {
 127         switch (state) {
 128                 case SSLEEP: return ('S');
 129                 case SRUN: return ('R');
 130                 case SZOMB: return ('Z');
 131                 case SIDL: return ('I');
 132                 case SONPROC: return ('O');
 133                 case SSTOP: return ('T');
 134                 case SWAIT: return ('W');
 135                 default: return ('?');
 136         }
 137 }
______unchanged_portion_omitted_

3845 static const mdb_dcmd_t dcmds[] = {

3847         /* from genunix.c */
3848         { "as2proc", ":", "convert as to proc_t address", as2proc },
3849         { "binding_hash_entry", ":", "print driver names hash table entry",
3850                 binding_hash_entry },
3851         { "callout", "?[-r|n] [-s|l] [-xhB] [-t | -ab nsec [-dkD]]"
3852             " [-C addr | -S seqid] [-f name|addr] [-p name| addr] [-T|L [-E]]"
3853             " [-FivVA]",
3854             "display callouts", callout, callout_help },
3855         { "calloutid", "[-d|v] xid", "print callout by extended id",
3856             calloutid, calloutid_help },
3857         { "class", NULL, "print process scheduler classes", class },
3858         { "cpuinfo", "?[-v]", "print CPUs and runnable threads", cpuinfo },
3859         { "did2thread", "? kt_did", "find kernel thread for this id",
3860                 did2thread },
3861         { "errorq", "?[-v]", "display kernel error queues", errorq },
3862         { "fd", ":[fd num]", "get a file pointer from an fd", fd },
3863         { "flipone", ":", "the vik_rev_level 2 special", flipone },
3864         { "lminfo", NULL, "print lock manager information", lminfo },
3865         { "ndi_event_hdl", "?", "print ndi_event_hdl", ndi_event_hdl },
3866         { "panicinfo", NULL, "print panic information", panicinfo },
3867         { "pid2proc", "?", "convert PID to proc_t address", pid2proc },
3868         { "project", NULL, "display kernel project(s)", project },
3869         { "ps", "[-fltzTP]", "list processes (and associated thr,lwp)", ps },
3870         { "pgrep", "[-x] [-n | -o] pattern",
3871                 "pattern match against all processes", pgrep },
3872         { "ptree", NULL, "print process tree", ptree },
3873         { "sysevent", "?[-sv]", "print sysevent pending or sent queue",
3874                 sysevent},
3875         { "sysevent_channel", "?", "print sysevent channel database",
3876                 sysevent_channel},
3877         { "sysevent_class_list", ":", "print sysevent class list",
3878                 sysevent_class_list},
3879         { "sysevent_subclass_list", ":",
3880                 "print sysevent subclass list", sysevent_subclass_list},
3881         { "system", NULL, "print contents of /etc/system file", sysfile },
3882         { "task", NULL, "display kernel task(s)", task },
3883         { "time", "[-dlx]", "display system time", time, time_help },
3884         { "vnode2path", ":[-F]", "vnode address to pathname", vnode2path },
3885         { "whereopen", ":", "given a vnode, dumps procs which have it open",
3886             whereopen },

3888         /* from bio.c */
3889         { "bufpagefind", ":addr", "find page_t on buf_t list", bufpagefind },

3891         /* from bitset.c */
3892         { "bitset", ":", "display a bitset", bitset, bitset_help },

3894         /* from contract.c */
3895         { "contract", "?", "display a contract", cmd_contract },
3896         { "ctevent", ":", "display a contract event", cmd_ctevent },
```



```
3897         { "ctid", ":", "convert id to a contract pointer", cmd_ctid },

3899         /* from cpupart.c */
3900         { "cpupart", "?[-v]", "print cpu partition info", cpupart },

3902         /* from cred.c */
3903         { "cred", ":[-v]", "display a credential", cmd_cred },
3904         { "credgrp", ":[-v]", "display cred_t groups", cmd_credgrp },
3905         { "credsid", ":[-v]", "display a credsid_t", cmd_credsid },
3906         { "ksidlist", ":[-v]", "display a ksidlist_t", cmd_ksidlist },

3908         /* from cyclic.c */
3909         { "cyccover", NULL, "dump cyclic coverage information", cyccover },
3910         { "cycid", "?", "dump a cyclic id", cycid },
3911         { "cycinfo", "?", "dump cyc_cpu info", cycinfo },
3912         { "cyclic", ":", "developer information", cyclic },
3913         { "cyctrace", "?", "dump cyclic trace buffer", cyctrace },

3915         /* from damap.c */
3916         { "damap", ":", "display a damap_t", damap, damap_help },

3918         /* from ddi_periodic.c */
3919         { "ddi_periodic", "?[-v]", "dump ddi_periodic_impl_t info", dprinfo },

3921         /* from devinfo.c */
3922         { "devbindings", "?[-qs] [device-name | major-num]",
3923             "print devinfo nodes bound to device-name or major-num",
3924             devbindings, devinfo_help },
3925         { "devinfo", ":[-qs]", "detailed devinfo of one node", devinfo,
3926             devinfo_help },
3927         { "devinfo_audit", ":[-v]", "devinfo configuration audit record",
3928             devinfo_audit },
3929         { "devinfo_audit_log", "?[-v]", "system wide devinfo configuration log",
3930             devinfo_audit_log },
3931         { "devinfo_audit_node", ":[-v]", "devinfo node configuration history",
3932             devinfo_audit_node },
3933         { "devinfo2driver", ":", "find driver name for this devinfo node",
3934             devinfo2driver },
3935         { "devnames", "?[-vm] [num]", "print devnames array", devnames },
3936         { "dev2major", "?<dev_t>", "convert dev_t to a major number",
3937             dev2major },
3938         { "dev2minor", "?<dev_t>", "convert dev_t to a minor number",
3939             dev2minor },
3940         { "devt", "?<dev_t>", "display a dev_t's major and minor numbers",
3941             devt },
3942         { "major2name", "?<major-num>", "convert major number to dev name",
3943             major2name },
3944         { "minornodes", ":", "given a devinfo node, print its minor nodes",
3945             minornodes },
3946         { "modctl2devinfo", ":", "given a modctl, list its devinfos",
3947             modctl2devinfo },
3948         { "name2major", "<dev-name>", "convert dev name to major number",
3949             name2major },
3950         { "prtconf", "?[-vpc]", "print devinfo tree", prtconf, prtconf_help },
3951         { "softstate", ":<instance>", "retrieve soft-state pointer",
3952             softstate },
3953         { "devinfo_fm", ":", "devinfo fault managment configuration",
3954             devinfo_fm },
3955         { "devinfo_fmce", ":", "devinfo fault managment cache entry",
3956             devinfo_fmce},

3958         /* from findstack.c */
3959         { "findstack", ":[-v]", "find kernel thread stack", findstack },
3960         { "findstack_debug", NULL, "toggle findstack debugging",
3961                 findstack_debug },
3962         { "stacks", "?[-afiv] [-c func] [-C func] [-m module] [-M module] "
```

```
3963                 "[-s sobj | -S sobj] [-t tstate | -T tstate]",
3964                 "print unique kernel thread stacks",
3965                 stacks, stacks_help },

3967         /* from fm.c */
3968         { "ereport", "[-v]", "print ereports logged in dump",
3969             ereport },

3971         /* from group.c */
3972         { "group", "?[-q]", "display a group", group},

3974         /* from hotplug.c */
3975         { "hotplug", "?[-p]", "display a registered hotplug attachment",
3976             hotplug, hotplug_help },

3978         /* from irm.c */
3979         { "irmpools", NULL, "display interrupt pools", irmpools_dcmd },
3980         { "irmreqs", NULL, "display interrupt requests in an interrupt pool",
3981             irmreqs_dcmd },
3982         { "irmreq", NULL, "display an interrupt request", irmreq_dcmd },

3984         /* from kgrep.c + genunix.c */
3985         { "kgrep", KGREP_USAGE, "search kernel as for a pointer", kgrep,
3986                 kgrep_help },

3988         /* from kmem.c */
3989         { "allocdby", ":", "given a thread, print its allocated buffers",
3990                 allocdby },
3991         { "bufctl", ":[-vh] [-a addr] [-c caller] [-e earliest] [-l latest] "
3992                 "[-t thd]", "print or filter a bufctl", bufctl, bufctl_help },
3993         { "freedby", ":", "given a thread, print its freed buffers", freedby },
3994         { "kmalog", "?[ fail | slab ]",
3995             "display kmem transaction log and stack traces", kmalog },
3996         { "kmastat", "[-kmg]", "kernel memory allocator stats",
3997             kmastat },
3998         { "kmausers", "?[-ef] [cache ...]", "current medium and large users "
3999                 "of the kmem allocator", kmausers, kmausers_help },
4000         { "kmem_cache", "?[-n name]",
4001                 "print kernel memory caches", kmem_cache, kmem_cache_help},
4002         { "kmem_slabs", "?[-v] [-n cache] [-N cache] [-b maxbins] "
4003                 "[-B minbinsize]", "display slab usage per kmem cache",
4004                 kmem_slabs, kmem_slabs_help },
4005         { "kmem_debug", NULL, "toggle kmem dcmd/walk debugging", kmem_debug },
4006         { "kmem_log", "?[-b]", "dump kmem transaction log", kmem_log },
4007         { "kmem_verify", "?", "check integrity of kmem-managed memory",
4008                 kmem_verify },
4009         { "vmem", "?", "print a vmem_t", vmem },
4010         { "vmem_seg", ":[-sv] [-c caller] [-e earliest] [-l latest] "
4011                 "[-m minsize] [-M maxsize] [-t thread] [-T type]",
4012                 "print or filter a vmem_seg", vmem_seg, vmem_seg_help },
4013         { "whatthread", ":[-v]", "print threads whose stack contains the "
4014                 "given address", whatthread },

4016         /* from ldi.c */
4017         { "ldi_handle", "?[-i]", "display a layered driver handle",
4018             ldi_handle, ldi_handle_help },
4019         { "ldi_ident", NULL, "display a layered driver identifier",
4020             ldi_ident, ldi_ident_help },

4022         /* from leaky.c + leaky_subr.c */
4023         { "findleaks", FINDLEAKS_USAGE,
4024             "search for potential kernel memory leaks", findleaks,
4025             findleaks_help },

4027         /* from lgrp.c */
4028         { "lgrp", "?[-q] [-p | -Pih]", "display an lgrp", lgrp},
```



```
4029         { "lgrp_set", "", "display bitmask of lgroups as a list", lgrp_set},

4031         /* from log.c */
4032         { "msgbuf", "?[-v]", "print most recent console messages", msgbuf },

4034         /* from mdi.c */
4035         { "mdipi", NULL, "given a path, dump mdi_pathinfo "
4036                 "and detailed pi_prop list", mdipi },
4037         { "mdiprops", NULL, "given a pi_prop, dump the pi_prop list",
4038                 mdiprops },
4039         { "mdiphci", NULL, "given a phci, dump mdi_phci and "
4040                 "list all paths", mdiphci },
4041         { "mdivhci", NULL, "given a vhci, dump mdi_vhci and list "
4042                 "all phcis", mdivhci },
4043         { "mdiclient_paths", NULL, "given a path, walk mdi_pathinfo "
4044                 "client links", mdiclient_paths },
4045         { "mdiphci_paths", NULL, "given a path, walk through mdi_pathinfo "
4046                 "phci links", mdiphci_paths },
4047         { "mdiphcis", NULL, "given a phci, walk through mdi_phci ph_next links",
4048                 mdiphcis },

4050         /* from memory.c */
4051         { "addr2smap", ":[offset]", "translate address to smap", addr2smap },
4052         { "memlist", "?[-iav]", "display a struct memlist", memlist },
4053         { "memstat", NULL, "display memory usage summary", memstat },
4054         { "page", "?", "display a summarized page_t", page },
4055         { "pagelookup", "?[-v vp] [-o offset]",
4056                 "find the page_t with the name {vp, offset}",
4057                 pagelookup, pagelookup_help },
4058         { "page_num2pp", ":", "find the page_t for a given page frame number",
4059                 page_num2pp },
4060         { "pmap", ":[-q]", "print process memory map", pmap },
4061         { "seg", ":", "print address space segment", seg },
4062         { "swapinfo", "?", "display a struct swapinfo", swapinfof },
4063         { "vnode2smap", ":[offset]", "translate vnode to smap", vnode2smap },

4065         /* from mmd.c */
4066         { "multidata", ":[-sv]", "display a summarized multidata_t",
4067                 multidata },
4068         { "pattbl", ":", "display a summarized multidata attribute table",
4069                 pattbl },
4070         { "pattr2multidata", ":", "print multidata pointer from pattr_t",
4071                 pattr2multidata },
4072         { "pdesc2slab", ":", "print pdesc slab pointer from pdesc_t",
4073                 pdesc2slab },
4074         { "pdesc_verify", ":", "verify integrity of a pdesc_t", pdesc_verify },
4075         { "slab2multidata", ":", "print multidata pointer from pdesc_slab_t",
4076                 slab2multidata },

4078         /* from modhash.c */
4079         { "modhash", "?[-ceht] [-k key] [-v val] [-i index]",
4080                 "display information about one or all mod_hash structures",
4081                 modhash, modhash_help },
4082         { "modent", ":[-k | -v | -t type]",
4083                 "display information about a mod_hash_entry", modent,
4084                 modent_help },

4086         /* from net.c */
4087         { "dladm", "?<sub-command> [flags]", "show data link information",
4088                 dladm, dladm_help },
4089         { "mi", ":[-p] [-d | -m]", "filter and display MI object or payload",
4090                 mi },
4091         { "netstat", "[-arv] [-f inet | inet6 | unix] [-P tcp | udp | icmp]",
4092                 "show network statistics", netstat },
4093         { "sonode", "?[-f inet | inet6 | unix | #] "
4094                 "[-t stream | dgram | raw | #] [-p #]",
```

```
4095                 "filter and display sonode", sonode },

4097         /* from netstack.c */
4098         { "netstack", "", "show stack instances", netstack },

4100         /* from nvpair.c */
4101         { NVPAIR_DCMD_NAME, NVPAIR_DCMD_USAGE, NVPAIR_DCMD_DESCR,
4102                 nvpair_print },
4103         { NVLIST_DCMD_NAME, NVLIST_DCMD_USAGE, NVLIST_DCMD_DESCR,
4104                 print_nvlist },

4106         /* from pg.c */
4107         { "pg", "?[-q]", "display a pg", pg},

4109         /* from rctl.c */
4110         { "rctl_dict", "?", "print systemwide default rctl definitions",
4111                 rctl_dict },
4112         { "rctl_list", ":[handle]", "print rctls for the given proc",
4113                 rctl_list },
4114         { "rctl", ":[handle]", "print a rctl_t, only if it matches the handle",
4115                 rctl },
4116         { "rctl_validate", ":[-v] [-n #]", "test resource control value "
4117                 "sequence", rctl_validate },

4119         /* from sobj.c */
4120         { "rwlock", ":", "dump out a readers/writer lock", rwlock },
4121         { "mutex", ":[-f]", "dump out an adaptive or spin mutex", mutex,
4122                 mutex_help },
4123         { "sobj2ts", ":", "perform turnstile lookup on synch object", sobj2ts },
4124         { "wchaninfo", "?[-v]", "dump condition variable", wchaninfo },
4125         { "turnstile", "?", "display a turnstile", turnstile },

4127         /* from stream.c */
4128         { "mblk", ":[-q|v] [-f|F flag] [-t|T type] [-l|L|B len] [-d dbaddr]",
4129                 "print an mblk", mblk_prt, mblk_help },
4130         { "mblk_verify", "?", "verify integrity of an mblk", mblk_verify },
4131         { "mblk2dblk", ":", "convert mblk_t address to dblk_t address",
4132                 mblk2dblk },
4133         { "q2otherq", ":", "print peer queue for a given queue", q2otherq },
4134         { "q2rdq", ":", "print read queue for a given queue", q2rdq },
4135         { "q2syncq", ":", "print syncq for a given queue", q2syncq },
4136         { "q2stream", ":", "print stream pointer for a given queue", q2stream },
4137         { "q2wrq", ":", "print write queue for a given queue", q2wrq },
4138         { "queue", ":[-q|v] [-m mod] [-f flag] [-F flag] [-s syncq_addr]",
4139                 "filter and display STREAM queue", queue, queue_help },
4140         { "stdata", ":[-q|v] [-f flag] [-F flag]",
4141                 "filter and display STREAM head", stdata, stdata_help },
4142         { "str2mate", ":", "print mate of this stream", str2mate },
4143         { "str2wrq", ":", "print write queue of this stream", str2wrq },
4144         { "stream", ":", "display STREAM", stream },
4145         { "strftevent", ":", "print STREAMS flow trace event", strftevent },
4146         { "syncq", ":[-q|v] [-f flag] [-F flag] [-t type] [-T type]",
4147                 "filter and display STREAM sync queue", syncq, syncq_help },
4148         { "syncq2q", ":", "print queue for a given syncq", syncq2q },

4150         /* from taskq.c */
4151         { "taskq", ":[-atT] [-m min_maxq] [-n name]",
4152             "display a taskq", taskq, taskq_help },
4153         { "taskq_entry", ":", "display a taskq_ent_t", taskq_ent },

4155         /* from thread.c */
4156         { "thread", "?[-bdfimps]", "display a summarized kthread_t", thread,
4157                 thread_help },
4158         { "threadlist", "?[-t] [-v [count]]",
4159                 "display threads and associated C stack traces", threadlist,
4160                 threadlist_help },
```



```
4161         { "stackinfo", "?[-h|-a]", "display kthread_t stack usage", stackinfo,
4162                 stackinfo_help },

4164         /* from tsd.c */
4165         { "tsd", ":-k key", "print tsd[key-1] for this thread", ttotsd },
4166         { "tsdtot", ":", "find thread with this tsd", tsdtot },

4168         /*
4169          * typegraph does not work under kmdb, as it requires too much memory
4170          * for its internal data structures.
4171          */
4172 #ifndef _KMDB
4173         /* from typegraph.c */
4174         { "findlocks", ":", "find locks held by specified thread", findlocks },
4175         { "findfalse", "?[-v]", "find potentially falsely shared structures",
4176                 findfalse },
4177         { "typegraph", NULL, "build type graph", typegraph },
4178         { "istype", ":type", "manually set object type", istype },
4179         { "notype", ":", "manually clear object type", notype },
4180         { "whattype", ":", "determine object type", whattype },
4181 #endif

4183         /* from vfs.c */
4184         { "fsinfo", "?[-v]", "print mounted filesystems", fsinfo },
4185         { "pfiles", ":[-fp]", "print process file information", pfiles,
4186                 pfiles_help },

4188         /* from zone.c */
4189         { "zone", "?[-r [-v]]", "display kernel zone(s)", zoneprt },
4190         { "zsd", ":[-v] [zsd_key]", "display zone-specific-data entries for "
4191             "selected zones", zsd },

4193 #ifndef _KMDB
4194         { "gcore", NULL, "generate a user core for the given process",
4195             gcore_dcmd },
4196 #endif

4198         { NULL }
4199 };
______unchanged_portion_omitted_

4607 const mdb_modinfo_t *
4608 _mdb_init(void)
4609 {
4610         kmem_init();

4612         (void) mdb_callback_add(MDB_CALLBACK_STCHG,
4613             genunix_statechange_cb, NULL);

4615 #ifndef _KMDB
4616         gcore_init();
4617 #endif

4619         return (&modinfo);
4620 }
______unchanged_portion_omitted_
```

```
**********************************************************
    2236 Wed Aug 21 14:46:18 2013
new/usr/src/cmd/mdb/intel/amd64/genunix/Makefile
3946 ::gcore
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Matthew Ahrens <mahrens@delphix.com>
**********************************************************
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END


#
#
# Copyright 2007 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#
# Copyright (c) 2013 by Delphix. All rights reserved.
#

MODULE = genunix.so
MDBTGT = kvm

include ../../../common/modules/genunix/Makefile.files

COMMONSRCS = \
        $(GENUNIX_SRCS)

KMODSRCS = \
        $(COMMONSRCS)

MODSRCS = \
        $(COMMONSRCS) \
        typegraph.c

#
# This signals that $(KMODSRCS) != $(MODSRCS).  Typegraph is not usable under
# kmdb.  As such, we don't bother compiling it.
KMOD_SOURCES_DIFFERENT=$(POUND_SIGN)

include ../../../../Makefile.cmd
include ../../../../Makefile.cmd.64
include ../../Makefile.amd64
include ../../../Makefile.module

dmod/$(MODULE) := LDLIBS += -lm -lproc
dmod/$(MODULE) := LDLIBS += -lm

#
# We are not actually hardwiring some dependency on i86pc, we just need to
```



```
# include sys/param.h with _MACHDEP defined, and this forces the inclusion of
# machparam.h, even though we don't use anything there.  This is a temporary
# kludge until we invent -DDONTINCLUDEMACHPARAM or something.
#
CPPFLAGS += -I$(SRC)/uts/i86pc
CPPFLAGS += -I$(SRC)/uts/i86xpv

# Needed to include c2/audit.h (from cred.h)
CPPFLAGS += -I$(SRC)/uts/common

# Needed to find include file mutex_impl.h
CPPFLAGS += -I$(SRC)/uts/intel

CERRWARN += -_gcc=-Wno-char-subscripts
CERRWARN += -_gcc=-Wno-unused-label
CERRWARN += -_gcc=-Wno-uninitialized
CERRWARN += -_gcc=-Wno-parentheses
CERRWARN += -_gcc=-Wno-type-limits

LINTFLAGS64 += -erroff=E_EMPTY_TRANSLATION_UNIT
```

```
**********************************************************
    2194 Wed Aug 21 14:46:19 2013
new/usr/src/cmd/mdb/intel/ia32/genunix/Makefile
3946 ::gcore
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Matthew Ahrens <mahrens@delphix.com>
**********************************************************
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END

#
# Copyright 2007 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#
# Copyright (c) 2013 by Delphix. All rights reserved.
#

MODULE = genunix.so
MDBTGT = kvm

include ../../../common/modules/genunix/Makefile.files

COMMONSRCS = \
        $(GENUNIX_SRCS)

KMODSRCS = \
        $(COMMONSRCS)

MODSRCS = \
        $(COMMONSRCS) \
        typegraph.c

#
# This signals that $(KMODSRCS) != $(MODSRCS).  Typegraph is not usable under
# kmdb.  As such, we don't bother compiling it.
KMOD_SOURCES_DIFFERENT=$(POUND_SIGN)

include ../../../../Makefile.cmd
include ../../Makefile.ia32
include ../../../Makefile.module

dmod/$(MODULE) := LDLIBS += -lm -lproc
dmod/$(MODULE) := LDLIBS += -lm

#
# We are not actually hardwiring some dependency on i86pc, we just need to
# include sys/param.h with _MACHDEP defined, and this forces the inclusion of
# machparam.h, even though we don't use anything there.  This is a temporary
# kludge until we invent -DDONTINCLUDEMACHPARAM or something.
```



```
#
CPPFLAGS += -I$(SRC)/uts/i86pc
CPPFLAGS += -I$(SRC)/uts/i86xpv

# Needed to include c2/audit.h (from cred.h)
CPPFLAGS += -I$(SRC)/uts/common

# Needed to find include file mutex_impl.h
CPPFLAGS += -I$(SRC)/uts/intel

CERRWARN += -_gcc=-Wno-char-subscripts
CERRWARN += -_gcc=-Wno-unused-label
CERRWARN += -_gcc=-Wno-uninitialized
CERRWARN += -_gcc=-Wno-parentheses
CERRWARN += -_gcc=-Wno-type-limits

LINTFLAGS += -erroff=E_EMPTY_TRANSLATION_UNIT
```

```
**********************************************************
   14721 Wed Aug 21 14:46:20 2013
new/usr/src/cmd/ptools/ppriv/ppriv.c
3946 ::gcore
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Matthew Ahrens <mahrens@delphix.com>
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright (c) 2003, 2010, Oracle and/or its affiliates. All rights reserved.
 */
/*
 * Copyright (c) 2013 by Delphix. All rights reserved.
 */
/*
 *
 * Program to examine or set process privileges.
 */

#include <stdio.h>
#include <stdio_ext.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <string.h>
#include <limits.h>
#include <sys/types.h>
#include <libproc.h>
#include <priv.h>
#include <errno.h>
#include <ctype.h>

#include <locale.h>
#include <langinfo.h>

static int      look(char *);
static void     perr(char *);
static void     usage(void);
static void     loadprivinfo(void);
static int      parsespec(const char *);
static void     privupdate(prpriv_t *, const char *);
static void     privupdate_self(void);
static int      dumppriv(char **);
static void     flags2str(uint_t);

static char             *command;
static char             *procname;
```



```
static boolean_t        verb = B_FALSE;
static boolean_t        set = B_FALSE;
static boolean_t        exec = B_FALSE;
static boolean_t        Don = B_FALSE;
static boolean_t        Doff = B_FALSE;
static boolean_t        list = B_FALSE;
static boolean_t        mac_aware = B_FALSE;
static boolean_t        pfexec = B_FALSE;
static boolean_t        xpol = B_FALSE;
static int              mode = PRIV_STR_PORT;

int
main(int argc, char **argv)
{
        int rc = 0;
        int opt;
        struct rlimit rlim;

        (void) setlocale(LC_ALL, "");
        (void) textdomain(TEXT_DOMAIN);

        if ((command = strrchr(argv[0], '/')) != NULL)
                command++;
        else
                command = argv[0];

        while ((opt = getopt(argc, argv, "lDMNPevs:xS")) != EOF) {
                switch (opt) {
                case 'l':
                        list = B_TRUE;
                        break;
                case 'D':
                        set = B_TRUE;
                        Don = B_TRUE;
                        break;
                case 'M':
                        mac_aware = B_TRUE;
                        break;
                case 'N':
                        set = B_TRUE;
                        Doff = B_TRUE;
                        break;
                case 'P':
                        set = B_TRUE;
                        pfexec = B_TRUE;
                        break;
                case 'e':
                        exec = B_TRUE;
                        break;
                case 'S':
                        mode = PRIV_STR_SHORT;
                        break;
                case 'v':
                        verb = B_TRUE;
                        mode = PRIV_STR_LIT;
                        break;
                case 's':
                        set = B_TRUE;
                        if ((rc = parsespec(optarg)) != 0)
                                return (rc);
                        break;
                case 'x':
                        set = B_TRUE;
                        xpol = B_TRUE;
                        break;
                default:
```

```
125                         usage();
126                         /*NOTREACHED*/
127                 }
128         }

130         argc -= optind;
131         argv += optind;

133         if ((argc < 1 && !list) || Doff && Don || list && (set || exec) ||
134             (mac_aware && !exec))
135                 usage();

137         /*
138          * Make sure we'll have enough file descriptors to handle a target
139          * that has many many mappings.
140          */
141         if (getrlimit(RLIMIT_NOFILE, &rlim) == 0) {
142                 rlim.rlim_cur = rlim.rlim_max;
143                 (void) setrlimit(RLIMIT_NOFILE, &rlim);
144                 (void) enable_extended_FILE_stdio(-1, -1);
145         }

147         if (exec) {
148                 privupdate_self();
149                 rc = execvp(argv[0], &argv[0]);
150                 (void) fprintf(stderr, "%s: %s: %s\n", command, argv[0],
151                     strerror(errno));
152         } else if (list) {
153                 rc = dumppriv(argv);
154         } else {
155                 while (argc-- > 0)
156                         rc += look(*argv++);
157         }

159         return (rc);
160 }

162 static int
163 look(char *arg)
164 {
161         static size_t pprivsz = sizeof (prpriv_t);
162         static prpriv_t *ppriv;

165         struct ps_prochandle *Pr;
166         int gcode;
167         size_t sz;
168         void *pdata;
169         char *x;
170         int i;
171         boolean_t nodata;
172         prpriv_t *ppriv;

174         procname = arg;         /* for perr() */

176         if ((Pr = proc_arg_grab(arg, set ? PR_ARG_PIDS : PR_ARG_ANY,
177             PGRAB_RETAIN | PGRAB_FORCE | (set ? 0 : PGRAB_RDONLY) |
178             PGRAB_NOSTOP, &gcode)) == NULL) {
179                 (void) fprintf(stderr, "%s: cannot examine %s: %s\n",
180                     command, arg, Pgrab_error(gcode));
181                 return (1);
182         }

184         if (Ppriv(Pr, &ppriv) == -1) {
182         if (ppriv == NULL)
183                 ppriv = malloc(pprivsz);
```



```
185         if (Ppriv(Pr, ppriv, pprivsz) == -1) {
185                 perr(command);
186                 Prelease(Pr, 0);
187                 return (1);
188         }

189         sz = PRIV_PRPRIV_SIZE(ppriv);

191         /*
192          * The ppriv fields are unsigned and may overflow, so check them
193          * separately.  Size must be word aligned, so check that too.
194          * Make sure size is "smallish" too.
195          */
196         if ((sz & 3) || ppriv->pr_nsets == 0 ||
197             sz / ppriv->pr_nsets < ppriv->pr_setsize ||
198             ppriv->pr_infosize > sz || sz > 1024 * 1024) {
199                 (void) fprintf(stderr,
200                     "%s: %s: bad PRNOTES section, size = %lx\n",
201                     command, arg, (long)sz);
202                 Prelease(Pr, 0);
203                 free(ppriv);
204                 return (1);
205         }

208         if (sz > pprivsz) {
209                 ppriv = realloc(ppriv, sz);

211                 if (ppriv == NULL || Ppriv(Pr, ppriv, sz) != sz) {
212                         perr(command);
213                         Prelease(Pr, 0);
214                         return (1);
215                 }
216                 pprivsz = sz;
217         }

207         if (set) {
208                 privupdate(ppriv, arg);
209                 if (Psetpriv(Pr, ppriv) != 0) {
210                         perr(command);
211                         Prelease(Pr, 0);
212                         free(ppriv);
213                         return (1);
214                 }
215                 Prelease(Pr, 0);
216                 free(ppriv);
217                 return (0);
218         }

220         if (Pstate(Pr) == PS_DEAD) {
221                 (void) printf("core '%s' of %d:\t%.70s\n",
222                     arg, (int)Ppsinfo(Pr)->pr_pid, Ppsinfo(Pr)->pr_psargs);
223                 pdata = Pprivinfo(Pr);
224                 nodata = Pstate(Pr) == PS_DEAD && pdata == NULL;
225         } else {
226                 (void) printf("%d:\t%.70s\n",
227                     (int)Ppsinfo(Pr)->pr_pid, Ppsinfo(Pr)->pr_psargs);
228                 pdata = NULL;
229                 nodata = B_FALSE;
230         }

232         x = (char *)ppriv + sz - ppriv->pr_infosize;
233         while (x < (char *)ppriv + sz) {
234                 /* LINTED: alignment */
235                 priv_info_t *pi = (priv_info_t *)x;
236                 priv_info_uint_t *pii;
```

```
238                 switch (pi->priv_info_type) {
239                 case PRIV_INFO_FLAGS:
240                         /* LINTED: alignment */
241                         pii = (priv_info_uint_t *)x;
242                         (void) printf("flags =");
243                         flags2str(pii->val);
244                         (void) putchar('\n');
245                         break;
246                 default:
247                         (void) fprintf(stderr, "%s: unknown priv_info: %d\n",
248                             arg, pi->priv_info_type);
249                         break;
250                 }
251                 if (pi->priv_info_size > ppriv->pr_infosize ||
252                     pi->priv_info_size <=  sizeof (priv_info_t) ||
253                     (pi->priv_info_size & 3) != 0) {
254                         (void) fprintf(stderr, "%s: bad priv_info_size: %u\n",
255                             arg, pi->priv_info_size);
256                         break;
257                 }
258                 x += pi->priv_info_size;
259         }

261         for (i = 0; i < ppriv->pr_nsets; i++) {
262                 extern const char *__priv_getsetbynum(const void *, int);
263                 const char *setnm = pdata ? __priv_getsetbynum(pdata, i) :
264                     priv_getsetbynum(i);
265                 priv_chunk_t *pc =
266                     (priv_chunk_t *)&ppriv->pr_sets[ppriv->pr_setsize * i];


269                 (void) printf("\t%c: ", setnm && !nodata ? *setnm : '?');
270                 if (!nodata) {
271                         extern char *__priv_set_to_str(void *,
272                             const priv_set_t *, char, int);
273                         priv_set_t *pset = (priv_set_t *)pc;

275                         char *s;

277                         if (pdata)
278                                 s = __priv_set_to_str(pdata, pset, ',', mode);
279                         else
280                                 s = priv_set_to_str(pset, ',', mode);
281                         (void) puts(s);
282                         free(s);
283                 } else {
284                         int j;
285                         for (j = 0; j < ppriv->pr_setsize; j++)
286                                 (void) printf("%08x", pc[j]);
287                         (void) putchar('\n');
288                 }
289         }
290         Prelease(Pr, 0);
291         free(ppriv);
292         return (0);
293 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    2896 Wed Aug 21 14:46:21 2013
new/usr/src/lib/libproc/Makefile.com
3946 ::gcore
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Matthew Ahrens <mahrens@delphix.com>
**********************************************************
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#
#
# Copyright (c) 1997, 2010, Oracle and/or its affiliates. All rights reserved.
# Copyright 2012 DEY Storage Systems, Inc.  All rights reserved.
# Copyright (c) 2013 by Delphix. All rights reserved.
#

LIBRARY = libproc.a
VERS = .1

CMNOBJS =       \
        P32ton.o        \
        Pcontrol.o      \
        Pcore.o         \
        Pexecname.o     \
        Pfdinfo.o       \
        Pgcore.o        \
        Pidle.o         \
        Pisprocdir.o    \
        Plwpregs.o      \
        Pservice.o      \
        Psymtab.o       \
        Psymtab_machelf32.o \
        $(CMNOBJS64)    \
        Pscantext.o     \
        Pstack.o        \
        Psyscall.o      \
        Putil.o         \
        Pzone.o         \
        pr_door.o       \
        pr_exit.o       \
        pr_fcntl.o      \
        pr_getitimer.o  \
        pr_getrctl.o    \
        pr_getrlimit.o  \
        pr_getsockname.o \
        pr_ioctl.o      \
        pr_lseek.o      \
        pr_memcntl.o    \
        pr_meminfo.o    \
```



```
        pr_mmap.o       \
        pr_open.o       \
        pr_pbind.o      \
        pr_rename.o     \
        pr_sigaction.o  \
        pr_stat.o       \
        pr_statvfs.o    \
        pr_tasksys.o    \
        pr_waitid.o     \
        proc_get_info.o \
        proc_names.o    \
        proc_arg.o      \
        proc_set.o      \
        proc_stdio.o

ISAOBJS =       \
        Pisadep.o

OBJECTS = $(CMNOBJS) $(ISAOBJS) $(SAVEOBJS)

# include library definitions
include ../../Makefile.lib
include ../../Makefile.rootfs

SRCS =          $(CMNOBJS:%.o=../common/%.c) $(ISAOBJS:%.o=%.c)

LIBS =          $(DYNLIB) $(LINTLIB)
LDLIBS +=       -lrtld_db -lelf -lctf -lc
C99MODE =       $(C99_ENABLE)
CPPFLAGS +=     $($(MACH64)_CPPFLAGS)

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

CFLAGS +=       $(CCVERBOSE)
CPPFLAGS +=     -I$(SRCDIR)

CERRWARN +=     -_gcc=-Wno-uninitialized
CERRWARN +=     -_gcc=-Wno-parentheses
CERRWARN +=     -_gcc=-Wno-type-limits
CERRWARN +=     -_gcc=-Wno-unused-label

# All interfaces are interposable, therefore don't allow direct binding to
# libproc.  Disable libproc from directly binding to itself, but allow libperl
# to directly bind to its dependencies (ie. map -Bdirect -> -zdirect).  Ensure
# lazy loading is established (which is enabled automatically with -Bdirect).
BDIRECT =
DYNFLAGS +=     $(BNODIRECT) $(ZDIRECT) $(ZLAZYLOAD)

.KEEP_STATE:

all: $(LIBS)

lint: lintcheck

# include library targets
include ../../Makefile.targ

objs/%.o pics/%.o: %.c
        $(COMPILE.c) -o $@ $<
        $(POST_PROCESS_O)

objs/%.o pics/%.o: $(SRC)/common/saveargs/%.c
        $(COMPILE.c) -o $@ $<
        $(POST_PROCESS_O)
```

```
**********************************************************
   89973 Wed Aug 21 14:46:22 2013
new/usr/src/lib/libproc/common/Pcontrol.c
3946 ::gcore
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Matthew Ahrens <mahrens@delphix.com>
**********************************************************
```

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

/*
 * Copyright 2010 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 *
 * Portions Copyright 2007 Chad Mynhier
 * Copyright 2012 DEY Storage Systems, Inc.  All rights reserved.
 * Copyright (c) 2013 by Delphix. All rights reserved.
 */

#include <assert.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <ctype.h>
#include <fcntl.h>
#include <string.h>
#include <strings.h>
#include <memory.h>
#include <errno.h>
#include <dirent.h>
#include <limits.h>
#include <signal.h>
#include <atomic.h>
#include <zone.h>
#include <sys/types.h>
#include <sys/uio.h>
#include <sys/stat.h>
#include <sys/resource.h>
#include <sys/param.h>
#include <sys/stack.h>
#include <sys/fault.h>
#include <sys/syscall.h>
#include <sys/sysmacros.h>
#include <sys/systeminfo.h>

#include "libproc.h"
#include "Pcontrol.h"
#include "Putil.h"
```



```
#include "P32ton.h"

int     _libproc_debug;         /* set non-zero to enable debugging printfs */
int     _libproc_no_qsort;      /* set non-zero to inhibit sorting */
                                /* of symbol tables */
int     _libproc_incore_elf;    /* only use in-core elf data */

sigset_t blockable_sigs;        /* signals to block when we need to be safe */
static  int     minfd;  /* minimum file descriptor returned by dupfd(fd, 0) */
char    procfs_path[PATH_MAX] = "/proc";

/*
 * Function prototypes for static routines in this module.
 */
static  void    deadcheck(struct ps_prochandle *);
static  void    restore_tracing_flags(struct ps_prochandle *);
static  void    Lfree_internal(struct ps_prochandle *, struct ps_lwphandle *);
static  prheader_t *read_lfile(struct ps_prochandle *, const char *);

/*
 * Ops vector functions for live processes.
 * Read/write interface for live processes: just pread/pwrite the
 * /proc/<pid>/as file:
 */

/*ARGSUSED*/
static ssize_t
Pread_live(struct ps_prochandle *P, void *buf, size_t n, uintptr_t addr,
    void *data)
{
        return (pread(P->asfd, buf, n, (off_t)addr));
}

/*ARGSUSED*/
static ssize_t
Pwrite_live(struct ps_prochandle *P, const void *buf, size_t n, uintptr_t addr,
    void *data)
{
        return (pwrite(P->asfd, buf, n, (off_t)addr));
}

/*ARGSUSED*/
static int
Pread_maps_live(struct ps_prochandle *P, prmap_t **Pmapp, ssize_t *nmapp,
    void *data)
{
        char mapfile[PATH_MAX];
        int mapfd;
        struct stat statb;
        ssize_t nmap;
        prmap_t *Pmap = NULL;
static const ps_rwops_t P_live_ops = { Pread_live, Pwrite_live };

        (void) snprintf(mapfile, sizeof (mapfile), "%s/%d/map",
            procfs_path, (int)P->pid);
        if ((mapfd = open(mapfile, O_RDONLY)) < 0 ||
            fstat(mapfd, &statb) != 0 ||
            statb.st_size < sizeof (prmap_t) ||
            (Pmap = malloc(statb.st_size)) == NULL ||
            (nmap = pread(mapfd, Pmap, statb.st_size, 0L)) <= 0 ||
            (nmap /= sizeof (prmap_t)) == 0) {
                if (Pmap != NULL)
                        free(Pmap);
                if (mapfd >= 0)
```

```
121                         (void) close(mapfd);
122                 Preset_maps(P); /* utter failure; destroy tables */
123                 return (-1);
124         }
125         (void) close(mapfd);

127         *Pmapp = Pmap;
128         *nmapp = nmap;

130         return (0);
131 }

133 /*ARGSUSED*/
134 static void
135 Pread_aux_live(struct ps_prochandle *P, auxv_t **auxvp, int *nauxp, void *data)
136 {
137         char auxfile[64];
138         int fd;
139         struct stat statb;
140         auxv_t *auxv;
141         ssize_t naux;

143         (void) snprintf(auxfile, sizeof (auxfile), "%s/%d/auxv",
144             procfs_path, (int)P->pid);
145         if ((fd = open(auxfile, O_RDONLY)) < 0) {
146                 dprintf("%s: failed to open %s: %s\n",
147                     __func__, auxfile, strerror(errno));
148                 return;
149         }

151         if (fstat(fd, &statb) == 0 &&
152             statb.st_size >= sizeof (auxv_t) &&
153             (auxv = malloc(statb.st_size + sizeof (auxv_t))) != NULL) {
154                 if ((naux = read(fd, auxv, statb.st_size)) < 0 ||
155                     (naux /= sizeof (auxv_t)) < 1) {
156                         dprintf("%s: read failed: %s\n",
157                             __func__, strerror(errno));
158                         free(auxv);
159                 } else {
160                         auxv[naux].a_type = AT_NULL;
161                         auxv[naux].a_un.a_val = 0L;

163                         *auxvp = auxv;
164                         *nauxp = (int)naux;
165                 }
166         }

168         (void) close(fd);
169 }

171 /*ARGSUSED*/
172 static int
173 Pcred_live(struct ps_prochandle *P, prcred_t *pcrp, int ngroups, void *data)
174 {
175         return (proc_get_cred(P->pid, pcrp, ngroups));
176 }

178 /*ARGSUSED*/
179 static int
180 Ppriv_live(struct ps_prochandle *P, prpriv_t **pprv, void *data)
181 {
182         prpriv_t *pp;

184         pp = proc_get_priv(P->pid);
185         if (pp == NULL) {
186                 return (-1);
```



```
187         }

189         *pprv = pp;
190         return (0);
191 }

193 /*ARGSUSED*/
194 static const psinfo_t *
195 Ppsinfo_live(struct ps_prochandle *P, psinfo_t *psinfo, void *data)
196 {
197         if (proc_get_psinfo(P->pid, psinfo) == -1)
198                 return (NULL);

200         return (psinfo);
201 }

203 /*ARGSUSED*/
204 static prheader_t *
205 Plstatus_live(struct ps_prochandle *P, void *data)
206 {
207         return (read_lfile(P, "lstatus"));
208 }

210 /*ARGSUSED*/
211 static prheader_t *
212 Plpsinfo_live(struct ps_prochandle *P, void *data)
213 {
214         return (read_lfile(P, "lpsinfo"));
215 }

217 /*ARGSUSED*/
218 static char *
219 Pplatform_live(struct ps_prochandle *P, char *s, size_t n, void *data)
220 {
221         if (sysinfo(SI_PLATFORM, s, n) == -1)
222                 return (NULL);
223         return (s);
224 }

226 /*ARGSUSED*/
227 static int
228 Puname_live(struct ps_prochandle *P, struct utsname *u, void *data)
229 {
230         return (uname(u));
231 }

233 /*ARGSUSED*/
234 static char *
235 Pzonename_live(struct ps_prochandle *P, char *s, size_t n, void *data)
236 {
237         if (getzonenamebyid(P->status.pr_zoneid, s, n) < 0)
238                 return (NULL);
239         s[n - 1] = '\0';
240         return (s);
241 }

243 /*
244  * Callback function for Pfindexec().  We return a match if we can stat the
245  * suggested pathname and confirm its device and inode number match our
246  * previous information about the /proc/<pid>/object/a.out file.
247  */
248 static int
249 stat_exec(const char *path, void *arg)
250 {
251         struct stat64 *stp = arg;
252         struct stat64 st;
```

```
254         return (stat64(path, &st) == 0 && S_ISREG(st.st_mode) &&
255             stp->st_dev == st.st_dev && stp->st_ino == st.st_ino);
256 }

258 /*ARGSUSED*/
259 static char *
260 Pexecname_live(struct ps_prochandle *P, char *buf, size_t buflen, void *data)
261 {
262         char exec_name[PATH_MAX];
263         char cwd[PATH_MAX];
264         char proc_cwd[64];
265         struct stat64 st;
266         int ret;

268         /*
269          * Try to get the path information first.
270          */
271         (void) snprintf(exec_name, sizeof (exec_name),
272             "%s/%d/path/a.out", procfs_path, (int)P->pid);
273         if ((ret = readlink(exec_name, buf, buflen - 1)) > 0) {
274                 buf[ret] = '\0';
275                 (void) Pfindobj(P, buf, buf, buflen);
276                 return (buf);
277         }

279         /*
280          * Stat the executable file so we can compare Pfindexec's
281          * suggestions to the actual device and inode number.
282          */
283         (void) snprintf(exec_name, sizeof (exec_name),
284             "%s/%d/object/a.out", procfs_path, (int)P->pid);

286         if (stat64(exec_name, &st) != 0 || !S_ISREG(st.st_mode))
287                 return (NULL);

289         /*
290          * Attempt to figure out the current working directory of the
291          * target process.  This only works if the target process has
292          * not changed its current directory since it was exec'd.
293          */
294         (void) snprintf(proc_cwd, sizeof (proc_cwd),
295             "%s/%d/path/cwd", procfs_path, (int)P->pid);

297         if ((ret = readlink(proc_cwd, cwd, PATH_MAX - 1)) > 0)
298                 cwd[ret] = '\0';

300         (void) Pfindexec(P, ret > 0 ? cwd : NULL, stat_exec, &st);

302         return (NULL);
303 }

305 #if defined(__i386) || defined(__amd64)
306 /*ARGSUSED*/
307 static int
308 Pldt_live(struct ps_prochandle *P, struct ssd *pldt, int nldt, void *data)
309 {
310         return (proc_get_ldt(P->pid, pldt, nldt));
311 }
312 #endif

314 static const ps_ops_t P_live_ops = {
315         .pop_pread      = Pread_live,
316         .pop_pwrite     = Pwrite_live,
317         .pop_read_maps  = Pread_maps_live,
318         .pop_read_aux   = Pread_aux_live,
```



```
319         .pop_cred       = Pcred_live,
320         .pop_priv       = Ppriv_live,
321         .pop_psinfo     = Ppsinfo_live,
322         .pop_lstatus    = Plstatus_live,
323         .pop_lpsinfo    = Plpsinfo_live,
324         .pop_platform   = Pplatform_live,
325         .pop_uname      = Puname_live,
326         .pop_zonename   = Pzonename_live,
327         .pop_execname   = Pexecname_live,
328 #if defined(__i386) || defined(__amd64)
329         .pop_ldt        = Pldt_live
330 #endif
331 };

333 /*
334  * This is the library's .init handler.
335  */
336 #pragma init(_libproc_init)
337 void
338 _libproc_init(void)
339 {
340         _libproc_debug = getenv("LIBPROC_DEBUG") != NULL;
341         _libproc_no_qsort = getenv("LIBPROC_NO_QSORT") != NULL;
342         _libproc_incore_elf = getenv("LIBPROC_INCORE_ELF") != NULL;

344         (void) sigfillset(&blockable_sigs);
345         (void) sigdelset(&blockable_sigs, SIGKILL);
346         (void) sigdelset(&blockable_sigs, SIGSTOP);
347 }
______unchanged_portion_omitted_

412 /*
413  * Create a new controlled process.
414  * Leave it stopped on successful exit from exec() or execve().
415  * Return an opaque pointer to its process control structure.
416  * Return NULL if process cannot be created (fork()/exec() not successful).
417  */
418 struct ps_prochandle *
419 Pxcreate(const char *file,      /* executable file name */
420         char *const *argv,      /* argument vector */
421         char *const *envp,      /* environment */
422         int *perr,      /* pointer to error return code */
423         char *path,     /* if non-null, holds exec path name on return */
424         size_t len)     /* size of the path buffer */
425 {
426         char execpath[PATH_MAX];
427         char procname[PATH_MAX];
428         struct ps_prochandle *P;
429         pid_t pid;
430         int fd;
431         char *fname;
432         int rc;
433         int lasterrno = 0;

435         if (len == 0)   /* zero length, no path */
436                 path = NULL;
437         if (path != NULL)
438                 *path = '\0';

440         if ((P = malloc(sizeof (struct ps_prochandle))) == NULL) {
441                 *perr = C_STRANGE;
442                 return (NULL);
443         }

445         if ((pid = fork1()) == -1) {
446                 free(P);
```

```
447                 *perr = C_FORK;
448                 return (NULL);
449         }

451         if (pid == 0) {                 /* child process */
452                 id_t id;
453                 extern char **environ;

455                 /*
456                  * If running setuid or setgid, reset credentials to normal.
457                  */
458                 if ((id = getgid()) != getegid())
459                         (void) setgid(id);
460                 if ((id = getuid()) != geteuid())
461                         (void) setuid(id);

463                 Pcreate_callback(P);    /* execute callback (see below) */
464                 (void) pause();         /* wait for PRSABORT from parent */

466                 /*
467                  * This is ugly.  There is no execvep() function that takes a
468                  * path and an environment.  We cheat here by replacing the
469                  * global 'environ' variable right before we call this.
470                  */
471                 if (envp)
472                         environ = (char **)envp;

474                 (void) execvp(file, argv);  /* execute the program */
475                 _exit(127);
476         }

478         /*
479          * Initialize the process structure.
480          */
481         (void) memset(P, 0, sizeof (*P));
482         (void) mutex_init(&P->proc_lock, USYNC_THREAD, NULL);
483         P->flags |= CREATED;
484         P->state = PS_RUN;
485         P->pid = pid;
486         P->asfd = -1;
487         P->ctlfd = -1;
488         P->statfd = -1;
489         P->agentctlfd = -1;
490         P->agentstatfd = -1;
491         Pinit_ops(&P->ops, &P_live_ops);
252         P->ops = &P_live_ops;
492         Pinitsym(P);

494         /*
495          * Open the /proc/pid files.
496          */
497         (void) snprintf(procname, sizeof (procname), "%s/%d/",
498             procfs_path, (int)pid);
499         fname = procname + strlen(procname);
500         (void) set_minfd();

502         /*
503          * Exclusive write open advises others not to interfere.
504          * There is no reason for any of these open()s to fail.
505          */
506         (void) strcpy(fname, "as");
507         if ((fd = open(procname, (O_RDWR|O_EXCL))) < 0 ||
508             (fd = dupfd(fd, 0)) < 0) {
509                 dprintf("Pcreate: failed to open %s: %s\n",
510                     procname, strerror(errno));
511                 rc = C_STRANGE;
```



```
512                 goto bad;
513         }
514         P->asfd = fd;

516         (void) strcpy(fname, "status");
517         if ((fd = open(procname, O_RDONLY)) < 0 ||
518             (fd = dupfd(fd, 0)) < 0) {
519                 dprintf("Pcreate: failed to open %s: %s\n",
520                     procname, strerror(errno));
521                 rc = C_STRANGE;
522                 goto bad;
523         }
524         P->statfd = fd;

526         (void) strcpy(fname, "ctl");
527         if ((fd = open(procname, O_WRONLY)) < 0 ||
528             (fd = dupfd(fd, 0)) < 0) {
529                 dprintf("Pcreate: failed to open %s: %s\n",
530                     procname, strerror(errno));
531                 rc = C_STRANGE;
532                 goto bad;
533         }
534         P->ctlfd = fd;

536         (void) Pstop(P, 0);     /* stop the controlled process */

538         /*
539          * Wait for process to sleep in pause().
540          * If the process has already called pause(), then it should be
541          * stopped (PR_REQUESTED) while asleep in pause and we are done.
542          * Else we set up to catch entry/exit to pause() and set the process
543          * running again, expecting it to stop when it reaches pause().
544          * There is no reason for this to fail other than an interrupt.
545          */
546         (void) Psysentry(P, SYS_pause, 1);
547         (void) Psysexit(P, SYS_pause, 1);
548         for (;;) {
549                 if (P->state == PS_STOP &&
550                     P->status.pr_lwp.pr_syscall == SYS_pause &&
551                     (P->status.pr_lwp.pr_why == PR_REQUESTED ||
552                     P->status.pr_lwp.pr_why == PR_SYSENTRY ||
553                     P->status.pr_lwp.pr_why == PR_SYSEXIT))
554                         break;

556                 if (P->state != PS_STOP ||      /* interrupt or process died */
557                     Psetrun(P, 0, 0) != 0) {    /* can't restart */
558                         if (errno == EINTR || errno == ERESTART)
559                                 rc = C_INTR;
560                         else {
561                                 dprintf("Pcreate: Psetrun failed: %s\n",
562                                     strerror(errno));
563                                 rc = C_STRANGE;
564                         }
565                         goto bad;
566                 }

568                 (void) Pwait(P, 0);
569         }
570         (void) Psysentry(P, SYS_pause, 0);
571         (void) Psysexit(P, SYS_pause, 0);

573         /*
574          * Kick the process off the pause() and catch
575          * it again on entry to exec() or exit().
576          */
577         (void) Psysentry(P, SYS_exit, 1);
```

```
578         (void) Psysentry(P, SYS_execve, 1);
579         if (Psetrun(P, 0, PRSABORT) == -1) {
580                 dprintf("Pcreate: Psetrun failed: %s\n", strerror(errno));
581                 rc = C_STRANGE;
582                 goto bad;
583         }
584         (void) Pwait(P, 0);
585         if (P->state != PS_STOP) {
586                 dprintf("Pcreate: Pwait failed: %s\n", strerror(errno));
587                 rc = C_STRANGE;
588                 goto bad;
589         }

591         /*
592          * Move the process through instances of failed exec()s
593          * to reach the point of stopped on successful exec().
594          */
595         (void) Psysexit(P, SYS_execve, TRUE);

597         while (P->state == PS_STOP &&
598             P->status.pr_lwp.pr_why == PR_SYSENTRY &&
599             P->status.pr_lwp.pr_what == SYS_execve) {
600                 /*
601                  * Fetch the exec path name now, before we complete
602                  * the exec().  We may lose the process and be unable
603                  * to get the information later.
604                  */
605                 (void) Pread_string(P, execpath, sizeof (execpath),
606                     (off_t)P->status.pr_lwp.pr_sysarg[0]);
607                 if (path != NULL)
608                         (void) strncpy(path, execpath, len);
609                 /*
610                  * Set the process running and wait for
611                  * it to stop on exit from the exec().
612                  */
613                 (void) Psetrun(P, 0, 0);
614                 (void) Pwait(P, 0);

616                 if (P->state == PS_LOST &&              /* we lost control */
617                     Preopen(P) != 0) {          /* and we can't get it back */
618                         rc = C_PERM;
619                         goto bad;
620                 }

622                 /*
623                  * If the exec() failed, continue the loop, expecting
624                  * there to be more attempts to exec(), based on PATH.
625                  */
626                 if (P->state == PS_STOP &&
627                     P->status.pr_lwp.pr_why == PR_SYSEXIT &&
628                     P->status.pr_lwp.pr_what == SYS_execve &&
629                     (lasterrno = P->status.pr_lwp.pr_errno) != 0) {
630                         /*
631                          * The exec() failed.  Set the process running and
632                          * wait for it to stop on entry to the next exec().
633                          */
634                         (void) Psetrun(P, 0, 0);
635                         (void) Pwait(P, 0);

637                         continue;
638                 }
639                 break;
640         }

642         if (P->state == PS_STOP &&
643             P->status.pr_lwp.pr_why == PR_SYSEXIT &&
```



```
644             P->status.pr_lwp.pr_what == SYS_execve &&
645             P->status.pr_lwp.pr_errno == 0) {
646                 /*
647                  * The process is stopped on successful exec() or execve().
648                  * Turn off all tracing flags and return success.
649                  */
650                 restore_tracing_flags(P);
651 #ifndef _LP64
652                 /* We must be a 64-bit process to deal with a 64-bit process */
653                 if (P->status.pr_dmodel == PR_MODEL_LP64) {
654                         rc = C_LP64;
655                         goto bad;
656                 }
657 #endif
658                 /*
659                  * Set run-on-last-close so the controlled process
660                  * runs even if we die on a signal.
661                  */
662                 (void) Psetflags(P, PR_RLC);
663                 *perr = 0;
664                 return (P);
665         }

667         rc = lasterrno == ENOENT ? C_NOENT : C_NOEXEC;

669 bad:
670         (void) kill(pid, SIGKILL);
671         if (path != NULL && rc != C_PERM && rc != C_LP64)
672                 *path = '\0';
673         Pfree(P);
674         *perr = rc;
675         return (NULL);
676 }
```

unchanged_portion_omitted_

```
742 /*
743  * Grab an existing process.
744  * Return an opaque pointer to its process control structure.
745  *
746  * pid:         UNIX process ID.
747  * flags:
748  *      PGRAB_RETAIN    Retain tracing flags (default clears all tracing flags).
749  *      PGRAB_FORCE     Grab regardless of whether process is already traced.
750  *      PGRAB_RDONLY    Open the address space file O_RDONLY instead of O_RDWR,
751  *                      and do not open the process control file.
752  *      PGRAB_NOSTOP    Open the process but do not force it to stop.
753  * perr:        pointer to error return code.
754  */
755 struct ps_prochandle *
756 Pgrab(pid_t pid, int flags, int *perr)
757 {
758         struct ps_prochandle *P;
759         int fd, omode;
760         char procname[PATH_MAX];
761         char *fname;
762         int rc = 0;

764         /*
765          * PGRAB_RDONLY means that we do not open the /proc/<pid>/control file,
766          * and so it implies RETAIN and NOSTOP since both require control.
767          */
768         if (flags & PGRAB_RDONLY)
769                 flags |= PGRAB_RETAIN | PGRAB_NOSTOP;

771         if ((P = malloc(sizeof (struct ps_prochandle))) == NULL) {
772                 *perr = G_STRANGE;
```

```
773                 return (NULL);
774         }

776         P->asfd = -1;
777         P->ctlfd = -1;
778         P->statfd = -1;

780 again:  /* Come back here if we lose it in the Window of Vulnerability */
781         if (P->ctlfd >= 0)
782                 (void) close(P->ctlfd);
783         if (P->asfd >= 0)
784                 (void) close(P->asfd);
785         if (P->statfd >= 0)
786                 (void) close(P->statfd);
787         (void) memset(P, 0, sizeof (*P));
788         (void) mutex_init(&P->proc_lock, USYNC_THREAD, NULL);
789         P->ctlfd = -1;
790         P->asfd = -1;
791         P->statfd = -1;
792         P->agentctlfd = -1;
793         P->agentstatfd = -1;
794         Pinit_ops(&P->ops, &P_live_ops);
555         P->ops = &P_live_ops;
795         Pinitsym(P);

797         /*
798          * Open the /proc/pid files
799          */
800         (void) snprintf(procname, sizeof (procname), "%s/%d/",
801             procfs_path, (int)pid);
802         fname = procname + strlen(procname);
803         (void) set_minfd();

805         /*
806          * Request exclusive open to avoid grabbing someone else's
807          * process and to prevent others from interfering afterwards.
808          * If this fails and the 'PGRAB_FORCE' flag is set, attempt to
809          * open non-exclusively.
810          */
811         (void) strcpy(fname, "as");
812         omode = (flags & PGRAB_RDONLY) ? O_RDONLY : O_RDWR;

814         if (((fd = open(procname, omode | O_EXCL)) < 0 &&
815             (fd = ((flags & PGRAB_FORCE)? open(procname, omode) : -1)) < 0) ||
816             (fd = dupfd(fd, 0)) < 0) {
817                 switch (errno) {
818                 case ENOENT:
819                         rc = G_NOPROC;
820                         break;
821                 case EACCES:
822                 case EPERM:
823                         rc = G_PERM;
824                         break;
825                 case EMFILE:
826                         rc = G_NOFD;
827                         break;
828                 case EBUSY:
829                         if (!(flags & PGRAB_FORCE) || geteuid() != 0) {
830                                 rc = G_BUSY;
831                                 break;
832                         }
833                         /* FALLTHROUGH */
834                 default:
835                         dprintf("Pgrab: failed to open %s: %s\n",
836                             procname, strerror(errno));
837                         rc = G_STRANGE;
```



```
838                         break;
839                 }
840                 goto err;
841         }
842         P->asfd = fd;

844         (void) strcpy(fname, "status");
845         if ((fd = open(procname, O_RDONLY)) < 0 ||
846             (fd = dupfd(fd, 0)) < 0) {
847                 switch (errno) {
848                 case ENOENT:
849                         rc = G_NOPROC;
850                         break;
851                 case EMFILE:
852                         rc = G_NOFD;
853                         break;
854                 default:
855                         dprintf("Pgrab: failed to open %s: %s\n",
856                             procname, strerror(errno));
857                         rc = G_STRANGE;
858                         break;
859                 }
860                 goto err;
861         }
862         P->statfd = fd;

864         if (!(flags & PGRAB_RDONLY)) {
865                 (void) strcpy(fname, "ctl");
866                 if ((fd = open(procname, O_WRONLY)) < 0 ||
867                     (fd = dupfd(fd, 0)) < 0) {
868                         switch (errno) {
869                         case ENOENT:
870                                 rc = G_NOPROC;
871                                 break;
872                         case EMFILE:
873                                 rc = G_NOFD;
874                                 break;
875                         default:
876                                 dprintf("Pgrab: failed to open %s: %s\n",
877                                     procname, strerror(errno));
878                                 rc = G_STRANGE;
879                                 break;
880                         }
881                         goto err;
882                 }
883                 P->ctlfd = fd;
884         }

886         P->state = PS_RUN;
887         P->pid = pid;

889         /*
890          * We are now in the Window of Vulnerability (WoV).  The process may
891          * exec() a setuid/setgid or unreadable object file between the open()
892          * and the PCSTOP.  We will get EAGAIN in this case and must start over.
893          * As Pstopstatus will trigger the first read() from a /proc file,
894          * we also need to handle EOVERFLOW here when 32-bit as an indicator
895          * that this process is 64-bit.  Finally, if the process has become
896          * a zombie (PS_UNDEAD) while we were trying to grab it, just remain
897          * silent about this and pretend there was no process.
898          */
899         if (Pstopstatus(P, PCNULL, 0) != 0) {
900 #ifndef _LP64
901                 if (errno == EOVERFLOW) {
902                         rc = G_LP64;
903                         goto err;
```

```
904                 }
905 #endif
906                 if (P->state == PS_LOST) {      /* WoV */
907                         (void) mutex_destroy(&P->proc_lock);
908                         goto again;
909                 }

911                 if (P->state == PS_UNDEAD)
912                         rc = G_NOPROC;
913                 else
914                         rc = G_STRANGE;

916                 goto err;
917         }

919         /*
920          * If the process is a system process, we can't control it even as root
921          */
922         if (P->status.pr_flags & PR_ISSYS) {
923                 rc = G_SYS;
924                 goto err;
925         }
926 #ifndef _LP64
927         /*
928          * We must be a 64-bit process to deal with a 64-bit process
929          */
930         if (P->status.pr_dmodel == PR_MODEL_LP64) {
931                 rc = G_LP64;
932                 goto err;
933         }
934 #endif

936         /*
937          * Remember the status for use by Prelease().
938          */
939         P->orig_status = P->status;     /* structure copy */

941         /*
942          * Before stopping the process, make sure we are not grabbing ourselves.
943          * If we are, make sure we are doing it PGRAB_RDONLY.
944          */
945         if (pid == getpid()) {
946                 /*
947                  * Verify that the process is really ourself:
948                  * Set a magic number, read it through the
949                  * /proc file and see if the results match.
950                  */
951                 uint32_t magic1 = 0;
952                 uint32_t magic2 = 2;

954                 errno = 0;

956                 if (Pread(P, &magic2, sizeof (magic2), (uintptr_t)&magic1)
957                     == sizeof (magic2) &&
958                     magic2 == 0 &&
959                     (magic1 = 0xfeedbeef) &&
960                     Pread(P, &magic2, sizeof (magic2), (uintptr_t)&magic1)
961                     == sizeof (magic2) &&
962                     magic2 == 0xfeedbeef &&
963                     !(flags & PGRAB_RDONLY)) {
964                         rc = G_SELF;
965                         goto err;
966                 }
967         }

969         /*
```



```
970          * If the process is already stopped or has been directed
971          * to stop via /proc, do not set run-on-last-close.
972          */
973         if (!(P->status.pr_lwp.pr_flags & (PR_ISTOP|PR_DSTOP)) &&
974             !(flags & PGRAB_RDONLY)) {
975                 /*
976                  * Mark the process run-on-last-close so
977                  * it runs even if we die from SIGKILL.
978                  */
979                 if (Psetflags(P, PR_RLC) != 0) {
980                         if (errno == EAGAIN) {  /* WoV */
981                                 (void) mutex_destroy(&P->proc_lock);
982                                 goto again;
983                         }
984                         if (errno == ENOENT)    /* No complaint about zombies */
985                                 rc = G_ZOMB;
986                         else {
987                                 dprintf("Pgrab: failed to set RLC\n");
988                                 rc = G_STRANGE;
989                         }
990                         goto err;
991                 }
992         }

994         /*
995          * If a stop directive is pending and the process has not yet stopped,
996          * then synchronously wait for the stop directive to take effect.
997          * Limit the time spent waiting for the process to stop by iterating
998          * at most 10 times. The time-out of 20 ms corresponds to the time
999          * between sending the stop directive and the process actually stopped
1000         * as measured by DTrace on a slow, busy system. If the process doesn't
1001         * stop voluntarily, clear the PR_DSTOP flag so that the code below
1002         * forces the process to stop.
1003         */
1004        if (!(flags & PGRAB_RDONLY)) {
1005                int niter = 0;
1006                while ((P->status.pr_lwp.pr_flags & (PR_STOPPED|PR_DSTOP)) ==
1007                    PR_DSTOP && niter < 10 &&
1008                    Pstopstatus(P, PCTWSTOP, 20) != 0) {
1009                        niter++;
1010                        if (flags & PGRAB_NOSTOP)
1011                                break;
1012                }
1013                if (niter == 10 && !(flags & PGRAB_NOSTOP)) {
1014                        /* Try it harder down below */
1015                        P->status.pr_lwp.pr_flags &= ~PR_DSTOP;
1016                }
1017        }

1019        /*
1020         * If the process is not already stopped or directed to stop
1021         * and PGRAB_NOSTOP was not specified, stop the process now.
1022         */
1023        if (!(P->status.pr_lwp.pr_flags & (PR_ISTOP|PR_DSTOP)) &&
1024            !(flags & PGRAB_NOSTOP)) {
1025                /*
1026                 * Stop the process, get its status and signal/syscall masks.
1027                 */
1028                if (((P->status.pr_lwp.pr_flags & PR_STOPPED) &&
1029                    Pstopstatus(P, PCDSTOP, 0) != 0) ||
1030                    Pstopstatus(P, PCSTOP, 2000) != 0) {
1031 #ifndef _LP64
1032                        if (errno == EOVERFLOW) {
1033                                rc = G_LP64;
1034                                goto err;
1035                        }
```

```
#endif
                        if (P->state == PS_LOST) {      /* WoV */
                                (void) mutex_destroy(&P->proc_lock);
                                goto again;
                        }
                        if ((errno != EINTR && errno != ERESTART) ||
                            (P->state != PS_STOP &&
                            !(P->status.pr_flags & PR_DSTOP))) {
                                if (P->state != PS_RUN && errno != ENOENT) {
                                        dprintf("Pgrab: failed to PCSTOP\n");
                                        rc = G_STRANGE;
                                } else {
                                        rc = G_ZOMB;
                                }
                                goto err;
                        }
                }

                /*
                 * Process should now either be stopped via /proc or there
                 * should be an outstanding stop directive.
                 */
                if (!(P->status.pr_flags & (PR_ISTOP|PR_DSTOP))) {
                        dprintf("Pgrab: process is not stopped\n");
                        rc = G_STRANGE;
                        goto err;
                }
#ifndef _LP64
                /*
                 * Test this again now because the 32-bit victim process may
                 * have exec'd a 64-bit process in the meantime.
                 */
                if (P->status.pr_dmodel == PR_MODEL_LP64) {
                        rc = G_LP64;
                        goto err;
                }
#endif
        }

        /*
         * Cancel all tracing flags unless the PGRAB_RETAIN flag is set.
         */
        if (!(flags & PGRAB_RETAIN)) {
                (void) Psysentry(P, 0, FALSE);
                (void) Psysexit(P, 0, FALSE);
                (void) Psignal(P, 0, FALSE);
                (void) Pfault(P, 0, FALSE);
                Psync(P);
        }

        *perr = 0;
        return (P);

err:
        Pfree(P);
        *perr = rc;
        return (NULL);
}
```

unchanged_portion_omitted_

```
/*
 * Free a process control structure.
 * Close the file descriptors but don't do the Prelease logic.
 */
void
Pfree(struct ps_prochandle *P)
```



```
{
        uint_t i;

        if (P->core != NULL) {
                extern void __priv_free_info(void *);
                lwp_info_t *nlwp, *lwp = list_next(&P->core->core_lwp_head);

                for (i = 0; i < P->core->core_nlwp; i++, lwp = nlwp) {
                        nlwp = list_next(lwp);
#ifdef __sparc
                        if (lwp->lwp_gwins != NULL)
                                free(lwp->lwp_gwins);
                        if (lwp->lwp_xregs != NULL)
                                free(lwp->lwp_xregs);
                        if (lwp->lwp_asrs != NULL)
                                free(lwp->lwp_asrs);
#endif
                        free(lwp);
                }

                if (P->core->core_platform != NULL)
                        free(P->core->core_platform);
                if (P->core->core_uts != NULL)
                        free(P->core->core_uts);
                if (P->core->core_cred != NULL)
                        free(P->core->core_cred);
                if (P->core->core_priv != NULL)
                        free(P->core->core_priv);
                if (P->core->core_privinfo != NULL)
                        __priv_free_info(P->core->core_privinfo);
                if (P->core->core_ppii != NULL)
                        free(P->core->core_ppii);
                if (P->core->core_zonename != NULL)
                        free(P->core->core_zonename);
#if defined(__i386) || defined(__amd64)
                if (P->core->core_ldt != NULL)
                        free(P->core->core_ldt);
#endif

                free(P->core);
        }

        if (P->ucaddrs != NULL) {
                free(P->ucaddrs);
                P->ucaddrs = NULL;
                P->ucnelems = 0;
        }

        (void) mutex_lock(&P->proc_lock);
        if (P->hashtab != NULL) {
                struct ps_lwphandle *L;
                for (i = 0; i < HASHSIZE; i++) {
                        while ((L = P->hashtab[i]) != NULL)
                                Lfree_internal(P, L);
                }
                free(P->hashtab);
        }

        while (P->num_fd > 0) {
                fd_info_t *fip = list_next(&P->fd_head);
                list_unlink(fip);
                free(fip);
                P->num_fd--;
        }
        (void) mutex_unlock(&P->proc_lock);
        (void) mutex_destroy(&P->proc_lock);
```

```
1200         if (P->agentctlfd >= 0)
1201                 (void) close(P->agentctlfd);
1202         if (P->agentstatfd >= 0)
1203                 (void) close(P->agentstatfd);
1204         if (P->ctlfd >= 0)
1205                 (void) close(P->ctlfd);
1206         if (P->asfd >= 0)
1207                 (void) close(P->asfd);
1208         if (P->statfd >= 0)
1209                 (void) close(P->statfd);
1210         Preset_maps(P);
1211         P->ops.pop_fini(P, P->data);

1213         /* clear out the structure as a precaution against reuse */
1214         (void) memset(P, 0, sizeof (*P));
1215         P->ctlfd = -1;
1216         P->asfd = -1;
1217         P->statfd = -1;
1218         P->agentctlfd = -1;
1219         P->agentstatfd = -1;

1221         free(P);
1222 }
_____unchanged_portion_omitted_

1255 /*
1256  * Return a pointer to the process psinfo structure.
1257  * Clients should not hold on to this pointer indefinitely.
1258  * It will become invalid on Prelease().
1259  */
1260 const psinfo_t *
1261 Ppsinfo(struct ps_prochandle *P)
1262 {
1263         return (P->ops.pop_psinfo(P, &P->psinfo, P->data));
1062         if (P->state == PS_IDLE) {
1063                 errno = ENODATA;
1064                 return (NULL);
1065         }

1067         if (P->state != PS_DEAD && proc_get_psinfo(P->pid, &P->psinfo) == -1)
1068                 return (NULL);

1070         return (&P->psinfo);
1264 }
_____unchanged_portion_omitted_

1277 static void
1278 Pread_status(struct ps_prochandle *P)
1279 {
1280         P->ops.pop_status(P, &P->status, P->data);
1281 }

1283 /*
1284  * Fill in a pointer to a process credentials structure.  The ngroups parameter
1285  * is the number of supplementary group entries allocated in the caller's cred
1286  * structure.  It should equal zero or one unless extra space has been
1287  * allocated for the group list by the caller.
1288  */
1289 int
1290 Pcred(struct ps_prochandle *P, prcred_t *pcrp, int ngroups)
1291 {
1292         return (P->ops.pop_cred(P, pcrp, ngroups, P->data));
1293 }
1093         if (P->state == PS_IDLE) {
1094                 errno = ENODATA;
```



```
1095                 return (-1);
1096         }

1295 static prheader_t *
1296 Plstatus(struct ps_prochandle *P)
1297 {
1298         return (P->ops.pop_lstatus(P, P->data));
1299 }
1098         if (P->state != PS_DEAD)
1099                 return (proc_get_cred(P->pid, pcrp, ngroups));

1301 static prheader_t *
1302 Plpsinfo(struct ps_prochandle *P)
1303 {
1304         return (P->ops.pop_lpsinfo(P, P->data));
1305 }
1101         if (P->core->core_cred != NULL) {
1102                 /*
1103                  * Avoid returning more supplementary group data than the
1104                  * caller has allocated in their buffer.  We expect them to
1105                  * check pr_ngroups afterward and potentially call us again.
1106                  */
1107                 ngroups = MIN(ngroups, P->core->core_cred->pr_ngroups);

1109                 (void) memcpy(pcrp, P->core->core_cred,
1110                     sizeof (prcred_t) + (ngroups - 1) * sizeof (gid_t));

1112                 return (0);
1113         }

1115         errno = ENODATA;
1116         return (-1);
1117 }

1308 #if defined(__i386) || defined(__amd64)
1309 /*
1310  * Fill in a pointer to a process LDT structure.
1311  * The caller provides a buffer of size 'nldt * sizeof (struct ssd)';
1312  * If pldt == NULL or nldt == 0, we return the number of existing LDT entries.
1313  * Otherwise we return the actual number of LDT entries fetched (<= nldt).
1314  */
1315 int
1316 Pldt(struct ps_prochandle *P, struct ssd *pldt, int nldt)
1317 {
1318         return (P->ops.pop_ldt(P, pldt, nldt, P->data));
1129         if (P->state == PS_IDLE) {
1130                 errno = ENODATA;
1131                 return (-1);
1132         }

1134         if (P->state != PS_DEAD)
1135                 return (proc_get_ldt(P->pid, pldt, nldt));

1137         if (pldt == NULL || nldt == 0)
1138                 return (P->core->core_nldt);

1140         if (P->core->core_ldt != NULL) {
1141                 nldt = MIN(nldt, P->core->core_nldt);

1143                 (void) memcpy(pldt, P->core->core_ldt,
1144                     nldt * sizeof (struct ssd));

1146                 return (nldt);
1147         }

1149         errno = ENODATA;
```

```
1150         return (-1);
1320 }
1321 #endif  /* __i386 */

1323 /*
1324  * Return a malloced process privilege structure in *pprv.
1155  * Fill in a pointer to a process privilege structure.
1325  */
1326 int
1327 Ppriv(struct ps_prochandle *P, prpriv_t **pprv)
1157 ssize_t
1158 Ppriv(struct ps_prochandle *P, prpriv_t *pprv, size_t size)
1328 {
1329         return (P->ops.pop_priv(P, pprv, P->data));
1160         if (P->state != PS_DEAD) {
1161                 prpriv_t *pp = proc_get_priv(P->pid);
1162                 if (pp != NULL) {
1163                         size = MIN(size, PRIV_PRPRIV_SIZE(pp));
1164                         (void) memcpy(pprv, pp, size);
1165                         free(pp);
1166                         return (size);
1167                 }
1168                 return (-1);
1169         }

1171         if (P->core->core_priv != NULL) {
1172                 size = MIN(P->core->core_priv_size, size);
1173                 (void) memcpy(pprv, P->core->core_priv, size);
1174                 return (size);
1175         }
1176         errno = ENODATA;
1177         return (-1);
1330 }
______unchanged_portion_omitted_

1366 void *
1367 Pprivinfo(struct ps_prochandle *P)
1368 {
1369         core_info_t *core = P->data;

1371         /* Use default from libc */
1372         if (P->state != PS_DEAD)
1373                 return (NULL);

1375         return (core->core_privinfo);
1221         return (P->core->core_privinfo);
1376 }
______unchanged_portion_omitted_

2129 ssize_t
2130 Pread(struct ps_prochandle *P,
2131         void *buf,              /* caller's buffer */
2132         size_t nbyte,           /* number of bytes to read */
2133         uintptr_t address)      /* address in process */
2134 {
2135         return (P->ops.pop_pread(P, buf, nbyte, address, P->data));
1981         return (P->ops->p_pread(P, buf, nbyte, address));
2136 }

2138 ssize_t
2139 Pread_string(struct ps_prochandle *P,
2140         char *buf,              /* caller's buffer */
2141         size_t size,            /* upper limit on bytes to read */
2142         uintptr_t addr)         /* address in process */
2143 {
2144         enum { STRSZ = 40 };
```



```
2145         char string[STRSZ + 1];
2146         ssize_t leng = 0;
2147         int nbyte;

2149         if (size < 2) {
2150                 errno = EINVAL;
2151                 return (-1);
2152         }

2154         size--;                 /* ensure trailing null fits in buffer */

2156         *buf = '\0';
2157         string[STRSZ] = '\0';

2159         for (nbyte = STRSZ; nbyte == STRSZ && leng < size; addr += STRSZ) {
2160                 if ((nbyte = P->ops.pop_pread(P, string, STRSZ, addr,
2161                     P->data)) <= 0) {
2006                 if ((nbyte = P->ops->p_pread(P, string, STRSZ, addr)) <= 0) {
2162                         buf[leng] = '\0';
2163                         return (leng ? leng : -1);
2164                 }
2165                 if ((nbyte = strlen(string)) > 0) {
2166                         if (leng + nbyte > size)
2167                                 nbyte = size - leng;
2168                         (void) strncpy(buf + leng, string, nbyte);
2169                         leng += nbyte;
2170                 }
2171         }
2172         buf[leng] = '\0';
2173         return (leng);
2174 }

2176 ssize_t
2177 Pwrite(struct ps_prochandle *P,
2178         const void *buf,        /* caller's buffer */
2179         size_t nbyte,           /* number of bytes to write */
2180         uintptr_t address)      /* address in process */
2181 {
2182         return (P->ops.pop_pwrite(P, buf, nbyte, address, P->data));
2027         return (P->ops->p_pwrite(P, buf, nbyte, address));
2183 }
______unchanged_portion_omitted_

2886 /*
2887  * LWP iteration interface.
2888  */
2889 int
2890 Plwp_iter(struct ps_prochandle *P, proc_lwp_f *func, void *cd)
2891 {
2892         prheader_t *Lhp;
2893         lwpstatus_t *Lsp;
2894         long nlwp;
2895         int rv;

2897         switch (P->state) {
2898         case PS_RUN:
2899                 (void) Pstopstatus(P, PCNULL, 0);
2900                 break;

2902         case PS_STOP:
2903                 Psync(P);
2904                 break;

2906         case PS_IDLE:
2907                 errno = ENODATA;
2908                 return (-1);
```

```
2909         }

2911         /*
2912          * For either live processes or cores, the single LWP case is easy:
2913          * the pstatus_t contains the lwpstatus_t for the only LWP.
2914          */
2915         if (P->status.pr_nlwp <= 1)
2916                 return (func(cd, &P->status.pr_lwp));

2918         /*
2919          * For the core file multi-LWP case, we just iterate through the
2920          * list of LWP structs we read in from the core file.
2921          */
2922         if (P->state == PS_DEAD) {
2923                 core_info_t *core = P->data;
2924                 lwp_info_t *lwp = list_prev(&core->core_lwp_head);
2768                 lwp_info_t *lwp = list_prev(&P->core->core_lwp_head);
2925                 uint_t i;

2927                 for (i = 0; i < core->core_nlwp; i++, lwp = list_prev(lwp)) {
2771                 for (i = 0; i < P->core->core_nlwp; i++, lwp = list_prev(lwp)) {
2928                         if (lwp->lwp_psinfo.pr_sname != 'Z' &&
2929                             (rv = func(cd, &lwp->lwp_status)) != 0)
2930                                 break;
2931                 }

2933                 return (rv);
2934         }

2936         /*
2937          * For the live process multi-LWP case, we have to work a little
2938          * harder: the /proc/pid/lstatus file has the array of LWP structs.
2939          */
2940         if ((Lhp = Plstatus(P)) == NULL)
2784         if ((Lhp = read_lfile(P, "lstatus")) == NULL)
2941                 return (-1);

2943         for (nlwp = Lhp->pr_nent, Lsp = (lwpstatus_t *)(uintptr_t)(Lhp + 1);
2944             nlwp > 0;
2945             nlwp--, Lsp = (lwpstatus_t *)((uintptr_t)Lsp + Lhp->pr_entsize)) {
2946                 if ((rv = func(cd, Lsp)) != 0)
2947                         break;
2948         }

2950         free(Lhp);
2951         return (rv);
2952 }

2954 /*
2955  * Extended LWP iteration interface.
2956  * Iterate over all LWPs, active and zombie.
2957  */
2958 int
2959 Plwp_iter_all(struct ps_prochandle *P, proc_lwp_all_f *func, void *cd)
2960 {
2961         prheader_t *Lhp = NULL;
2962         lwpstatus_t *Lsp;
2963         lwpstatus_t *sp;
2964         prheader_t *Lphp = NULL;
2965         lwpsinfo_t *Lpsp;
2966         long nstat;
2967         long ninfo;
2968         int rv;

2970 retry:
2971         if (Lhp != NULL)
```



```
2972                 free(Lhp);
2973         if (Lphp != NULL)
2974                 free(Lphp);
2975         if (P->state == PS_RUN)
2976                 (void) Pstopstatus(P, PCNULL, 0);
2977         (void) Ppsinfo(P);

2979         if (P->state == PS_STOP)
2980                 Psync(P);

2982         /*
2983          * For either live processes or cores, the single LWP case is easy:
2984          * the pstatus_t contains the lwpstatus_t for the only LWP and
2985          * the psinfo_t contains the lwpsinfo_t for the only LWP.
2986          */
2987         if (P->status.pr_nlwp + P->status.pr_nzomb <= 1)
2988                 return (func(cd, &P->status.pr_lwp, &P->psinfo.pr_lwp));

2990         /*
2991          * For the core file multi-LWP case, we just iterate through the
2992          * list of LWP structs we read in from the core file.
2993          */
2994         if (P->state == PS_DEAD) {
2995                 core_info_t *core = P->data;
2996                 lwp_info_t *lwp = list_prev(&core->core_lwp_head);
2839                 lwp_info_t *lwp = list_prev(&P->core->core_lwp_head);
2997                 uint_t i;

2999                 for (i = 0; i < core->core_nlwp; i++, lwp = list_prev(lwp)) {
2842                 for (i = 0; i < P->core->core_nlwp; i++, lwp = list_prev(lwp)) {
3000                         sp = (lwp->lwp_psinfo.pr_sname == 'Z')? NULL :
3001                             &lwp->lwp_status;
3002                         if ((rv = func(cd, sp, &lwp->lwp_psinfo)) != 0)
3003                                 break;
3004                 }

3006                 return (rv);
3007         }

3009         /*
3010          * For all other cases retrieve the array of lwpstatus_t's and
3011          * lwpsinfo_t's.
2853          * For the live process multi-LWP case, we have to work a little
2854          * harder: the /proc/pid/lstatus file has the array of lwpstatus_t's
2855          * and the /proc/pid/lpsinfo file has the array of lwpsinfo_t's.
3012          */
3013         if ((Lhp = Plstatus(P)) == NULL)
2857         if ((Lhp = read_lfile(P, "lstatus")) == NULL)
3014                 return (-1);
3015         if ((Lphp = Plpsinfo(P)) == NULL) {
2859         if ((Lphp = read_lfile(P, "lpsinfo")) == NULL) {
3016                 free(Lhp);
3017                 return (-1);
3018         }

3020         /*
3021          * If we are looking at a running process, or one we do not control,
3022          * the active and zombie lwps in the process may have changed since
3023          * we read the process status structure.  If so, just start over.
3024          */
3025         if (Lhp->pr_nent != P->status.pr_nlwp ||
3026             Lphp->pr_nent != P->status.pr_nlwp + P->status.pr_nzomb)
3027                 goto retry;

3029         /*
3030          * To be perfectly safe, prescan the two arrays, checking consistency.
```

```
3031          * We rely on /proc giving us lwpstatus_t's and lwpsinfo_t's in the
3032          * same order (the lwp directory order) in their respective files.
3033          * We also rely on there being (possibly) more lwpsinfo_t's than
3034          * lwpstatus_t's (the extra lwpsinfo_t's are for zombie lwps).
3035          */
3036         Lsp = (lwpstatus_t *)(uintptr_t)(Lhp + 1);
3037         Lpsp = (lwpsinfo_t *)(uintptr_t)(Lphp + 1);
3038         nstat = Lhp->pr_nent;
3039         for (ninfo = Lphp->pr_nent; ninfo != 0; ninfo--) {
3040                 if (Lpsp->pr_sname != 'Z') {
3041                         /*
3042                          * Not a zombie lwp; check for matching lwpids.
3043                          */
3044                         if (nstat == 0 || Lsp->pr_lwpid != Lpsp->pr_lwpid)
3045                                 goto retry;
3046                         Lsp = (lwpstatus_t *)((uintptr_t)Lsp + Lhp->pr_entsize);
3047                         nstat--;
3048                 }
3049                 Lpsp = (lwpsinfo_t *)((uintptr_t)Lpsp + Lphp->pr_entsize);
3050         }
3051         if (nstat != 0)
3052                 goto retry;

3054         /*
3055          * Rescan, this time for real.
3056          */
3057         Lsp = (lwpstatus_t *)(uintptr_t)(Lhp + 1);
3058         Lpsp = (lwpsinfo_t *)(uintptr_t)(Lphp + 1);
3059         for (ninfo = Lphp->pr_nent; ninfo != 0; ninfo--) {
3060                 if (Lpsp->pr_sname != 'Z') {
3061                         sp = Lsp;
3062                         Lsp = (lwpstatus_t *)((uintptr_t)Lsp + Lhp->pr_entsize);
3063                 } else {
3064                         sp = NULL;
3065                 }
3066                 if ((rv = func(cd, sp, Lpsp)) != 0)
3067                         break;
3068                 Lpsp = (lwpsinfo_t *)((uintptr_t)Lpsp + Lphp->pr_entsize);
3069         }

3071         free(Lhp);
3072         free(Lphp);
3073         return (rv);
3074 }

3076 core_content_t
3077 Pcontent(struct ps_prochandle *P)
3078 {
3079         core_info_t *core = P->data;

3081         if (P->state == PS_DEAD)
3082                 return (core->core_content);
2924                 return (P->core->core_content);
3083         if (P->state == PS_IDLE)
3084                 return (CC_CONTENT_TEXT | CC_CONTENT_DATA | CC_CONTENT_CTF);

3086         return (CC_CONTENT_ALL);
3087 }
```

*_____unchanged_portion_omitted_*

```
3867 /*
3868  * Sort the current set of mappings.  Should be called during target
3869  * initialization after all calls to Padd_mapping() have been made.
3870  */
3871 void
3872 Psort_mappings(struct ps_prochandle *P)
```



```
3873 {
3874         int i;
3875         map_info_t *mp;

3877         qsort(P->mappings, P->map_count, sizeof (map_info_t), map_sort);

3879         /*
3880          * Update all the file_map pointers to refer to the new locations.
3881          */
3882         for (i = 0; i < P->map_count; i++) {
3883                 mp = &P->mappings[i];
3884                 if (mp->map_relocate)
3885                         mp->map_file->file_map = mp;
3886                 mp->map_relocate = 0;
3887         }
3888 }

3890 struct ps_prochandle *
3891 Pgrab_ops(pid_t pid, void *data, const ps_ops_t *ops, int flags)
3892 {
3893         struct ps_prochandle *P;

3895         if ((P = calloc(1, sizeof (*P))) == NULL) {
3896                 return (NULL);
3897         }

3899         Pinit_ops(&P->ops, ops);
3900         (void) mutex_init(&P->proc_lock, USYNC_THREAD, NULL);
3901         P->pid = pid;
3902         P->state = PS_STOP;
3903         P->asfd = -1;
3904         P->ctlfd = -1;
3905         P->statfd = -1;
3906         P->agentctlfd = -1;
3907         P->agentstatfd = -1;
3908         Pinitsym(P);
3909         P->data = data;
3910         Pread_status(P);

3912         if (flags & PGRAB_INCORE) {
3913                 P->flags |= INCORE;
3914         }

3916         return (P);
3917 }
```

*_____unchanged_portion_omitted_*

```
**********************************************************
   12762 Wed Aug 21 14:46:24 2013
new/usr/src/lib/libproc/common/Pcontrol.h
3946 ::gcore
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Matthew Ahrens <mahrens@delphix.com>
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2008 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */
/*
 * Copyright 2012 DEY Storage Systems, Inc.  All rights reserved.
 * Copyright (c) 2013, Joyent, Inc. All rights reserved.
 * Copyright (c) 2013 by Delphix. All rights reserved.
 */

#ifndef _PCONTROL_H
#define _PCONTROL_H

/*
 * Implemention-specific include file for libproc process management.
 * This is not to be seen by the clients of libproc.
 */

#include <stdio.h>
#include <gelf.h>
#include <synch.h>
#include <procfs.h>
#include <rtld_db.h>
#include <libproc.h>
#include <libctf.h>
#include <limits.h>
#include <libproc.h>

#ifdef  __cplusplus
extern "C" {
#endif

#include "Putil.h"

/*
 * Definitions of the process control structures, internal to libproc.
 * These may change without affecting clients of libproc.
 */
```



```
/*
 * sym_tbl_t contains a primary and an (optional) auxiliary symbol table, which
 * we wish to treat as a single logical symbol table. In this logical table,
 * the data from the auxiliary table preceeds that from the primary. Symbol
 * indices start at [0], which is the first item in the auxiliary table
 * if there is one. The sole purpose for this is so that we can treat the
 * combination of .SUNW_ldynsym and .dynsym sections as a logically single
 * entity without having to violate the public interface to libelf.
 *
 * Both tables must share the same string table section.
 *
 * The symtab_getsym() function serves as a gelf_getsym() replacement
 * that is aware of the two tables and makes them look like a single table
 * to the caller.
 *
 */
typedef struct sym_tbl {        /* symbol table */
        Elf_Data *sym_data_pri; /* primary table */
        Elf_Data *sym_data_aux; /* auxiliary table */
        size_t  sym_symn_aux;   /* number of entries in auxiliary table */
        size_t  sym_symn;       /* total number of entries in both tables */
        char    *sym_strs;      /* ptr to strings */
        size_t  sym_strsz;      /* size of string table */
        GElf_Shdr sym_hdr_pri;  /* primary symbol table section header */
        GElf_Shdr sym_hdr_aux;  /* auxiliary symbol table section header */
        GElf_Shdr sym_strhdr;   /* string table section header */
        Elf     *sym_elf;       /* faked-up ELF handle from core file */
        void    *sym_elfmem;    /* data for faked-up ELF handle */
        uint_t  *sym_byname;    /* symbols sorted by name */
        uint_t  *sym_byaddr;    /* symbols sorted by addr */
        size_t  sym_count;      /* number of symbols in each sorted list */
} sym_tbl_t;
```

_____*unchanged_portion_omitted_*

```
typedef struct ps_rwops {       /* ops vector for Pread() and Pwrite() */
        ssize_t (*p_pread)(struct ps_prochandle *,
            void *, size_t, uintptr_t);
        ssize_t (*p_pwrite)(struct ps_prochandle *,
            const void *, size_t, uintptr_t);
} ps_rwops_t;
```

```
#define HASHSIZE                1024    /* hash table size, power of 2 */

struct ps_prochandle {
        struct ps_lwphandle **hashtab;  /* hash table for LWPs (Lgrab()) */
        mutex_t proc_lock;      /* protects hash table; serializes Lgrab() */
        pstatus_t orig_status;  /* remembered status on Pgrab() */
        pstatus_t status;       /* status when stopped */
        psinfo_t psinfo;        /* psinfo_t from last Ppsinfo() request */
        uintptr_t sysaddr;      /* address of most recent syscall instruction */
        pid_t   pid;            /* process-ID */
        int     state;          /* state of the process, see "libproc.h" */
        uint_t  flags;          /* see defines below */
        uint_t  agentcnt;       /* Pcreate_agent()/Pdestroy_agent() ref count */
        int     asfd;           /* /proc/<pid>/as filedescriptor */
        int     ctlfd;          /* /proc/<pid>/ctl filedescriptor */
        int     statfd;         /* /proc/<pid>/status filedescriptor */
        int     agentctlfd;     /* /proc/<pid>/lwp/agent/ctl */
        int     agentstatfd;    /* /proc/<pid>/lwp/agent/status */
        int     info_valid;     /* if zero, map and file info need updating */
        map_info_t *mappings;   /* cached process mappings */
        size_t  map_count;      /* number of mappings */
        size_t  map_alloc;      /* number of mappings allocated */
        uint_t  num_files;      /* number of file elements in file_info */
        plist_t file_head;      /* head of mapped files w/ symbol table info */
        char    *execname;      /* name of the executable file */
```

```
220         auxv_t  *auxv;          /* the process's aux vector */
221         int     nauxv;          /* number of aux vector entries */
222         rd_agent_t *rap;        /* cookie for rtld_db */
223         map_info_t *map_exec;   /* the mapping for the executable file */
224         map_info_t *map_ldso;   /* the mapping for ld.so.1 */
225         ps_ops_t ops;           /* ops-vector */
230         const ps_rwops_t *ops;  /* pointer to ops-vector for read and write */
231         core_info_t *core;      /* information specific to core (if PS_DEAD) */
226         uintptr_t *ucaddrs;     /* ucontext-list addresses */
227         uint_t  ucnelems;       /* number of elements in the ucaddrs list */
228         char    *zoneroot;      /* cached path to zone root */
229         plist_t fd_head;        /* head of file desc info list */
230         int     num_fd;         /* number of file descs in list */
231         uintptr_t map_missing;  /* first missing mapping in core due to sig */
232         siginfo_t killinfo;     /* signal that interrupted core dump */
233         psinfo_t spymaster;     /* agent LWP's spymaster, if any */
234         void *data;             /* private data */
235 };

237 /* flags */
238 #define CREATED         0x01    /* process was created by Pcreate() */
239 #define SETSIG          0x02    /* set signal trace mask before continuing */
240 #define SETFAULT        0x04    /* set fault trace mask before continuing */
241 #define SETENTRY        0x08    /* set sysentry trace mask before continuing */
242 #define SETEXIT         0x10    /* set sysexit trace mask before continuing */
243 #define SETHOLD         0x20    /* set signal hold mask before continuing */
244 #define SETREGS         0x40    /* set registers before continuing */
245 #define INCORE          0x80    /* use in-core data to build symbol tables */

247 struct ps_lwphandle {
248         struct ps_prochandle *lwp_proc; /* process to which this lwp belongs */
249         struct ps_lwphandle *lwp_hash;  /* hash table linked list */
250         lwpstatus_t     lwp_status;     /* status when stopped */
251         lwpsinfo_t      lwp_psinfo;     /* lwpsinfo_t from last Lpsinfo() */
252         lwpid_t         lwp_id;         /* lwp identifier */
253         int             lwp_state;      /* state of the lwp, see "libproc.h" */
254         uint_t          lwp_flags;      /* SETHOLD and/or SETREGS */
255         int             lwp_ctlfd;      /* /proc/<pid>/lwp/<lwpid>/lwpctl */
256         int             lwp_statfd;     /* /proc/<pid>/lwp/<lwpid>/lwpstatus */
257 };
```

*unchanged_portion_omitted_*

```
**********************************************************
   65232 Wed Aug 21 14:46:25 2013
new/usr/src/lib/libproc/common/Pcore.c
3946 ::gcore
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Matthew Ahrens <mahrens@delphix.com>
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */
/*
 * Copyright 2012 DEY Storage Systems, Inc.  All rights reserved.
 * Copyright (c) 2013, Joyent, Inc. All rights reserved.
 * Copyright (c) 2013 by Delphix. All rights reserved.
 */

#include <sys/types.h>
#include <sys/utsname.h>
#include <sys/sysmacros.h>
#include <sys/proc.h>

#include <alloca.h>
#include <rtld_db.h>
#include <libgen.h>
#include <limits.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <gelf.h>
#include <stddef.h>
#include <signal.h>

#include "libproc.h"
#include "Pcontrol.h"
#include "P32ton.h"
#include "Putil.h"

/*
 * Pcore.c - Code to initialize a ps_prochandle from a core dump.  We
 * allocate an additional structure to hold information from the core
 * file, and attach this to the standard ps_prochandle in place of the
 * ability to examine /proc/<pid>/ files.
 */
```



```
/*
 * Basic i/o function for reading and writing from the process address space
 * stored in the core file and associated shared libraries.  We compute the
 * appropriate fd and offsets, and let the provided prw function do the rest.
 */
static ssize_t
core_rw(struct ps_prochandle *P, void *buf, size_t n, uintptr_t addr,
    ssize_t (*prw)(int, void *, size_t, off64_t))
{
        ssize_t resid = n;

        while (resid != 0) {
                map_info_t *mp = Paddr2mptr(P, addr);

                uintptr_t mapoff;
                ssize_t len;
                off64_t off;
                int fd;

                if (mp == NULL)
                        break;  /* No mapping for this address */

                if (mp->map_pmap.pr_mflags & MA_RESERVED1) {
                        if (mp->map_file == NULL || mp->map_file->file_fd < 0)
                                break;  /* No file or file not open */

                        fd = mp->map_file->file_fd;
                } else
                        fd = P->asfd;

                mapoff = addr - mp->map_pmap.pr_vaddr;
                len = MIN(resid, mp->map_pmap.pr_size - mapoff);
                off = mp->map_offset + mapoff;

                if ((len = prw(fd, buf, len, off)) <= 0)
                        break;

                resid -= len;
                addr += len;
                buf = (char *)buf + len;
        }

        /*
         * Important: Be consistent with the behavior of i/o on the as file:
         * writing to an invalid address yields EIO; reading from an invalid
         * address falls through to returning success and zero bytes.
         */
        if (resid == n && n != 0 && prw != pread64) {
                errno = EIO;
                return (-1);
        }

        return (n - resid);
}

/*ARGSUSED*/
static ssize_t
Pread_core(struct ps_prochandle *P, void *buf, size_t n, uintptr_t addr,
    void *data)
Pread_core(struct ps_prochandle *P, void *buf, size_t n, uintptr_t addr)
{
        return (core_rw(P, buf, n, addr, pread64));
}

/*ARGSUSED*/
static ssize_t
```

```
125 Pwrite_core(struct ps_prochandle *P, const void *buf, size_t n, uintptr_t addr,
126     void *data)
121 Pwrite_core(struct ps_prochandle *P, const void *buf, size_t n, uintptr_t addr)
127 {
128         return (core_rw(P, (void *)buf, n, addr,
129             (ssize_t (*)(int, void *, size_t, off64_t)) pwrite64));
130 }

132 /*ARGSUSED*/
133 static int
134 Pcred_core(struct ps_prochandle *P, prcred_t *pcrp, int ngroups, void *data)
135 {
136         core_info_t *core = data;
127 static const ps_rwops_t P_core_ops = { Pread_core, Pwrite_core };

138         if (core->core_cred != NULL) {
139                 /*
140                  * Avoid returning more supplementary group data than the
141                  * caller has allocated in their buffer.  We expect them to
142                  * check pr_ngroups afterward and potentially call us again.
143                  */
144                 ngroups = MIN(ngroups, core->core_cred->pr_ngroups);

146                 (void) memcpy(pcrp, core->core_cred,
147                     sizeof (prcred_t) + (ngroups - 1) * sizeof (gid_t));

149                 return (0);
150         }

152         errno = ENODATA;
153         return (-1);
154 }

156 /*ARGSUSED*/
157 static int
158 Ppriv_core(struct ps_prochandle *P, prpriv_t **pprv, void *data)
159 {
160         core_info_t *core = data;

162         if (core->core_priv == NULL) {
163                 errno = ENODATA;
164                 return (-1);
165         }

167         *pprv = malloc(core->core_priv_size);
168         if (*pprv == NULL) {
169                 return (-1);
170         }

172         (void) memcpy(*pprv, core->core_priv, core->core_priv_size);
173         return (0);
174 }

176 /*ARGSUSED*/
177 static const psinfo_t *
178 Ppsinfo_core(struct ps_prochandle *P, psinfo_t *psinfo, void *data)
179 {
180         return (&P->psinfo);
181 }

183 /*ARGSUSED*/
184 static void
185 Pfini_core(struct ps_prochandle *P, void *data)
186 {
187         core_info_t *core = data;
```



```
189         if (core != NULL) {
190                 extern void __priv_free_info(void *);
191                 lwp_info_t *nlwp, *lwp = list_next(&core->core_lwp_head);
192                 int i;

194                 for (i = 0; i < core->core_nlwp; i++, lwp = nlwp) {
195                         nlwp = list_next(lwp);
196 #ifdef __sparc
197                         if (lwp->lwp_gwins != NULL)
198                                 free(lwp->lwp_gwins);
199                         if (lwp->lwp_xregs != NULL)
200                                 free(lwp->lwp_xregs);
201                         if (lwp->lwp_asrs != NULL)
202                                 free(lwp->lwp_asrs);
203 #endif
204                         free(lwp);
205                 }

207                 if (core->core_platform != NULL)
208                         free(core->core_platform);
209                 if (core->core_uts != NULL)
210                         free(core->core_uts);
211                 if (core->core_cred != NULL)
212                         free(core->core_cred);
213                 if (core->core_priv != NULL)
214                         free(core->core_priv);
215                 if (core->core_privinfo != NULL)
216                         __priv_free_info(core->core_privinfo);
217                 if (core->core_ppii != NULL)
218                         free(core->core_ppii);
219                 if (core->core_zonename != NULL)
220                         free(core->core_zonename);
221 #if defined(__i386) || defined(__amd64)
222                 if (core->core_ldt != NULL)
223                         free(core->core_ldt);
224 #endif

226                 free(core);
227         }
228 }

230 /*ARGSUSED*/
231 static char *
232 Pplatform_core(struct ps_prochandle *P, char *s, size_t n, void *data)
233 {
234         core_info_t *core = data;

236         if (core->core_platform == NULL) {
237                 errno = ENODATA;
238                 return (NULL);
239         }
240         (void) strncpy(s, core->core_platform, n - 1);
241         s[n - 1] = '\0';
242         return (s);
243 }

245 /*ARGSUSED*/
246 static int
247 Puname_core(struct ps_prochandle *P, struct utsname *u, void *data)
248 {
249         core_info_t *core = data;

251         if (core->core_uts == NULL) {
252                 errno = ENODATA;
253                 return (-1);
254         }
```

```
255         (void) memcpy(u, core->core_uts, sizeof (struct utsname));
256         return (0);
257 }

259 /*ARGSUSED*/
260 static char *
261 Pzonename_core(struct ps_prochandle *P, char *s, size_t n, void *data)
262 {
263         core_info_t *core = data;

265         if (core->core_zonename == NULL) {
266                 errno = ENODATA;
267                 return (NULL);
268         }
269         (void) strlcpy(s, core->core_zonename, n);
270         return (s);
271 }

273 #if defined(__i386) || defined(__amd64)
274 /*ARGSUSED*/
275 static int
276 Pldt_core(struct ps_prochandle *P, struct ssd *pldt, int nldt, void *data)
277 {
278         core_info_t *core = data;

280         if (pldt == NULL || nldt == 0)
281                 return (core->core_nldt);

283         if (core->core_ldt != NULL) {
284                 nldt = MIN(nldt, core->core_nldt);

286                 (void) memcpy(pldt, core->core_ldt,
287                     nldt * sizeof (struct ssd));

289                 return (nldt);
290         }

292         errno = ENODATA;
293         return (-1);
294 }
295 #endif

297 static const ps_ops_t P_core_ops = {
298         .pop_pread      = Pread_core,
299         .pop_pwrite     = Pwrite_core,
300         .pop_cred       = Pcred_core,
301         .pop_priv       = Ppriv_core,
302         .pop_psinfo     = Ppsinfo_core,
303         .pop_fini       = Pfini_core,
304         .pop_platform   = Pplatform_core,
305         .pop_uname      = Puname_core,
306         .pop_zonename   = Pzonename_core,
307 #if defined(__i386) || defined(__amd64)
308         .pop_ldt        = Pldt_core
309 #endif
310 };

312 /*
313  * Return the lwp_info_t for the given lwpid.  If no such lwpid has been
314  * encountered yet, allocate a new structure and return a pointer to it.
315  * Create a list of lwp_info_t structures sorted in decreasing lwp_id order.
316  */
317 static lwp_info_t *
318 lwpid2info(struct ps_prochandle *P, lwpid_t id)
319 {
320         core_info_t *core = P->data;
```



```
321         lwp_info_t *lwp = list_next(&core->core_lwp_head);
137         lwp_info_t *lwp = list_next(&P->core->core_lwp_head);
322         lwp_info_t *next;
323         uint_t i;

325         for (i = 0; i < core->core_nlwp; i++, lwp = list_next(lwp)) {
141         for (i = 0; i < P->core->core_nlwp; i++, lwp = list_next(lwp)) {
326                 if (lwp->lwp_id == id) {
327                         core->core_lwp = lwp;
143                         P->core->core_lwp = lwp;
328                         return (lwp);
329                 }
330                 if (lwp->lwp_id < id) {
331                         break;
332                 }
333         }

335         next = lwp;
336         if ((lwp = calloc(1, sizeof (lwp_info_t))) == NULL)
337                 return (NULL);

339         list_link(lwp, next);
340         lwp->lwp_id = id;

342         core->core_lwp = lwp;
343         core->core_nlwp++;
158         P->core->core_lwp = lwp;
159         P->core->core_nlwp++;

345         return (lwp);
346 }

348 /*
349  * The core file itself contains a series of NOTE segments containing saved
350  * structures from /proc at the time the process died.  For each note we
351  * comprehend, we define a function to read it in from the core file,
352  * convert it to our native data model if necessary, and store it inside
353  * the ps_prochandle.  Each function is invoked by Pfgrab_core() with the
354  * seek pointer on P->asfd positioned appropriately.  We populate a table
355  * of pointers to these note functions below.
356  */

358 static int
359 note_pstatus(struct ps_prochandle *P, size_t nbytes)
360 {
361 #ifdef _LP64
362         core_info_t *core = P->data;

364         if (core->core_dmodel == PR_MODEL_ILP32) {
178         if (P->core->core_dmodel == PR_MODEL_ILP32) {
365                 pstatus32_t ps32;

367                 if (nbytes < sizeof (pstatus32_t) ||
368                     read(P->asfd, &ps32, sizeof (ps32)) != sizeof (ps32))
369                         goto err;

371                 pstatus_32_to_n(&ps32, &P->status);

373         } else
374 #endif
375         if (nbytes < sizeof (pstatus_t) ||
376             read(P->asfd, &P->status, sizeof (pstatus_t)) != sizeof (pstatus_t))
377                 goto err;

379         P->orig_status = P->status;
380         P->pid = P->status.pr_pid;
```

```
382         return (0);

384 err:
385         dprintf("Pgrab_core: failed to read NT_PSTATUS\n");
386         return (-1);
387 }

389 static int
390 note_lwpstatus(struct ps_prochandle *P, size_t nbytes)
391 {
392         lwp_info_t *lwp;
393         lwpstatus_t lps;

395 #ifdef _LP64
396         core_info_t *core = P->data;

398         if (core->core_dmodel == PR_MODEL_ILP32) {
210         if (P->core->core_dmodel == PR_MODEL_ILP32) {
399                 lwpstatus32_t l32;

401                 if (nbytes < sizeof (lwpstatus32_t) ||
402                     read(P->asfd, &l32, sizeof (l32)) != sizeof (l32))
403                         goto err;

405                 lwpstatus_32_to_n(&l32, &lps);
406         } else
407 #endif
408         if (nbytes < sizeof (lwpstatus_t) ||
409             read(P->asfd, &lps, sizeof (lps)) != sizeof (lps))
410                 goto err;

412         if ((lwp = lwpid2info(P, lps.pr_lwpid)) == NULL) {
413                 dprintf("Pgrab_core: failed to add NT_LWPSTATUS\n");
414                 return (-1);
415         }

417         /*
418          * Erase a useless and confusing artifact of the kernel implementation:
419          * the lwps which did *not* create the core will show SIGKILL.  We can
420          * be assured this is bogus because SIGKILL can't produce core files.
421          */
422         if (lps.pr_cursig == SIGKILL)
423                 lps.pr_cursig = 0;

425         (void) memcpy(&lwp->lwp_status, &lps, sizeof (lps));
426         return (0);

428 err:
429         dprintf("Pgrab_core: failed to read NT_LWPSTATUS\n");
430         return (-1);
431 }

433 static int
434 note_psinfo(struct ps_prochandle *P, size_t nbytes)
435 {
436 #ifdef _LP64
437         core_info_t *core = P->data;

439         if (core->core_dmodel == PR_MODEL_ILP32) {
249         if (P->core->core_dmodel == PR_MODEL_ILP32) {
440                 psinfo32_t ps32;

442                 if (nbytes < sizeof (psinfo32_t) ||
443                     read(P->asfd, &ps32, sizeof (ps32)) != sizeof (ps32))
444                         goto err;
```



```
446                 psinfo_32_to_n(&ps32, &P->psinfo);
447         } else
448 #endif
449         if (nbytes < sizeof (psinfo_t) ||
450             read(P->asfd, &P->psinfo, sizeof (psinfo_t)) != sizeof (psinfo_t))
451                 goto err;

453         dprintf("pr_fname = <%s>\n", P->psinfo.pr_fname);
454         dprintf("pr_psargs = <%s>\n", P->psinfo.pr_psargs);
455         dprintf("pr_wstat = 0x%x\n", P->psinfo.pr_wstat);

457         return (0);

459 err:
460         dprintf("Pgrab_core: failed to read NT_PSINFO\n");
461         return (-1);
462 }

464 static int
465 note_lwpsinfo(struct ps_prochandle *P, size_t nbytes)
466 {
467         lwp_info_t *lwp;
468         lwpsinfo_t lps;

470 #ifdef _LP64
471         core_info_t *core = P->data;

473         if (core->core_dmodel == PR_MODEL_ILP32) {
281         if (P->core->core_dmodel == PR_MODEL_ILP32) {
474                 lwpsinfo32_t l32;

476                 if (nbytes < sizeof (lwpsinfo32_t) ||
477                     read(P->asfd, &l32, sizeof (l32)) != sizeof (l32))
478                         goto err;

480                 lwpsinfo_32_to_n(&l32, &lps);
481         } else
482 #endif
483         if (nbytes < sizeof (lwpsinfo_t) ||
484             read(P->asfd, &lps, sizeof (lps)) != sizeof (lps))
485                 goto err;

487         if ((lwp = lwpid2info(P, lps.pr_lwpid)) == NULL) {
488                 dprintf("Pgrab_core: failed to add NT_LWPSINFO\n");
489                 return (-1);
490         }

492         (void) memcpy(&lwp->lwp_psinfo, &lps, sizeof (lps));
493         return (0);

495 err:
496         dprintf("Pgrab_core: failed to read NT_LWPSINFO\n");
497         return (-1);
498 }
_____unchanged_portion_omitted_

520 static int
521 note_platform(struct ps_prochandle *P, size_t nbytes)
522 {
523         core_info_t *core = P->data;
524         char *plat;

526         if (core->core_platform != NULL)
333         if (P->core->core_platform != NULL)
527                 return (0);     /* Already seen */
```

```
529         if (nbytes != 0 && ((plat = malloc(nbytes + 1)) != NULL)) {
530                 if (read(P->asfd, plat, nbytes) != nbytes) {
531                         dprintf("Pgrab_core: failed to read NT_PLATFORM\n");
532                         free(plat);
533                         return (-1);
534                 }
535                 plat[nbytes - 1] = '\0';
536                 core->core_platform = plat;
343                 P->core->core_platform = plat;
537         }

539         return (0);
540 }

542 static int
543 note_utsname(struct ps_prochandle *P, size_t nbytes)
544 {
545         core_info_t *core = P->data;
546         size_t ubytes = sizeof (struct utsname);
547         struct utsname *utsp;

549         if (core->core_uts != NULL || nbytes < ubytes)
355         if (P->core->core_uts != NULL || nbytes < ubytes)
550                 return (0);     /* Already seen or bad size */

552         if ((utsp = malloc(ubytes)) == NULL)
553                 return (-1);

555         if (read(P->asfd, utsp, ubytes) != ubytes) {
556                 dprintf("Pgrab_core: failed to read NT_UTSNAME\n");
557                 free(utsp);
558                 return (-1);
559         }

561         if (_libproc_debug) {
562                 dprintf("uts.sysname = \"%s\"\n", utsp->sysname);
563                 dprintf("uts.nodename = \"%s\"\n", utsp->nodename);
564                 dprintf("uts.release = \"%s\"\n", utsp->release);
565                 dprintf("uts.version = \"%s\"\n", utsp->version);
566                 dprintf("uts.machine = \"%s\"\n", utsp->machine);
567         }

569         core->core_uts = utsp;
375         P->core->core_uts = utsp;
570         return (0);
571 }

573 static int
574 note_content(struct ps_prochandle *P, size_t nbytes)
575 {
576         core_info_t *core = P->data;
577         core_content_t content;

579         if (sizeof (core->core_content) != nbytes)
384         if (sizeof (P->core->core_content) != nbytes)
580                 return (-1);

582         if (read(P->asfd, &content, sizeof (content)) != sizeof (content))
583                 return (-1);

585         core->core_content = content;
390         P->core->core_content = content;

587         dprintf("core content = %llx\n", content);
```



```
589         return (0);
590 }

592 static int
593 note_cred(struct ps_prochandle *P, size_t nbytes)
594 {
595         core_info_t *core = P->data;
596         prcred_t *pcrp;
597         int ngroups;
598         const size_t min_size = sizeof (prcred_t) - sizeof (gid_t);

600         /*
601          * We allow for prcred_t notes that are actually smaller than a
602          * prcred_t since the last member isn't essential if there are
603          * no group memberships. This allows for more flexibility when it
604          * comes to slightly malformed -- but still valid -- notes.
605          */
606         if (core->core_cred != NULL || nbytes < min_size)
410         if (P->core->core_cred != NULL || nbytes < min_size)
607                 return (0);     /* Already seen or bad size */

609         ngroups = (nbytes - min_size) / sizeof (gid_t);
610         nbytes = sizeof (prcred_t) + (ngroups - 1) * sizeof (gid_t);

612         if ((pcrp = malloc(nbytes)) == NULL)
613                 return (-1);

615         if (read(P->asfd, pcrp, nbytes) != nbytes) {
616                 dprintf("Pgrab_core: failed to read NT_PRCRED\n");
617                 free(pcrp);
618                 return (-1);
619         }

621         if (pcrp->pr_ngroups > ngroups) {
622                 dprintf("pr_ngroups = %d; resetting to %d based on note size\n",
623                     pcrp->pr_ngroups, ngroups);
624                 pcrp->pr_ngroups = ngroups;
625         }

627         core->core_cred = pcrp;
431         P->core->core_cred = pcrp;
628         return (0);
629 }

631 #if defined(__i386) || defined(__amd64)
632 static int
633 note_ldt(struct ps_prochandle *P, size_t nbytes)
634 {
635         core_info_t *core = P->data;
636         struct ssd *pldt;
637         uint_t nldt;

639         if (core->core_ldt != NULL || nbytes < sizeof (struct ssd))
442         if (P->core->core_ldt != NULL || nbytes < sizeof (struct ssd))
640                 return (0);     /* Already seen or bad size */

642         nldt = nbytes / sizeof (struct ssd);
643         nbytes = nldt * sizeof (struct ssd);

645         if ((pldt = malloc(nbytes)) == NULL)
646                 return (-1);

648         if (read(P->asfd, pldt, nbytes) != nbytes) {
649                 dprintf("Pgrab_core: failed to read NT_LDT\n");
650                 free(pldt);
651                 return (-1);
```

```
652         }

654         core->core_ldt = pldt;
655         core->core_nldt = nldt;
457         P->core->core_ldt = pldt;
458         P->core->core_nldt = nldt;
656         return (0);
657 }
658 #endif  /* __i386 */

660 static int
661 note_priv(struct ps_prochandle *P, size_t nbytes)
662 {
663         core_info_t *core = P->data;
664         prpriv_t *pprvp;

666         if (core->core_priv != NULL || nbytes < sizeof (prpriv_t))
468         if (P->core->core_priv != NULL || nbytes < sizeof (prpriv_t))
667                 return (0);     /* Already seen or bad size */

669         if ((pprvp = malloc(nbytes)) == NULL)
670                 return (-1);

672         if (read(P->asfd, pprvp, nbytes) != nbytes) {
673                 dprintf("Pgrab_core: failed to read NT_PRPRIV\n");
674                 free(pprvp);
675                 return (-1);
676         }

678         core->core_priv = pprvp;
679         core->core_priv_size = nbytes;
480         P->core->core_priv = pprvp;
481         P->core->core_priv_size = nbytes;
680         return (0);
681 }

683 static int
684 note_priv_info(struct ps_prochandle *P, size_t nbytes)
685 {
686         core_info_t *core = P->data;
687         extern void *__priv_parse_info();
688         priv_impl_info_t *ppii;

690         if (core->core_privinfo != NULL ||
491         if (P->core->core_privinfo != NULL ||
691             nbytes < sizeof (priv_impl_info_t))
692                 return (0);     /* Already seen or bad size */

694         if ((ppii = malloc(nbytes)) == NULL)
695                 return (-1);

697         if (read(P->asfd, ppii, nbytes) != nbytes ||
698             PRIV_IMPL_INFO_SIZE(ppii) != nbytes) {
699                 dprintf("Pgrab_core: failed to read NT_PRPRIVINFO\n");
700                 free(ppii);
701                 return (-1);
702         }

704         core->core_privinfo = __priv_parse_info(ppii);
705         core->core_ppii = ppii;
505         P->core->core_privinfo = __priv_parse_info(ppii);
506         P->core->core_ppii = ppii;
706         return (0);
707 }

709 static int
```



```
710 note_zonename(struct ps_prochandle *P, size_t nbytes)
711 {
712         core_info_t *core = P->data;
713         char *zonename;

715         if (core->core_zonename != NULL)
515         if (P->core->core_zonename != NULL)
716                 return (0);     /* Already seen */

718         if (nbytes != 0) {
719                 if ((zonename = malloc(nbytes)) == NULL)
720                         return (-1);
721                 if (read(P->asfd, zonename, nbytes) != nbytes) {
722                         dprintf("Pgrab_core: failed to read NT_ZONENAME\n");
723                         free(zonename);
724                         return (-1);
725                 }
726                 zonename[nbytes - 1] = '\0';
727                 core->core_zonename = zonename;
527                 P->core->core_zonename = zonename;
728         }

730         return (0);
731 }

733 static int
734 note_auxv(struct ps_prochandle *P, size_t nbytes)
735 {
736         size_t n, i;

738 #ifdef _LP64
739         core_info_t *core = P->data;

741         if (core->core_dmodel == PR_MODEL_ILP32) {
539         if (P->core->core_dmodel == PR_MODEL_ILP32) {
742                 auxv32_t *a32;

744                 n = nbytes / sizeof (auxv32_t);
745                 nbytes = n * sizeof (auxv32_t);
746                 a32 = alloca(nbytes);

748                 if (read(P->asfd, a32, nbytes) != nbytes) {
749                         dprintf("Pgrab_core: failed to read NT_AUXV\n");
750                         return (-1);
751                 }

753                 if ((P->auxv = malloc(sizeof (auxv_t) * (n + 1))) == NULL)
754                         return (-1);

756                 for (i = 0; i < n; i++)
757                         auxv_32_to_n(&a32[i], &P->auxv[i]);

759         } else {
760 #endif
761                 n = nbytes / sizeof (auxv_t);
762                 nbytes = n * sizeof (auxv_t);

764                 if ((P->auxv = malloc(nbytes + sizeof (auxv_t))) == NULL)
765                         return (-1);

767                 if (read(P->asfd, P->auxv, nbytes) != nbytes) {
768                         free(P->auxv);
769                         P->auxv = NULL;
770                         return (-1);
771                 }
772 #ifdef _LP64
```

```
773         }
774 #endif

776         if (_libproc_debug) {
777                 for (i = 0; i < n; i++) {
778                         dprintf("P->auxv[%lu] = ( %d, 0x%lx )\n", (ulong_t)i,
779                             P->auxv[i].a_type, P->auxv[i].a_un.a_val);
780                 }
781         }

783         /*
784          * Defensive coding for loops which depend upon the auxv array being
785          * terminated by an AT_NULL element; in each case, we've allocated
786          * P->auxv to have an additional element which we force to be AT_NULL.
787          */
788         P->auxv[n].a_type = AT_NULL;
789         P->auxv[n].a_un.a_val = 0L;
790         P->nauxv = (int)n;

792         return (0);
793 }

795 #ifdef __sparc
796 static int
797 note_xreg(struct ps_prochandle *P, size_t nbytes)
798 {
799         core_info_t *core = P->data;
800         lwp_info_t *lwp = core->core_lwp;
597         lwp_info_t *lwp = P->core->core_lwp;
801         size_t xbytes = sizeof (prxregset_t);
802         prxregset_t *xregs;

804         if (lwp == NULL || lwp->lwp_xregs != NULL || nbytes < xbytes)
805                 return (0);     /* No lwp yet, already seen, or bad size */

807         if ((xregs = malloc(xbytes)) == NULL)
808                 return (-1);

810         if (read(P->asfd, xregs, xbytes) != xbytes) {
811                 dprintf("Pgrab_core: failed to read NT_PRXREG\n");
812                 free(xregs);
813                 return (-1);
814         }

816         lwp->lwp_xregs = xregs;
817         return (0);
818 }

820 static int
821 note_gwindows(struct ps_prochandle *P, size_t nbytes)
822 {
823         core_info_t *core = P->data;
824         lwp_info_t *lwp = core->core_lwp;
620         lwp_info_t *lwp = P->core->core_lwp;

826         if (lwp == NULL || lwp->lwp_gwins != NULL || nbytes == 0)
827                 return (0);     /* No lwp yet or already seen or no data */

829         if ((lwp->lwp_gwins = malloc(sizeof (gwindows_t))) == NULL)
830                 return (-1);

832         /*
833          * Since the amount of gwindows data varies with how many windows were
834          * actually saved, we just read up to the minimum of the note size
835          * and the size of the gwindows_t type.  It doesn't matter if the read
836          * fails since we have to zero out gwindows first anyway.
```



```
837          */
838 #ifdef _LP64
839         if (core->core_dmodel == PR_MODEL_ILP32) {
635         if (P->core->core_dmodel == PR_MODEL_ILP32) {
840                 gwindows32_t g32;

842                 (void) memset(&g32, 0, sizeof (g32));
843                 (void) read(P->asfd, &g32, MIN(nbytes, sizeof (g32)));
844                 gwindows_32_to_n(&g32, lwp->lwp_gwins);

846         } else {
847 #endif
848                 (void) memset(lwp->lwp_gwins, 0, sizeof (gwindows_t));
849                 (void) read(P->asfd, lwp->lwp_gwins,
850                     MIN(nbytes, sizeof (gwindows_t)));
851 #ifdef _LP64
852         }
853 #endif
854         return (0);
855 }

857 #ifdef __sparcv9
858 static int
859 note_asrs(struct ps_prochandle *P, size_t nbytes)
860 {
861         core_info_t *core = P->data;
862         lwp_info_t *lwp = core->core_lwp;
657         lwp_info_t *lwp = P->core->core_lwp;
863         int64_t *asrs;

865         if (lwp == NULL || lwp->lwp_asrs != NULL || nbytes < sizeof (asrset_t))
866                 return (0);     /* No lwp yet, already seen, or bad size */

868         if ((asrs = malloc(sizeof (asrset_t))) == NULL)
869                 return (-1);

871         if (read(P->asfd, asrs, sizeof (asrset_t)) != sizeof (asrset_t)) {
872                 dprintf("Pgrab_core: failed to read NT_ASRS\n");
873                 free(asrs);
874                 return (-1);
875         }

877         lwp->lwp_asrs = asrs;
878         return (0);
879 }
880 #endif  /* __sparcv9 */
881 #endif  /* __sparc */

883 static int
884 note_spymaster(struct ps_prochandle *P, size_t nbytes)
885 {
886 #ifdef _LP64
887         core_info_t *core = P->data;

889         if (core->core_dmodel == PR_MODEL_ILP32) {
682         if (P->core->core_dmodel == PR_MODEL_ILP32) {
890                 psinfo32_t ps32;

892                 if (nbytes < sizeof (psinfo32_t) ||
893                     read(P->asfd, &ps32, sizeof (ps32)) != sizeof (ps32))
894                         goto err;

896                 psinfo_32_to_n(&ps32, &P->spymaster);
897         } else
898 #endif
899         if (nbytes < sizeof (psinfo_t) || read(P->asfd,
```

```
 900             &P->spymaster, sizeof (psinfo_t)) != sizeof (psinfo_t))
 901                 goto err;

 903         dprintf("spymaster pr_fname = <%s>\n", P->psinfo.pr_fname);
 904         dprintf("spymaster pr_psargs = <%s>\n", P->psinfo.pr_psargs);
 905         dprintf("spymaster pr_wstat = 0x%x\n", P->psinfo.pr_wstat);

 907         return (0);

 909 err:
 910         dprintf("Pgrab_core: failed to read NT_SPYMASTER\n");
 911         return (-1);
 912 }
_____unchanged_portion_omitted_

1033 /*
1034  * Add information on the address space mapping described by the given
1035  * PT_LOAD program header.  We fill in more information on the mapping later.
1036  */
1037 static int
1038 core_add_mapping(struct ps_prochandle *P, GElf_Phdr *php)
1039 {
1040         core_info_t *core = P->data;
1041         prmap_t pmap;

1043         dprintf("mapping base %llx filesz %llu memsz %llu offset %llu\n",
1044             (u_longlong_t)php->p_vaddr, (u_longlong_t)php->p_filesz,
1045             (u_longlong_t)php->p_memsz, (u_longlong_t)php->p_offset);

1047         pmap.pr_vaddr = (uintptr_t)php->p_vaddr;
1048         pmap.pr_size = php->p_memsz;

1050         /*
1051          * If Pgcore() or elfcore() fail to write a mapping, they will set
1052          * PF_SUNW_FAILURE in the Phdr and try to stash away the errno for us.
1053          */
1054         if (php->p_flags & PF_SUNW_FAILURE) {
1055                 core_report_mapping(P, php);
1056         } else if (php->p_filesz != 0 && php->p_offset >= core->core_size) {
 848         } else if (php->p_filesz != 0 && php->p_offset >= P->core->core_size) {
1057                 Perror_printf(P, "core file may be corrupt -- data for mapping "
1058                     "at %p is missing\n", (void *)(uintptr_t)php->p_vaddr);
1059                 dprintf("core file may be corrupt -- data for mapping "
1060                     "at %p is missing\n", (void *)(uintptr_t)php->p_vaddr);
1061         }

1063         /*
1064          * The mapping name and offset will hopefully be filled in
1065          * by the librtld_db agent.  Unfortunately, if it isn't a
1066          * shared library mapping, this information is gone forever.
1067          */
1068         pmap.pr_mapname[0] = '\0';
1069         pmap.pr_offset = 0;

1071         pmap.pr_mflags = 0;
1072         if (php->p_flags & PF_R)
1073                 pmap.pr_mflags |= MA_READ;
1074         if (php->p_flags & PF_W)
1075                 pmap.pr_mflags |= MA_WRITE;
1076         if (php->p_flags & PF_X)
1077                 pmap.pr_mflags |= MA_EXEC;

1079         if (php->p_filesz == 0)
1080                 pmap.pr_mflags |= MA_RESERVED1;

1082         /*
```



```
1083          * At the time of adding this mapping, we just zero the pagesize.
1084          * Once we've processed more of the core file, we'll have the
1085          * pagesize from the auxv's AT_PAGESZ element and we can fill this in.
1086          */
1087         pmap.pr_pagesize = 0;

1089         /*
1090          * Unfortunately whether or not the mapping was a System V
1091          * shared memory segment is lost.  We use -1 to mark it as not shm.
1092          */
1093         pmap.pr_shmid = -1;

1095         return (Padd_mapping(P, php->p_offset, NULL, &pmap));
1096 }
_____unchanged_portion_omitted_

1684 /*
1685  * Librtld_db agent callback for iterating over load object mappings.
1686  * For each load object, we allocate a new file_info_t, perform naming,
1687  * and attempt to construct a symbol table for the load object.
1688  */
1689 static int
1690 core_iter_mapping(const rd_loadobj_t *rlp, struct ps_prochandle *P)
1691 {
1692         core_info_t *core = P->data;
1693         char lname[PATH_MAX], buf[PATH_MAX];
1694         file_info_t *fp;
1695         map_info_t *mp;

1697         if (Pread_string(P, lname, PATH_MAX, (off_t)rlp->rl_nameaddr) <= 0) {
1698                 dprintf("failed to read name %p\n", (void *)rlp->rl_nameaddr);
1699                 return (1); /* Keep going; forget this if we can't get a name */
1700         }

1702         dprintf("rd_loadobj name = \"%s\" rl_base = %p\n",
1703             lname, (void *)rlp->rl_base);

1705         if ((mp = Paddr2mptr(P, rlp->rl_base)) == NULL) {
1706                 dprintf("no mapping for %p\n", (void *)rlp->rl_base);
1707                 return (1); /* No mapping; advance to next mapping */
1708         }

1710         /*
1711          * Create a new file_info_t for this mapping, and therefore for
1712          * this load object.
1713          *
1714          * If there's an ELF header at the beginning of this mapping,
1715          * file_info_new() will try to use its section headers to
1716          * identify any other mappings that belong to this load object.
1717          */
1718         if ((fp = mp->map_file) == NULL &&
1719             (fp = file_info_new(P, mp)) == NULL) {
1720                 core->core_errno = errno;
1511                 P->core->core_errno = errno;
1721                 dprintf("failed to malloc mapping data\n");
1722                 return (0); /* Abort */
1723         }
1724         fp->file_map = mp;

1726         /* Create a local copy of the load object representation */
1727         if ((fp->file_lo = calloc(1, sizeof (rd_loadobj_t))) == NULL) {
1728                 core->core_errno = errno;
1519                 P->core->core_errno = errno;
1729                 dprintf("failed to malloc mapping data\n");
1730                 return (0); /* Abort */
1731         }
```

```
1732         *fp->file_lo = *rlp;

1734         if (lname[0] != '\0') {
1735                 /*
1736                  * Naming dance part 1: if we got a name from librtld_db, then
1737                  * copy this name to the prmap_t if it is unnamed.  If the
1738                  * file_info_t is unnamed, name it after the lname.
1739                  */
1740                 if (mp->map_pmap.pr_mapname[0] == '\0') {
1741                         (void) strncpy(mp->map_pmap.pr_mapname, lname, PRMAPSZ);
1742                         mp->map_pmap.pr_mapname[PRMAPSZ - 1] = '\0';
1743                 }

1745                 if (fp->file_lname == NULL)
1746                         fp->file_lname = strdup(lname);

1748         } else if (fp->file_lname == NULL &&
1749             mp->map_pmap.pr_mapname[0] != '\0') {
1750                 /*
1751                  * Naming dance part 2: if the mapping is named and the
1752                  * file_info_t is not, name the file after the mapping.
1753                  */
1754                 fp->file_lname = strdup(mp->map_pmap.pr_mapname);
1755         }

1757         if ((fp->file_rname == NULL) &&
1758             (Pfindmap(P, mp, buf, sizeof (buf)) != NULL))
1759                 fp->file_rname = strdup(buf);

1761         if (fp->file_lname != NULL)
1762                 fp->file_lbase = basename(fp->file_lname);
1763         if (fp->file_rname != NULL)
1764                 fp->file_rbase = basename(fp->file_rname);

1766         /* Associate the file and the mapping. */
1767         (void) strncpy(fp->file_pname, mp->map_pmap.pr_mapname, PRMAPSZ);
1768         fp->file_pname[PRMAPSZ - 1] = '\0';

1770         /*
1771          * If no section headers were available then we'll have to
1772          * identify this load object's other mappings with what we've
1773          * got: the start and end of the object's corresponding
1774          * address space.
1775          */
1776         if (fp->file_saddrs == NULL) {
1777                 for (mp = fp->file_map + 1; mp < P->mappings + P->map_count &&
1778                     mp->map_pmap.pr_vaddr < rlp->rl_bend; mp++) {

1780                         if (mp->map_file == NULL) {
1781                                 dprintf("core_iter_mapping %s: associating "
1782                                     "segment at %p\n",
1783                                     fp->file_pname,
1784                                     (void *)mp->map_pmap.pr_vaddr);
1785                                 mp->map_file = fp;
1786                                 fp->file_ref++;
1787                         } else {
1788                                 dprintf("core_iter_mapping %s: segment at "
1789                                     "%p already associated with %s\n",
1790                                     fp->file_pname,
1791                                     (void *)mp->map_pmap.pr_vaddr,
1792                                     (mp == fp->file_map ? "this file" :
1793                                     mp->map_file->file_pname));
1794                         }
1795                 }
1796         }
```



```
1798         /* Ensure that all this file's mappings are named. */
1799         for (mp = fp->file_map; mp < P->mappings + P->map_count &&
1800             mp->map_file == fp; mp++) {
1801                 if (mp->map_pmap.pr_mapname[0] == '\0' &&
1802                     !(mp->map_pmap.pr_mflags & MA_BREAK)) {
1803                         (void) strncpy(mp->map_pmap.pr_mapname, fp->file_pname,
1804                             PRMAPSZ);
1805                         mp->map_pmap.pr_mapname[PRMAPSZ - 1] = '\0';
1806                 }
1807         }

1809         /* Attempt to build a symbol table for this file. */
1810         Pbuild_file_symtab(P, fp);
1811         if (fp->file_elf == NULL)
1812                 dprintf("core_iter_mapping: no symtab for %s\n",
1813                     fp->file_pname);

1815         /* Locate the start of a data segment associated with this file. */
1816         if ((mp = core_find_data(P, fp->file_elf, fp->file_lo)) != NULL) {
1817                 dprintf("found data for %s at %p (pr_offset 0x%llx)\n",
1818                     fp->file_pname, (void *)fp->file_lo->rl_data_base,
1819                     mp->map_pmap.pr_offset);
1820         } else {
1821                 dprintf("core_iter_mapping: no data found for %s\n",
1822                     fp->file_pname);
1823         }

1825         return (1); /* Advance to next mapping */
1826 }
_____unchanged_portion_omitted_

1985 /*
1986  * Main engine for core file initialization: given an fd for the core file
1987  * and an optional pathname, construct the ps_prochandle.  The aout_path can
1988  * either be a suggested executable pathname, or a suggested directory to
1989  * use as a possible current working directory.
1990  */
1991 struct ps_prochandle *
1992 Pfgrab_core(int core_fd, const char *aout_path, int *perr)
1993 {
1994         struct ps_prochandle *P;
1995         core_info_t *core_info;
1996         map_info_t *stk_mp, *brk_mp;
1997         const char *execname;
1998         char *interp;
1999         int i, notes, pagesize;
2000         uintptr_t addr, base_addr;
2001         struct stat64 stbuf;
2002         void *phbuf, *php;
2003         size_t nbytes;

2005         elf_file_t aout;
2006         elf_file_t core;

2008         Elf_Scn *scn, *intp_scn = NULL;
2009         Elf_Data *dp;

2011         GElf_Phdr phdr, note_phdr;
2012         GElf_Shdr shdr;
2013         GElf_Xword nleft;

2015         if (elf_version(EV_CURRENT) == EV_NONE) {
2016                 dprintf("libproc ELF version is more recent than libelf\n");
2017                 *perr = G_ELF;
2018                 return (NULL);
2019         }
```

```
2021         aout.e_elf = NULL;
2022         aout.e_fd = -1;

2024         core.e_elf = NULL;
2025         core.e_fd = core_fd;

2027         /*
2028          * Allocate and initialize a ps_prochandle structure for the core.
2029          * There are several key pieces of initialization here:
2030          *
2031          * 1. The PS_DEAD state flag marks this prochandle as a core file.
2032          *    PS_DEAD also thus prevents all operations which require state
2033          *    to be PS_STOP from operating on this handle.
2034          *
2035          * 2. We keep the core file fd in P->asfd since the core file contains
2036          *    the remnants of the process address space.
2037          *
2038          * 3. We set the P->info_valid bit because all information about the
2039          *    core is determined by the end of this function; there is no need
2040          *    for proc_update_maps() to reload mappings at any later point.
2041          *
2042          * 4. The read/write ops vector uses our core_rw() function defined
2043          *    above to handle i/o requests.
2044          */
2045         if ((P = malloc(sizeof (struct ps_prochandle))) == NULL) {
2046                 *perr = G_STRANGE;
2047                 return (NULL);
2048         }

2050         (void) memset(P, 0, sizeof (struct ps_prochandle));
2051         (void) mutex_init(&P->proc_lock, USYNC_THREAD, NULL);
2052         P->state = PS_DEAD;
2053         P->pid = (pid_t)-1;
2054         P->asfd = core.e_fd;
2055         P->ctlfd = -1;
2056         P->statfd = -1;
2057         P->agentctlfd = -1;
2058         P->agentstatfd = -1;
2059         P->zoneroot = NULL;
2060         P->info_valid = 1;
2061         Pinit_ops(&P->ops, &P_core_ops);
1851         P->ops = &P_core_ops;

2063         Pinitsym(P);

2065         /*
2066          * Fstat and open the core file and make sure it is a valid ELF core.
2067          */
2068         if (fstat64(P->asfd, &stbuf) == -1) {
2069                 *perr = G_STRANGE;
2070                 goto err;
2071         }

2073         if (core_elf_fdopen(&core, ET_CORE, perr) == -1)
2074                 goto err;

2076         /*
2077          * Allocate and initialize a core_info_t to hang off the ps_prochandle
2078          * structure.  We keep all core-specific information in this structure.
2079          */
2080         if ((core_info = calloc(1, sizeof (core_info_t))) == NULL) {
1870         if ((P->core = calloc(1, sizeof (core_info_t))) == NULL) {
2081                 *perr = G_STRANGE;
2082                 goto err;
2083         }
```



```
2085         P->data = core_info;
2086         list_link(&core_info->core_lwp_head, NULL);
2087         core_info->core_size = stbuf.st_size;
1875         list_link(&P->core->core_lwp_head, NULL);
1876         P->core->core_size = stbuf.st_size;
2088         /*
2089          * In the days before adjustable core file content, this was the
2090          * default core file content. For new core files, this value will
2091          * be overwritten by the NT_CONTENT note section.
2092          */
2093         core_info->core_content = CC_CONTENT_STACK | CC_CONTENT_HEAP |
1882         P->core->core_content = CC_CONTENT_STACK | CC_CONTENT_HEAP |
2094             CC_CONTENT_DATA | CC_CONTENT_RODATA | CC_CONTENT_ANON |
2095             CC_CONTENT_SHANON;

2097         switch (core.e_hdr.e_ident[EI_CLASS]) {
2098         case ELFCLASS32:
2099                 core_info->core_dmodel = PR_MODEL_ILP32;
1888                 P->core->core_dmodel = PR_MODEL_ILP32;
2100                 break;
2101         case ELFCLASS64:
2102                 core_info->core_dmodel = PR_MODEL_LP64;
1891                 P->core->core_dmodel = PR_MODEL_LP64;
2103                 break;
2104         default:
2105                 *perr = G_FORMAT;
2106                 goto err;
2107         }

2109         /*
2110          * Because the core file may be a large file, we can't use libelf to
2111          * read the Phdrs.  We use e_phnum and e_phentsize to simplify things.
2112          */
2113         nbytes = core.e_hdr.e_phnum * core.e_hdr.e_phentsize;

2115         if ((phbuf = malloc(nbytes)) == NULL) {
2116                 *perr = G_STRANGE;
2117                 goto err;
2118         }

2120         if (pread64(core_fd, phbuf, nbytes, core.e_hdr.e_phoff) != nbytes) {
2121                 *perr = G_STRANGE;
2122                 free(phbuf);
2123                 goto err;
2124         }

2126         /*
2127          * Iterate through the program headers in the core file.
2128          * We're interested in two types of Phdrs: PT_NOTE (which
2129          * contains a set of saved /proc structures), and PT_LOAD (which
2130          * represents a memory mapping from the process's address space).
2131          * In the case of PT_NOTE, we're interested in the last PT_NOTE
2132          * in the core file; currently the first PT_NOTE (if present)
2133          * contains /proc structs in the pre-2.6 unstructured /proc format.
2134          */
2135         for (php = phbuf, notes = 0, i = 0; i < core.e_hdr.e_phnum; i++) {
2136                 if (core.e_hdr.e_ident[EI_CLASS] == ELFCLASS64)
2137                         (void) memcpy(&phdr, php, sizeof (GElf_Phdr));
2138                 else
2139                         core_phdr_to_gelf(php, &phdr);

2141                 switch (phdr.p_type) {
2142                 case PT_NOTE:
2143                         note_phdr = phdr;
2144                         notes++;
```

```
2145                            break;

2147                    case PT_LOAD:
2148                            if (core_add_mapping(P, &phdr) == -1) {
2149                                    *perr = G_STRANGE;
2150                                    free(phbuf);
2151                                    goto err;
2152                            }
2153                            break;
2154                    }

2156                    php = (char *)php + core.e_hdr.e_phentsize;
2157            }

2159            free(phbuf);

2161            Psort_mappings(P);

2163            /*
2164             * If we couldn't find anything of type PT_NOTE, or only one PT_NOTE
2165             * was present, abort.  The core file is either corrupt or too old.
2166             */
2167            if (notes == 0 || notes == 1) {
2168                    *perr = G_NOTE;
2169                    goto err;
2170            }

2172            /*
2173             * Advance the seek pointer to the start of the PT_NOTE data
2174             */
2175            if (lseek64(P->asfd, note_phdr.p_offset, SEEK_SET) == (off64_t)-1) {
2176                    dprintf("Pgrab_core: failed to lseek to PT_NOTE data\n");
2177                    *perr = G_STRANGE;
2178                    goto err;
2179            }

2181            /*
2182             * Now process the PT_NOTE structures.  Each one is preceded by
2183             * an Elf{32/64}_Nhdr structure describing its type and size.
2184             *
2185             *  +--------+
2186             *  | header |
2187             *  +--------+
2188             *  | name   |
2189             *  | ...    |
2190             *  +--------+
2191             *  | desc   |
2192             *  | ...    |
2193             *  +--------+
2194             */
2195            for (nleft = note_phdr.p_filesz; nleft > 0; ) {
2196                    Elf64_Nhdr nhdr;
2197                    off64_t off, namesz;

2199                    /*
2200                     * Although <sys/elf.h> defines both Elf32_Nhdr and Elf64_Nhdr
2201                     * as different types, they are both of the same content and
2202                     * size, so we don't need to worry about 32/64 conversion here.
2203                     */
2204                    if (read(P->asfd, &nhdr, sizeof (nhdr)) != sizeof (nhdr)) {
2205                            dprintf("Pgrab_core: failed to read ELF note header\n");
2206                            *perr = G_NOTE;
2207                            goto err;
2208                    }

2210                    /*
```



```
2211                     * According to the System V ABI, the amount of padding
2212                     * following the name field should align the description
2213                     * field on a 4 byte boundary for 32-bit binaries or on an 8
2214                     * byte boundary for 64-bit binaries. However, this change
2215                     * was not made correctly during the 64-bit port so all
2216                     * descriptions can assume only 4-byte alignment. We ignore
2217                     * the name field and the padding to 4-byte alignment.
2218                     */
2219                    namesz = P2ROUNDUP((off64_t)nhdr.n_namesz, (off64_t)4);
2220                    if (lseek64(P->asfd, namesz, SEEK_CUR) == (off64_t)-1) {
2221                            dprintf("failed to seek past name and padding\n");
2222                            *perr = G_STRANGE;
2223                            goto err;
2224                    }

2226                    dprintf("Note hdr n_type=%u n_namesz=%u n_descsz=%u\n",
2227                        nhdr.n_type, nhdr.n_namesz, nhdr.n_descsz);

2229                    off = lseek64(P->asfd, (off64_t)0L, SEEK_CUR);

2231                    /*
2232                     * Invoke the note handler function from our table
2233                     */
2234                    if (nhdr.n_type < sizeof (nhdlrs) / sizeof (nhdlrs[0])) {
2235                            if (nhdlrs[nhdr.n_type](P, nhdr.n_descsz) < 0) {
2236                                    *perr = G_NOTE;
2237                                    goto err;
2238                            }
2239                    } else
2240                            (void) note_notsup(P, nhdr.n_descsz);

2242                    /*
2243                     * Seek past the current note data to the next Elf_Nhdr
2244                     */
2245                    if (lseek64(P->asfd, off + nhdr.n_descsz,
2246                        SEEK_SET) == (off64_t)-1) {
2247                            dprintf("Pgrab_core: failed to seek to next nhdr\n");
2248                            *perr = G_STRANGE;
2249                            goto err;
2250                    }

2252                    /*
2253                     * Subtract the size of the header and its data from what
2254                     * we have left to process.
2255                     */
2256                    nleft -= sizeof (nhdr) + namesz + nhdr.n_descsz;
2257            }

2259            if (nleft != 0) {
2260                    dprintf("Pgrab_core: note section malformed\n");
2261                    *perr = G_STRANGE;
2262                    goto err;
2263            }

2265            if ((pagesize = Pgetauxval(P, AT_PAGESZ)) == -1) {
2266                    pagesize = getpagesize();
2267                    dprintf("AT_PAGESZ missing; defaulting to %d\n", pagesize);
2268            }

2270            /*
2271             * Locate and label the mappings corresponding to the end of the
2272             * heap (MA_BREAK) and the base of the stack (MA_STACK).
2273             */
2274            if ((P->status.pr_brkbase != 0 || P->status.pr_brksize != 0) &&
2275                (brk_mp = Paddr2mptr(P, P->status.pr_brkbase +
2276                    P->status.pr_brksize - 1)) != NULL)
```

```
2277                brk_mp->map_pmap.pr_mflags |= MA_BREAK;
2278        else
2279                brk_mp = NULL;

2281        if ((stk_mp = Paddr2mptr(P, P->status.pr_stkbase)) != NULL)
2282                stk_mp->map_pmap.pr_mflags |= MA_STACK;

2284        /*
2285         * At this point, we have enough information to look for the
2286         * executable and open it: we have access to the auxv, a psinfo_t,
2287         * and the ability to read from mappings provided by the core file.
2288         */
2289        (void) Pfindexec(P, aout_path, core_exec_open, &aout);
2290        dprintf("P->execname = \"%s\"\n", P->execname ? P->execname : "NULL");
2291        execname = P->execname ? P->execname : "a.out";

2293        /*
2294         * Iterate through the sections, looking for the .dynamic and .interp
2295         * sections.  If we encounter them, remember their section pointers.
2296         */
2297        for (scn = NULL; (scn = elf_nextscn(aout.e_elf, scn)) != NULL; ) {
2298                char *sname;

2300                if ((gelf_getshdr(scn, &shdr) == NULL) ||
2301                    (sname = elf_strptr(aout.e_elf, aout.e_hdr.e_shstrndx,
2302                    (size_t)shdr.sh_name)) == NULL)
2303                        continue;

2305                if (strcmp(sname, ".interp") == 0)
2306                        intp_scn = scn;
2307        }

2309        /*
2310         * Get the AT_BASE auxv element.  If this is missing (-1), then
2311         * we assume this is a statically-linked executable.
2312         */
2313        base_addr = Pgetauxval(P, AT_BASE);

2315        /*
2316         * In order to get librtld_db initialized, we'll need to identify
2317         * and name the mapping corresponding to the run-time linker.  The
2318         * AT_BASE auxv element tells us the address where it was mapped,
2319         * and the .interp section of the executable tells us its path.
2320         * If for some reason that doesn't pan out, just use ld.so.1.
2321         */
2322        if (intp_scn != NULL && (dp = elf_getdata(intp_scn, NULL)) != NULL &&
2323            dp->d_size != 0) {
2324                dprintf(".interp = <%s>\n", (char *)dp->d_buf);
2325                interp = dp->d_buf;

2327        } else if (base_addr != (uintptr_t)-1L) {
2328                if (core_info->core_dmodel == PR_MODEL_LP64)
2117                if (P->core->core_dmodel == PR_MODEL_LP64)
2329                        interp = "/usr/lib/64/ld.so.1";
2330                else
2331                        interp = "/usr/lib/ld.so.1";

2333                dprintf(".interp section is missing or could not be read; "
2334                    "defaulting to %s\n", interp);
2335        } else
2336                dprintf("detected statically linked executable\n");

2338        /*
2339         * If we have an AT_BASE element, name the mapping at that address
2340         * using the interpreter pathname.  Name the corresponding data
2341         * mapping after the interpreter as well.
```



```
2342         */
2343        if (base_addr != (uintptr_t)-1L) {
2344                elf_file_t intf;

2346                P->map_ldso = core_name_mapping(P, base_addr, interp);

2348                if (core_elf_open(&intf, interp, ET_DYN, NULL) == 0) {
2349                        rd_loadobj_t rl;
2350                        map_info_t *dmp;

2352                        rl.rl_base = base_addr;
2353                        dmp = core_find_data(P, intf.e_elf, &rl);

2355                        if (dmp != NULL) {
2356                                dprintf("renamed data at %p to %s\n",
2357                                    (void *)rl.rl_data_base, interp);
2358                                (void) strncpy(dmp->map_pmap.pr_mapname,
2359                                    interp, PRMAPSZ);
2360                                dmp->map_pmap.pr_mapname[PRMAPSZ - 1] = '\0';
2361                        }
2362                }

2364                core_elf_close(&intf);
2365        }

2367        /*
2368         * If we have an AT_ENTRY element, name the mapping at that address
2369         * using the special name "a.out" just like /proc does.
2370         */
2371        if ((addr = Pgetauxval(P, AT_ENTRY)) != (uintptr_t)-1L)
2372                P->map_exec = core_name_mapping(P, addr, "a.out");

2374        /*
2375         * If we're a statically linked executable, then just locate the
2376         * executable's text and data and name them after the executable.
2377         */
2378        if (base_addr == (uintptr_t)-1L) {
2379                map_info_t *tmp, *dmp;
2380                file_info_t *fp;
2381                rd_loadobj_t rl;

2383                if ((tmp = core_find_text(P, aout.e_elf, &rl)) != NULL &&
2384                    (dmp = core_find_data(P, aout.e_elf, &rl)) != NULL) {
2385                        (void) strncpy(tmp->map_pmap.pr_mapname,
2386                            execname, PRMAPSZ);
2387                        tmp->map_pmap.pr_mapname[PRMAPSZ - 1] = '\0';
2388                        (void) strncpy(dmp->map_pmap.pr_mapname,
2389                            execname, PRMAPSZ);
2390                        dmp->map_pmap.pr_mapname[PRMAPSZ - 1] = '\0';
2391                }

2393                if ((P->map_exec = tmp) != NULL &&
2394                    (fp = malloc(sizeof (file_info_t))) != NULL) {

2396                        (void) memset(fp, 0, sizeof (file_info_t));

2398                        list_link(fp, &P->file_head);
2399                        tmp->map_file = fp;
2400                        P->num_files++;

2402                        fp->file_ref = 1;
2403                        fp->file_fd = -1;

2405                        fp->file_lo = malloc(sizeof (rd_loadobj_t));
2406                        fp->file_lname = strdup(execname);
```

```
2408                        if (fp->file_lo)
2409                                *fp->file_lo = rl;
2410                        if (fp->file_lname)
2411                                fp->file_lbase = basename(fp->file_lname);
2412                        if (fp->file_rname)
2413                                fp->file_rbase = basename(fp->file_rname);

2415                        (void) strcpy(fp->file_pname,
2416                            P->mappings[0].map_pmap.pr_mapname);
2417                        fp->file_map = tmp;

2419                        Pbuild_file_symtab(P, fp);

2421                        if (dmp != NULL) {
2422                                dmp->map_file = fp;
2423                                fp->file_ref++;
2424                        }
2425                }
2426        }

2428        core_elf_close(&aout);

2430        /*
2431         * We now have enough information to initialize librtld_db.
2432         * After it warms up, we can iterate through the load object chain
2433         * in the core, which will allow us to construct the file info
2434         * we need to provide symbol information for the other shared
2435         * libraries, and also to fill in the missing mapping names.
2436         */
2437        rd_log(_libproc_debug);

2439        if ((P->rap = rd_new(P)) != NULL) {
2440                (void) rd_loadobj_iter(P->rap, (rl_iter_f *)
2441                    core_iter_mapping, P);

2443                if (core_info->core_errno != 0) {
2444                        errno = core_info->core_errno;
2232                if (P->core->core_errno != 0) {
2233                        errno = P->core->core_errno;
2445                        *perr = G_STRANGE;
2446                        goto err;
2447                }
2448        } else
2449                dprintf("failed to initialize rtld_db agent\n");

2451        /*
2452         * If there are sections, load them and process the data from any
2453         * sections that we can use to annotate the file_info_t's.
2454         */
2455        core_load_shdrs(P, &core);

2457        /*
2458         * If we previously located a stack or break mapping, and they are
2459         * still anonymous, we now assume that they were MAP_ANON mappings.
2460         * If brk_mp turns out to now have a name, then the heap is still
2461         * sitting at the end of the executable's data+bss mapping: remove
2462         * the previous MA_BREAK setting to be consistent with /proc.
2463         */
2464        if (stk_mp != NULL && stk_mp->map_pmap.pr_mapname[0] == '\0')
2465                stk_mp->map_pmap.pr_mflags |= MA_ANON;
2466        if (brk_mp != NULL && brk_mp->map_pmap.pr_mapname[0] == '\0')
2467                brk_mp->map_pmap.pr_mflags |= MA_ANON;
2468        else if (brk_mp != NULL)
2469                brk_mp->map_pmap.pr_mflags &= ~MA_BREAK;

2471        *perr = 0;
```



```
2472        return (P);

2474 err:
2475        Pfree(P);
2476        core_elf_close(&aout);
2477        return (NULL);
2478 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    7357 Wed Aug 21 14:46:26 2013
new/usr/src/lib/libproc/common/Pexecname.c
3946 ::gcore
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Matthew Ahrens <mahrens@delphix.com>
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2008 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */
/*
 * Copyright (c) 2013 by Delphix. All rights reserved.
 */

#define __EXTENSIONS__
#include <string.h>
#undef  __EXTENSIONS__

#include <libgen.h>
#include <limits.h>
#include <stdio.h>
#include <errno.h>
#include <unistd.h>
#include <zone.h>

#include "libproc.h"
#include "Pcontrol.h"

/*
 * Pexecname.c - Way too much code to attempt to derive the full pathname of
 * the executable file from a process handle, be it dead or alive.
 */

/*
 * Once we've computed a cwd and a relative path, we use try_exec() to
 * form an absolute path, call resolvepath() on it, and then let the
 * caller's function do the final confirmation.
 */
static int
try_exec(struct ps_prochandle *P, const char *cwd, const char *path, char *buf,
    int (*isexec)(const char *, void *), void *isdata)
{
        int i;

        if (path[0] != '/')
```



```
                (void) snprintf(buf, PATH_MAX, "%s/%s", cwd, path);
        else
                (void) strcpy(buf, path);

        dprintf("try_exec \"%s\"\n", buf);

        (void) Pfindobj(P, buf, buf, PATH_MAX);
        if ((i = resolvepath(buf, buf, PATH_MAX)) > 0) {
                buf[i] = '\0';
                return (isexec(buf, isdata));
        }

        return (0); /* resolvepath failed */
}
```

*unchanged_portion_omitted_*

```
/*
 * Return the full pathname for the executable file.
 * Callback function for Pfindexec().  We return a match if we can stat the
 * suggested pathname and confirm its device and inode number match our
 * previous information about the /proc/<pid>/object/a.out file.
 */
static int
stat_exec(const char *path, struct stat64 *stp)
{
        struct stat64 st;

        return (stat64(path, &st) == 0 && S_ISREG(st.st_mode) &&
            stp->st_dev == st.st_dev && stp->st_ino == st.st_ino);
}

/*
 * Return the full pathname for the executable file.  If the process handle is
 * a core file, we've already tried our best to get the executable name.
 * Otherwise, we make an attempt using Pfindexec().
 */
char *
Pexecname(struct ps_prochandle *P, char *buf, size_t buflen)
{
        if (P->execname != NULL) {
                (void) strncpy(buf, P->execname, buflen);
                return (buf);
        }

        return (P->ops.pop_execname(P, buf, buflen, P->data));
        if (P->state != PS_DEAD && P->state != PS_IDLE) {
                char exec_name[PATH_MAX];
                char cwd[PATH_MAX];
                char proc_cwd[64];
                struct stat64 st;
                int ret;

                /*
                 * Try to get the path information first.
                 */
                (void) snprintf(exec_name, sizeof (exec_name),
                    "%s/%d/path/a.out", procfs_path, (int)P->pid);
                if ((ret = readlink(exec_name, buf, buflen - 1)) > 0) {
                        buf[ret] = '\0';
                        (void) Pfindobj(P, buf, buf, buflen);
                        return (buf);
                }

                /*
                 * Stat the executable file so we can compare Pfindexec's
                 * suggestions to the actual device and inode number.
```

```
286                  */
287                 (void) snprintf(exec_name, sizeof (exec_name),
288                     "%s/%d/object/a.out", procfs_path, (int)P->pid);

290                 if (stat64(exec_name, &st) != 0 || !S_ISREG(st.st_mode))
291                         return (NULL);

293                 /*
294                  * Attempt to figure out the current working directory of the
295                  * target process.  This only works if the target process has
296                  * not changed its current directory since it was exec'd.
297                  */
298                 (void) snprintf(proc_cwd, sizeof (proc_cwd),
299                     "%s/%d/path/cwd", procfs_path, (int)P->pid);

301                 if ((ret = readlink(proc_cwd, cwd, PATH_MAX - 1)) > 0)
302                         cwd[ret] = '\0';

304                 (void) Pfindexec(P, ret > 0 ? cwd : NULL,
305                     (int (*)(const char *, void *))stat_exec, &st);
306         }

308         return (NULL);
 253 }
```

_____unchanged_portion_omitted_

```
**********************************************************
   39819 Wed Aug 21 14:46:28 2013
new/usr/src/lib/libproc/common/Pgcore.c
3946 ::gcore
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Matthew Ahrens <mahrens@delphix.com>
**********************************************************
```

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

/*
 * Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */
/*
 * Copyright 2012 DEY Storage Systems, Inc.  All rights reserved.
 * Copyright (c) 2013, Joyent, Inc. All rights reserved.
 * Copyright (c) 2013 by Delphix. All rights reserved.
 */

#define _STRUCTURED_PROC        1

#include <stdlib.h>
#include <ctype.h>
#include <string.h>
#include <strings.h>
#include <errno.h>
#include <procfs.h>
#include <priv.h>
#include <sys/elf.h>
#include <sys/machelf.h>
#include <sys/sysmacros.h>
#include <sys/systeminfo.h>
#include <sys/proc.h>
#include <sys/utsname.h>

#include <sys/old_procfs.h>

#include "Pcontrol.h"
#include "P32ton.h"

typedef enum {
        STR_NONE,
        STR_CTF,
        STR_SYMTAB,
        STR_DYNSYM,
        STR_STRTAB,
        STR_DYNSTR,
```



```
        STR_SHSTRTAB,
        STR_NUM
} shstrtype_t;
_____unchanged_portion_omitted_

/*
 * Don't explicity stop the process; that's up to the consumer.
 */
int
Pfgcore(struct ps_prochandle *P, int fd, core_content_t content)
{
        char plat[SYS_NMLN];
        char zonename[ZONENAME_MAX];
        int platlen = -1;
        pgcore_t pgc;
        off64_t poff, soff, doff, boff;
        struct utsname uts;
        uint_t nphdrs, nshdrs;

        if (ftruncate64(fd, 0) != 0)
                return (-1);

        if (content == CC_CONTENT_INVALID) {
                errno = EINVAL;
                return (-1);
        }

        /*
         * Cache the mappings and other useful data.
         */
        (void) Prd_agent(P);
        (void) Ppsinfo(P);

        pgc.P = P;
        pgc.pgc_fd = fd;
        pgc.pgc_poff = &poff;
        pgc.pgc_soff = &soff;
        pgc.pgc_doff = &doff;
        pgc.pgc_content = content;
        pgc.pgc_chunksz = PAGESIZE;
        if ((pgc.pgc_chunk = malloc(pgc.pgc_chunksz)) == NULL)
                return (-1);

        shstrtab_init(&pgc.pgc_shstrtab);

        /*
         * There are two PT_NOTE program headers for ancillary data, and
         * one for each mapping.
         */
        nphdrs = 2 + P->map_count;
        nshdrs = count_sections(&pgc);

        (void) Pplatform(P, plat, sizeof (plat));
        platlen = strlen(plat) + 1;
        Preadauxvec(P);
        (void) Puname(P, &uts);
        if (Pzonename(P, zonename, sizeof (zonename)) == NULL)
                zonename[0] = '\0';

        /*
         * The core file contents may required zero section headers, but if we
         * overflow the 16 bits allotted to the program header count in the ELF
         * header, we'll need that program header at index zero.
         */
        if (nshdrs == 0 && nphdrs >= PN_XNUM)
                nshdrs = 1;
```

```
1072         /*
1073          * Set up the ELF header.
1074          */
1075         if (P->status.pr_dmodel == PR_MODEL_ILP32) {
1076                 Elf32_Ehdr ehdr;

1078                 bzero(&ehdr, sizeof (ehdr));
1079                 ehdr.e_ident[EI_MAG0] = ELFMAG0;
1080                 ehdr.e_ident[EI_MAG1] = ELFMAG1;
1081                 ehdr.e_ident[EI_MAG2] = ELFMAG2;
1082                 ehdr.e_ident[EI_MAG3] = ELFMAG3;
1083                 ehdr.e_type = ET_CORE;

1085                 ehdr.e_ident[EI_CLASS] = ELFCLASS32;
1086 #if defined(__sparc)
1087                 ehdr.e_machine = EM_SPARC;
1088                 ehdr.e_ident[EI_DATA] = ELFDATA2MSB;
1089 #elif defined(__i386) || defined(__amd64)
1090                 ehdr.e_machine = EM_386;
1091                 ehdr.e_ident[EI_DATA] = ELFDATA2LSB;
1092 #else
1093 #error "unknown machine type"
1094 #endif
1095                 ehdr.e_ident[EI_VERSION] = EV_CURRENT;

1097                 ehdr.e_version = EV_CURRENT;
1098                 ehdr.e_ehsize = sizeof (ehdr);

1100                 if (nphdrs >= PN_XNUM)
1101                         ehdr.e_phnum = PN_XNUM;
1102                 else
1103                         ehdr.e_phnum = (unsigned short)nphdrs;

1105                 ehdr.e_phentsize = sizeof (Elf32_Phdr);
1106                 ehdr.e_phoff = ehdr.e_ehsize;

1108                 if (nshdrs > 0) {
1109                         if (nshdrs >= SHN_LORESERVE)
1110                                 ehdr.e_shnum = 0;
1111                         else
1112                                 ehdr.e_shnum = (unsigned short)nshdrs;

1114                         if (nshdrs - 1 >= SHN_LORESERVE)
1115                                 ehdr.e_shstrndx = SHN_XINDEX;
1116                         else
1117                                 ehdr.e_shstrndx = (unsigned short)(nshdrs - 1);

1119                         ehdr.e_shentsize = sizeof (Elf32_Shdr);
1120                         ehdr.e_shoff = ehdr.e_phoff + ehdr.e_phentsize * nphdrs;
1121                 }

1123                 if (pwrite64(fd, &ehdr, sizeof (ehdr), 0) != sizeof (ehdr))
1124                         goto err;

1126                 poff = ehdr.e_phoff;
1127                 soff = ehdr.e_shoff;
1128                 doff = boff = ehdr.e_ehsize +
1129                     ehdr.e_phentsize * nphdrs +
1130                     ehdr.e_shentsize * nshdrs;

1132 #ifdef _LP64
1133         } else {
1134                 Elf64_Ehdr ehdr;

1136                 bzero(&ehdr, sizeof (ehdr));
```



```
1137                 ehdr.e_ident[EI_MAG0] = ELFMAG0;
1138                 ehdr.e_ident[EI_MAG1] = ELFMAG1;
1139                 ehdr.e_ident[EI_MAG2] = ELFMAG2;
1140                 ehdr.e_ident[EI_MAG3] = ELFMAG3;
1141                 ehdr.e_type = ET_CORE;

1143                 ehdr.e_ident[EI_CLASS] = ELFCLASS64;
1144 #if defined(__sparc)
1145                 ehdr.e_machine = EM_SPARCV9;
1146                 ehdr.e_ident[EI_DATA] = ELFDATA2MSB;
1147 #elif defined(__i386) || defined(__amd64)
1148                 ehdr.e_machine = EM_AMD64;
1149                 ehdr.e_ident[EI_DATA] = ELFDATA2LSB;
1150 #else
1151 #error "unknown machine type"
1152 #endif
1153                 ehdr.e_ident[EI_VERSION] = EV_CURRENT;

1155                 ehdr.e_version = EV_CURRENT;
1156                 ehdr.e_ehsize = sizeof (ehdr);

1158                 if (nphdrs >= PN_XNUM)
1159                         ehdr.e_phnum = PN_XNUM;
1160                 else
1161                         ehdr.e_phnum = (unsigned short)nphdrs;

1163                 ehdr.e_phentsize = sizeof (Elf64_Phdr);
1164                 ehdr.e_phoff = ehdr.e_ehsize;

1166                 if (nshdrs > 0) {
1167                         if (nshdrs >= SHN_LORESERVE)
1168                                 ehdr.e_shnum = 0;
1169                         else
1170                                 ehdr.e_shnum = (unsigned short)nshdrs;

1172                         if (nshdrs - 1 >= SHN_LORESERVE)
1173                                 ehdr.e_shstrndx = SHN_XINDEX;
1174                         else
1175                                 ehdr.e_shstrndx = (unsigned short)(nshdrs - 1);

1177                         ehdr.e_shentsize = sizeof (Elf64_Shdr);
1178                         ehdr.e_shoff = ehdr.e_phoff + ehdr.e_phentsize * nphdrs;
1179                 }

1181                 if (pwrite64(fd, &ehdr, sizeof (ehdr), 0) != sizeof (ehdr))
1182                         goto err;

1184                 poff = ehdr.e_phoff;
1185                 soff = ehdr.e_shoff;
1186                 doff = boff = ehdr.e_ehsize +
1187                     ehdr.e_phentsize * nphdrs +
1188                     ehdr.e_shentsize * nshdrs;

1190 #endif  /* _LP64 */
1191         }

1193         /*
1194          * Write the zero indexed section if it exists.
1195          */
1196         if (nshdrs > 0 && write_shdr(&pgc, STR_NONE, 0, 0, 0, 0,
1197             nshdrs >= SHN_LORESERVE ? nshdrs : 0,
1198             nshdrs - 1 >= SHN_LORESERVE ? nshdrs - 1 : 0,
1199             nphdrs >= PN_XNUM ? nphdrs : 0, 0, 0) != 0)
1200                 goto err;

1202         /*
```

```
1203          * Construct the old-style note header and section.
1204          */

1206         if (P->status.pr_dmodel == PR_MODEL_NATIVE) {
1207                 prpsinfo_t prpsinfo;

1209                 mkprpsinfo(P, &prpsinfo);
1210                 if (write_note(fd, NT_PRPSINFO, &prpsinfo, sizeof (prpsinfo_t),
1211                     &doff) != 0) {
1212                         goto err;
1213                 }
1214                 if (write_note(fd, NT_AUXV, P->auxv,
1215                     P->nauxv * sizeof (P->auxv[0]), &doff) != 0) {
1216                         goto err;
1217                 }
1218 #ifdef _LP64
1219         } else {
1220                 prpsinfo32_t pi32;
1221                 auxv32_t *av32;
1222                 size_t size = sizeof (auxv32_t) * P->nauxv;
1223                 int i;

1225                 mkprpsinfo32(P, &pi32);
1226                 if (write_note(fd, NT_PRPSINFO, &pi32, sizeof (prpsinfo32_t),
1227                     &doff) != 0) {
1228                         goto err;
1229                 }

1231                 if ((av32 = malloc(size)) == NULL)
1232                         goto err;

1234                 for (i = 0; i < P->nauxv; i++) {
1235                         auxv_n_to_32(&P->auxv[i], &av32[i]);
1236                 }

1238                 if (write_note(fd, NT_AUXV, av32, size, &doff) != 0) {
1239                         free(av32);
1240                         goto err;
1241                 }

1243                 free(av32);
1244 #endif  /* _LP64 */
1245         }

1247         if (write_note(fd, NT_PLATFORM, plat, platlen, &doff) != 0)
1248                 goto err;

1250         if (Plwp_iter_all(P, old_per_lwp, &pgc) != 0)
1251                 goto err;

1253         if (P->status.pr_dmodel == PR_MODEL_ILP32) {
1254                 Elf32_Phdr phdr;

1256                 bzero(&phdr, sizeof (phdr));
1257                 phdr.p_type = PT_NOTE;
1258                 phdr.p_flags = PF_R;
1259                 phdr.p_offset = (Elf32_Off)boff;
1260                 phdr.p_filesz = doff - boff;
1261                 boff = doff;

1263                 if (pwrite64(fd, &phdr, sizeof (phdr), poff) != sizeof (phdr))
1264                         goto err;
1265                 poff += sizeof (phdr);
1266 #ifdef _LP64
1267         } else {
1268                 Elf64_Phdr phdr;
```



```
1270                 bzero(&phdr, sizeof (phdr));
1271                 phdr.p_type = PT_NOTE;
1272                 phdr.p_flags = PF_R;
1273                 phdr.p_offset = boff;
1274                 phdr.p_filesz = doff - boff;
1275                 boff = doff;

1277                 if (pwrite64(fd, &phdr, sizeof (phdr), poff) != sizeof (phdr))
1278                         goto err;
1279                 poff += sizeof (phdr);
1280 #endif  /* _LP64 */
1281         }

1283         /*
1284          * Construct the new-style note header and section.
1285          */

1287         if (P->status.pr_dmodel == PR_MODEL_NATIVE) {
1288                 if (write_note(fd, NT_PSINFO, &P->psinfo, sizeof (psinfo_t),
1289                     &doff) != 0) {
1290                         goto err;
1291                 }
1292                 if (write_note(fd, NT_PSTATUS, &P->status, sizeof (pstatus_t),
1293                     &doff) != 0) {
1294                         goto err;
1295                 }
1296                 if (write_note(fd, NT_AUXV, P->auxv,
1297                     P->nauxv * sizeof (P->auxv[0]), &doff) != 0) {
1298                         goto err;
1299                 }
1300 #ifdef _LP64
1301         } else {
1302                 psinfo32_t pi32;
1303                 pstatus32_t ps32;
1304                 auxv32_t *av32;
1305                 size_t size = sizeof (auxv32_t) * P->nauxv;
1306                 int i;

1308                 psinfo_n_to_32(&P->psinfo, &pi32);
1309                 if (write_note(fd, NT_PSINFO, &pi32, sizeof (psinfo32_t),
1310                     &doff) != 0) {
1311                         goto err;
1312                 }
1313                 pstatus_n_to_32(&P->status, &ps32);
1314                 if (write_note(fd, NT_PSTATUS, &ps32, sizeof (pstatus32_t),
1315                     &doff) != 0) {
1316                         goto err;
1317                 }
1318                 if ((av32 = malloc(size)) == NULL)
1319                         goto err;

1321                 for (i = 0; i < P->nauxv; i++) {
1322                         auxv_n_to_32(&P->auxv[i], &av32[i]);
1323                 }

1325                 if (write_note(fd, NT_AUXV, av32, size, &doff) != 0) {
1326                         free(av32);
1327                         goto err;
1328                 }

1330                 free(av32);
1331 #endif  /* _LP64 */
1332         }

1334         if (write_note(fd, NT_PLATFORM, plat, platlen, &doff) != 0 ||
```

```
1335             write_note(fd, NT_UTSNAME, &uts, sizeof (uts), &doff) != 0 ||
1336             write_note(fd, NT_CONTENT, &content, sizeof (content), &doff) != 0)
1337                 goto err;

1339         {
1340                 prcred_t cred, *cp;
1341                 size_t size = sizeof (prcred_t);

1343                 if (Pcred(P, &cred, 0) != 0)
1344                         goto err;

1346                 if (cred.pr_ngroups > 0)
1347                         size += sizeof (gid_t) * (cred.pr_ngroups - 1);
1348                 if ((cp = malloc(size)) == NULL)
1349                         goto err;

1351                 if (Pcred(P, cp, cred.pr_ngroups) != 0 ||
1352                     write_note(fd, NT_PRCRED, cp, size, &doff) != 0) {
1353                         free(cp);
1354                         goto err;
1355                 }

1357                 free(cp);
1358         }

1360         {
1361                 prpriv_t *ppriv = NULL;
1360                 prpriv_t *ppriv;
1362                 const priv_impl_info_t *pinfo;
1363                 size_t pprivsz, pinfosz;

1365                 if (Ppriv(P, &ppriv) == -1)
1364                 if ((ppriv = proc_get_priv(P->pid)) == NULL)
1366                         goto err;
1367                 pprivsz = PRIV_PRPRIV_SIZE(ppriv);

1369                 if (write_note(fd, NT_PRPRIV, ppriv, pprivsz, &doff) != 0) {
1370                         free(ppriv);
1371                         goto err;
1372                 }
1373                 free(ppriv);

1375                 if ((pinfo = getprivimplinfo()) == NULL)
1376                         goto err;
1377                 pinfosz = PRIV_IMPL_INFO_SIZE(pinfo);

1379                 if (write_note(fd, NT_PRPRIVINFO, pinfo, pinfosz, &doff) != 0)
1380                         goto err;
1381         }

1383         if (write_note(fd, NT_ZONENAME, zonename, strlen(zonename) + 1,
1384             &doff) != 0)
1385                 goto err;

1387         {
1388                 fditer_t iter;
1389                 iter.fd_fd = fd;
1390                 iter.fd_doff = &doff;

1392                 if (Pfdinfo_iter(P, iter_fd, &iter) != 0)
1393                         goto err;
1394         }

1396 #if defined(__i386) || defined(__amd64)
1397         /* CSTYLED */
1398         {
```



```
1399                 struct ssd *ldtp;
1400                 size_t size;
1401                 int nldt;

1403                 /*
1404                  * Only dump out non-zero sized LDT notes.
1405                  */
1406                 if ((nldt = Pldt(P, NULL, 0)) != 0) {
1407                         size = sizeof (struct ssd) * nldt;
1408                         if ((ldtp = malloc(size)) == NULL)
1409                                 goto err;

1411                         if (Pldt(P, ldtp, nldt) == -1 ||
1412                             write_note(fd, NT_LDT, ldtp, size, &doff) != 0) {
1413                                 free(ldtp);
1414                                 goto err;
1415                         }

1417                         free(ldtp);
1418                 }
1419         }
1420 #endif  /* __i386 || __amd64 */

1422         if (Plwp_iter_all(P, new_per_lwp, &pgc) != 0)
1423                 goto err;

1425         if (P->status.pr_dmodel == PR_MODEL_ILP32) {
1426                 Elf32_Phdr phdr;

1428                 bzero(&phdr, sizeof (phdr));
1429                 phdr.p_type = PT_NOTE;
1430                 phdr.p_flags = PF_R;
1431                 phdr.p_offset = (Elf32_Off)boff;
1432                 phdr.p_filesz = doff - boff;
1433                 boff = doff;

1435                 if (pwrite64(fd, &phdr, sizeof (phdr), poff) != sizeof (phdr))
1436                         goto err;
1437                 poff += sizeof (phdr);
1438 #ifdef _LP64
1439         } else {
1440                 Elf64_Phdr phdr;

1442                 bzero(&phdr, sizeof (phdr));
1443                 phdr.p_type = PT_NOTE;
1444                 phdr.p_flags = PF_R;
1445                 phdr.p_offset = boff;
1446                 phdr.p_filesz = doff - boff;
1447                 boff = doff;

1449                 if (pwrite64(fd, &phdr, sizeof (phdr), poff) != sizeof (phdr))
1450                         goto err;
1451                 poff += sizeof (phdr);
1452 #endif  /* _LP64 */
1453         }

1455         /*
1456          * Construct the headers for each mapping and write out its data
1457          * if the content parameter indicates that it should be present
1458          * in the core file.
1459          */
1460         if (Pmapping_iter(P, dump_map, &pgc) != 0)
1461                 goto err;

1463         if (dump_sections(&pgc) != 0)
1464                 goto err;
```

```
1466         if (write_shstrtab(P, &pgc) != 0)
1467                 goto err;

1469         free(pgc.pgc_chunk);

1471         return (0);

1473 err:
1474         /*
1475          * Wipe out anything we may have written if there was an error.
1476          */
1477         (void) ftruncate64(fd, 0);
1478         free(pgc.pgc_chunk);
1479         return (-1);
1480 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    7160 Wed Aug 21 14:46:29 2013
new/usr/src/lib/libproc/common/Pidle.c
3946 ::gcore
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Matthew Ahrens <mahrens@delphix.com>
**********************************************************
```

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

#include <stdlib.h>
#include <libelf.h>
#include <libgen.h>
#include <string.h>
#include <strings.h>
#include <errno.h>
#include <sys/sysmacros.h>

#include "libproc.h"
#include "Pcontrol.h"

/*ARGSUSED*/
static ssize_t
Pread_idle(struct ps_prochandle *P, void *buf, size_t n, uintptr_t addr,
    void *data)
Pread_idle(struct ps_prochandle *P, void *buf, size_t n, uintptr_t addr)
{
        size_t resid = n;

        while (resid > 0) {
                map_info_t *mp;
                uintptr_t mapoff;
                ssize_t len;
                off64_t off;

                if ((mp = Paddr2mptr(P, addr)) == NULL)
                        break;

                mapoff = addr - mp->map_pmap.pr_vaddr;
                len = MIN(resid, mp->map_pmap.pr_size - mapoff);
                off = mp->map_offset + mapoff;
```



```
                if ((len = pread64(P->asfd, buf, len, off)) <= 0)
                        break;

                resid -= len;
                addr += len;
                buf = (char *)buf + len;
        }

        return (n - resid);
}

/*ARGSUSED*/
static ssize_t
Pwrite_idle(struct ps_prochandle *P, const void *buf, size_t n, uintptr_t addr,
    void *data)
Pwrite_idle(struct ps_prochandle *P, const void *buf, size_t n, uintptr_t addr)
{
        errno = EIO;
        return (-1);
}

/*ARGSUSED*/
static int
Ppriv_idle(struct ps_prochandle *P, prpriv_t **pprv, void *data)
{
        prpriv_t *pp;

        pp = proc_get_priv(P->pid);
        if (pp == NULL) {
                return (-1);
        }

        *pprv = pp;
        return (0);
}

/* Default operations for the idl ops vector. */
static void *
Pidle_voidp()
{
        errno = ENODATA;
        return (NULL);
}

static int
Pidle_int()
{
        errno = ENODATA;
        return (-1);
}

static const ps_ops_t P_idle_ops = {
        .pop_pread      = Pread_idle,
        .pop_pwrite     = Pwrite_idle,
        .pop_cred       = (pop_cred_t)Pidle_int,
        .pop_priv       = Ppriv_idle,
        .pop_psinfo     = (pop_psinfo_t)Pidle_voidp,
        .pop_platform   = (pop_platform_t)Pidle_voidp,
        .pop_uname      = (pop_uname_t)Pidle_int,
        .pop_zonename   = (pop_zonename_t)Pidle_voidp,
#if defined(__i386) || defined(__amd64)
        .pop_ldt        = (pop_ldt_t)Pidle_int
#endif
static const ps_rwops_t P_idle_ops = {
        Pread_idle,
```

```
 76          Pwrite_idle
122 };
_______unchanged_portion_omitted_

153 struct ps_prochandle *
154 Pgrab_file(const char *fname, int *perr)
155 {
156         struct ps_prochandle *P = NULL;
157         char buf[PATH_MAX];
158         GElf_Ehdr ehdr;
159         Elf *elf = NULL;
160         size_t phnum;
161         file_info_t *fp = NULL;
162         int fd;
163         int i;

165         if ((fd = open64(fname, O_RDONLY)) < 0) {
166                 dprintf("couldn't open file");
167                 *perr = (errno == ENOENT) ? G_NOEXEC : G_STRANGE;
168                 return (NULL);
169         }

171         if (elf_version(EV_CURRENT) == EV_NONE) {
172                 dprintf("libproc ELF version is more recent than libelf");
173                 *perr = G_ELF;
174                 goto err;
175         }

177         if ((P = calloc(1, sizeof (struct ps_prochandle))) == NULL) {
178                 *perr = G_STRANGE;
179                 goto err;
180         }

182         (void) mutex_init(&P->proc_lock, USYNC_THREAD, NULL);
183         P->state = PS_IDLE;
184         P->pid = (pid_t)-1;
185         P->asfd = fd;
186         P->ctlfd = -1;
187         P->statfd = -1;
188         P->agentctlfd = -1;
189         P->agentstatfd = -1;
190         P->info_valid = -1;
191         Pinit_ops(&P->ops, &P_idle_ops);
146         P->ops = &P_idle_ops;
192         Pinitsym(P);

194         if ((elf = elf_begin(fd, ELF_C_READ, NULL)) == NULL) {
195                 *perr = G_ELF;
196                 return (NULL);
197         }

199         /*
200          * Construct a file_info_t that corresponds to this file.
201          */
202         if ((fp = calloc(1, sizeof (file_info_t))) == NULL) {
203                 *perr = G_STRANGE;
204                 goto err;
205         }

207         if ((fp->file_lo = calloc(1, sizeof (rd_loadobj_t))) == NULL) {
208                 *perr = G_STRANGE;
209                 goto err;
210         }

212         if (*fname == '/') {
213                 (void) strncpy(fp->file_pname, fname, sizeof (fp->file_pname));
```



```
214         } else {
215                 size_t sz;

217                 if (getcwd(fp->file_pname, sizeof (fp->file_pname) - 1) ==
218                     NULL) {
219                         *perr = G_STRANGE;
220                         goto err;
221                 }

223                 sz = strlen(fp->file_pname);
224                 (void) snprintf(&fp->file_pname[sz],
225                     sizeof (fp->file_pname) - sz, "/%s", fname);
226         }

228         fp->file_fd = fd;
229         fp->file_lo->rl_lmident = LM_ID_BASE;
230         if ((fp->file_lname = strdup(fp->file_pname)) == NULL) {
231                 *perr = G_STRANGE;
232                 goto err;
233         }
234         fp->file_lbase = basename(fp->file_lname);

236         if ((P->execname = strdup(fp->file_pname)) == NULL) {
237                 *perr = G_STRANGE;
238                 goto err;
239         }

241         P->num_files++;
242         list_link(fp, &P->file_head);

244         if (gelf_getehdr(elf, &ehdr) == NULL) {
245                 *perr = G_STRANGE;
246                 goto err;
247         }

249         if (elf_getphdrnum(elf, &phnum) == -1) {
250                 *perr = G_STRANGE;
251                 goto err;
252         }

254         dprintf("Pgrab_file: program header count = %lu\n", (ulong_t)phnum);

256         /*
257          * Sift through the program headers making the relevant maps.
258          */
259         for (i = 0; i < phnum; i++) {
260                 GElf_Phdr phdr, *php;

262                 if ((php = gelf_getphdr(elf, i, &phdr)) == NULL) {
263                         *perr = G_STRANGE;
264                         goto err;
265                 }

267                 if (php->p_type != PT_LOAD)
268                         continue;

270                 if (idle_add_mapping(P, php, fp) != 0) {
271                         *perr = G_STRANGE;
272                         goto err;
273                 }
274         }
275         Psort_mappings(P);

277         (void) elf_end(elf);

279         P->map_exec = fp->file_map;
```

```
281         P->status.pr_flags = PR_STOPPED;
282         P->status.pr_nlwp = 0;
283         P->status.pr_pid = (pid_t)-1;
284         P->status.pr_ppid = (pid_t)-1;
285         P->status.pr_pgid = (pid_t)-1;
286         P->status.pr_sid = (pid_t)-1;
287         P->status.pr_taskid = (taskid_t)-1;
288         P->status.pr_projid = (projid_t)-1;
289         P->status.pr_zoneid = (zoneid_t)-1;
290         switch (ehdr.e_ident[EI_CLASS]) {
291         case ELFCLASS32:
292                 P->status.pr_dmodel = PR_MODEL_ILP32;
293                 break;
294         case ELFCLASS64:
295                 P->status.pr_dmodel = PR_MODEL_LP64;
296                 break;
297         default:
298                 *perr = G_FORMAT;
299                 goto err;
300         }

302         /*
303          * Pfindobj() checks what zone a process is associated with, so
304          * we call it after initializing pr_zoneid to -1.  This ensures
305          * we don't get associated with any zone on the system.
306          */
307         if (Pfindobj(P, fp->file_lname, buf, sizeof (buf)) != NULL) {
308                 free(P->execname);
309                 P->execname = strdup(buf);
310                 if ((fp->file_rname = strdup(buf)) != NULL)
311                         fp->file_rbase = basename(fp->file_rname);
312         }

314         /*
315          * The file and map lists are complete, and will never need to be
316          * adjusted.
317          */
318         P->info_valid = 1;

320         return (P);
321 err:
322         (void) close(fd);
323         if (P != NULL)
324                 Pfree(P);
325         if (elf != NULL)
326                 (void) elf_end(elf);
327         return (NULL);
328 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   11620 Wed Aug 21 14:46:30 2013
new/usr/src/lib/libproc/common/Plwpregs.c
3946 ::gcore
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Matthew Ahrens <mahrens@delphix.com>
**********************************************************
```

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

/*
 * Copyright (c) 2013, Joyent, Inc. All rights reserved.
 * Copyright (c) 2013 by Delphix. All rights reserved.
 */

#include <sys/types.h>
#include <sys/uio.h>
#include <string.h>
#include <errno.h>
#include <limits.h>

#include "Pcontrol.h"
#include "P32ton.h"

/*
 * This file implements the routines to read and write per-lwp register
 * information from either a live process or core file opened with libproc.
 * We build up a few common routines for reading and writing register
 * information, and then the public functions are all trivial calls to these.
 */

/*
 * Utility function to return a pointer to the structure of cached information
 * about an lwp in the core file, given its lwpid.
 */
static lwp_info_t *
getlwpcore(struct ps_prochandle *P, lwpid_t lwpid)
{
        core_info_t *core = P->data;
        lwp_info_t *lwp = list_next(&core->core_lwp_head);
        lwp_info_t *lwp = list_next(&P->core->core_lwp_head);
        uint_t i;

        for (i = 0; i < core->core_nlwp; i++, lwp = list_next(lwp)) {
```



```
        for (i = 0; i < P->core->core_nlwp; i++, lwp = list_next(lwp)) {
                if (lwp->lwp_id == lwpid)
                        return (lwp);
        }

        errno = EINVAL;
        return (NULL);
}
_____unchanged_portion_omitted_

/*
 * Get the lwpstatus_t for an lwp from either the live process or our
 * cached information from the core file.  This is used to get the
 * general-purpose registers or floating point registers.
 */
int
getlwpstatus(struct ps_prochandle *P, lwpid_t lwpid, lwpstatus_t *lps)
{
        lwp_info_t *lwp;

        /*
         * For both live processes and cores, our job is easy if the lwpid
         * matches that of the representative lwp:
         */
        if (P->status.pr_lwp.pr_lwpid == lwpid) {
                (void) memcpy(lps, &P->status.pr_lwp, sizeof (lwpstatus_t));
                return (0);
        }

        /*
         * If this is a live process, then just read the information out
         * of the per-lwp status file:
         */
        if (P->state != PS_DEAD) {
                return (getlwpfile(P, lwpid, "lwpstatus",
                    lps, sizeof (lwpstatus_t)));
        }

        /*
         * If this is a core file, we need to iterate through our list of
         * cached lwp information and then copy out the status.
         */
        if (P->data != NULL && (lwp = getlwpcore(P, lwpid)) != NULL) {
        if (P->core != NULL && (lwp = getlwpcore(P, lwpid)) != NULL) {
                (void) memcpy(lps, &lwp->lwp_status, sizeof (lwpstatus_t));
                return (0);
        }

        return (-1);
}
_____unchanged_portion_omitted_
```

```
**********************************************************
    9105 Wed Aug 21 14:46:31 2013
new/usr/src/lib/libproc/common/Pservice.c
3946 ::gcore
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Matthew Ahrens <mahrens@delphix.com>
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */
/*
 * Copyright (c) 2013 by Delphix. All rights reserved.
 */

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

#include <stdarg.h>
#include <string.h>
#include "Pcontrol.h"

/*
 * This file implements the process services declared in <proc_service.h>.
 * This enables libproc to be used in conjunction with libc_db and
 * librtld_db.  As most of these facilities are already provided by
 * (more elegant) interfaces in <libproc.h>, we can just call those.
 *
 * NOTE: We explicitly do *not* implement the functions ps_kill() and
 * ps_lrolltoaddr() in this library.  The very existence of these functions
 * causes libc_db to create an "agent thread" in the target process.
 * The only way to turn off this behavior is to omit these functions.
 */

#pragma weak ps_pdread = ps_pread
#pragma weak ps_ptread = ps_pread
#pragma weak ps_pdwrite = ps_pwrite
#pragma weak ps_ptwrite = ps_pwrite

ps_err_e
ps_pdmodel(struct ps_prochandle *P, int *modelp)
{
        *modelp = P->status.pr_dmodel;
        return (PS_OK);
}

ps_err_e
```



```
ps_pread(struct ps_prochandle *P, psaddr_t addr, void *buf, size_t size)
{
        if (P->ops.pop_pread(P, buf, size, addr, P->data) != size)
        if (P->ops->p_pread(P, buf, size, addr) != size)
                return (PS_BADADDR);
        return (PS_OK);
}

ps_err_e
ps_pwrite(struct ps_prochandle *P, psaddr_t addr, const void *buf, size_t size)
{
        if (P->ops.pop_pwrite(P, buf, size, addr, P->data) != size)
        if (P->ops->p_pwrite(P, buf, size, addr) != size)
                return (PS_BADADDR);
        return (PS_OK);
}
______unchanged_portion_omitted_
```

```
**********************************************************
   81016 Wed Aug 21 14:46:32 2013
new/usr/src/lib/libproc/common/Psymtab.c
3946 ::gcore
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Matthew Ahrens <mahrens@delphix.com>
**********************************************************
```

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

/*
 * Copyright (c) 1997, 2010, Oracle and/or its affiliates. All rights reserved.
 * Copyright (c) 2013, Joyent, Inc. All rights reserved.
 * Copyright (c) 2013 by Delphix. All rights reserved.
 */

#include <assert.h>
#include <stdio.h>
#include <stdlib.h>
#include <stddef.h>
#include <unistd.h>
#include <ctype.h>
#include <fcntl.h>
#include <string.h>
#include <strings.h>
#include <memory.h>
#include <errno.h>
#include <dirent.h>
#include <signal.h>
#include <limits.h>
#include <libgen.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/systeminfo.h>
#include <sys/sysmacros.h>

#include "libproc.h"
#include "Pcontrol.h"
#include "Putil.h"
#include "Psymtab_machelf.h"

static file_info_t *build_map_symtab(struct ps_prochandle *, map_info_t *);
static map_info_t *exec_map(struct ps_prochandle *);
static map_info_t *object_to_map(struct ps_prochandle *, Lmid_t, const char *);
static map_info_t *object_name_to_map(struct ps_prochandle *,
        Lmid_t, const char *);
static GElf_Sym *sym_by_name(sym_tbl_t *, const char *, GElf_Sym *, uint_t *);
static int read_ehdr32(struct ps_prochandle *, Elf32_Ehdr *, uint_t *,
```



```
    uintptr_t);
#ifdef _LP64
static int read_ehdr64(struct ps_prochandle *, Elf64_Ehdr *, uint_t *,
    uintptr_t);
#endif

#define DATA_TYPES      \
        ((1 << STT_OBJECT) | (1 << STT_FUNC) | \
        (1 << STT_COMMON) | (1 << STT_TLS))
#define IS_DATA_TYPE(tp)        (((1 << (tp)) & DATA_TYPES) != 0)

#define MA_RWX  (MA_READ | MA_WRITE | MA_EXEC)

typedef enum {
        PRO_NATURAL,
        PRO_BYADDR,
        PRO_BYNAME
} pr_order_t;
```

*unchanged_portion_omitted_*

```
int
Preadmaps(struct ps_prochandle *P, prmap_t **Pmapp, ssize_t *nmapp)
{
        return (P->ops.pop_read_maps(P, Pmapp, nmapp, P->data));
}

/*
 * Go through all the address space mappings, validating or updating
 * the information already gathered, or gathering new information.
 *
 * This function is only called when we suspect that the mappings have changed
 * because this is the first time we're calling it or because of rtld activity.
 */
void
Pupdate_maps(struct ps_prochandle *P)
{
        char mapfile[PATH_MAX];
        int mapfd;
        struct stat statb;
        prmap_t *Pmap = NULL;
        prmap_t *pmap;
        ssize_t nmap;
        int i;
        uint_t oldmapcount;
        map_info_t *newmap, *newp;
        map_info_t *mptr;

        if (P->info_valid || P->state == PS_UNDEAD)
                return;

        Preadauxvec(P);

        if (Preadmaps(P, &Pmap, &nmap) != 0)
        (void) snprintf(mapfile, sizeof (mapfile), "%s/%d/map",
            procfs_path, (int)P->pid);
        if ((mapfd = open(mapfile, O_RDONLY)) < 0 ||
            fstat(mapfd, &statb) != 0 ||
            statb.st_size < sizeof (prmap_t) ||
            (Pmap = malloc(statb.st_size)) == NULL ||
            (nmap = pread(mapfd, Pmap, statb.st_size, 0L)) <= 0 ||
            (nmap /= sizeof (prmap_t)) == 0) {
                if (Pmap != NULL)
                        free(Pmap);
                if (mapfd >= 0)
                        (void) close(mapfd);
                Preset_maps(P); /* utter failure; destroy tables */
```

```
467                 return;
477         }
478         (void) close(mapfd);

469         if ((newmap = calloc(1, nmap * sizeof (map_info_t))) == NULL)
470                 return;

472         /*
473          * We try to merge any file information we may have for existing
474          * mappings, to avoid having to rebuild the file info.
475          */
476         mptr = P->mappings;
477         pmap = Pmap;
478         newp = newmap;
479         oldmapcount = P->map_count;
480         for (i = 0; i < nmap; i++, pmap++, newp++) {

482                 if (oldmapcount == 0) {
483                         /*
484                          * We've exhausted all the old mappings.  Every new
485                          * mapping should be added.
486                          */
487                         newp->map_pmap = *pmap;

489                 } else if (pmap->pr_vaddr == mptr->map_pmap.pr_vaddr &&
490                     pmap->pr_size == mptr->map_pmap.pr_size &&
491                     pmap->pr_offset == mptr->map_pmap.pr_offset &&
492                     (pmap->pr_mflags & ~(MA_BREAK | MA_STACK)) ==
493                     (mptr->map_pmap.pr_mflags & ~(MA_BREAK | MA_STACK)) &&
494                     pmap->pr_pagesize == mptr->map_pmap.pr_pagesize &&
495                     pmap->pr_shmid == mptr->map_pmap.pr_shmid &&
496                     strcmp(pmap->pr_mapname, mptr->map_pmap.pr_mapname) == 0) {

498                         /*
499                          * This mapping matches exactly.  Copy over the old
500                          * mapping, taking care to get the latest flags.
501                          * Make sure the associated file_info_t is updated
502                          * appropriately.
503                          */
504                         *newp = *mptr;
505                         if (P->map_exec == mptr)
506                                 P->map_exec = newp;
507                         if (P->map_ldso == mptr)
508                                 P->map_ldso = newp;
509                         newp->map_pmap.pr_mflags = pmap->pr_mflags;
510                         if (mptr->map_file != NULL &&
511                             mptr->map_file->file_map == mptr)
512                                 mptr->map_file->file_map = newp;
513                         oldmapcount--;
514                         mptr++;

516                 } else if (pmap->pr_vaddr + pmap->pr_size >
517                     mptr->map_pmap.pr_vaddr) {

519                         /*
520                          * The old mapping doesn't exist any more, remove it
521                          * from the list.
522                          */
523                         map_info_free(P, mptr);
524                         oldmapcount--;
525                         i--;
526                         newp--;
527                         pmap--;
528                         mptr++;

530                 } else {
```



```
532                         /*
533                          * This is a new mapping, add it directly.
534                          */
535                         newp->map_pmap = *pmap;
536                 }
537         }

539         /*
540          * Free any old maps
541          */
542         while (oldmapcount) {
543                 map_info_free(P, mptr);
544                 oldmapcount--;
545                 mptr++;
546         }

548         free(Pmap);
549         if (P->mappings != NULL)
550                 free(P->mappings);
551         P->mappings = newmap;
552         P->map_count = P->map_alloc = nmap;
553         P->info_valid = 1;

555         /*
556          * Consult librtld_db to get the load object
557          * names for all of the shared libraries.
558          */
559         if (P->rap != NULL)
560                 (void) rd_loadobj_iter(P->rap, map_iter, P);
561 }
```

_____***unchanged_portion_omitted_***

```
851 /*
852  * If we're not a core file, re-read the /proc/<pid>/auxv file and store
853  * its contents in P->auxv.  In the case of a core file, we either
854  * initialized P->auxv in Pcore() from the NT_AUXV, or we don't have an
855  * auxv because the note was missing.
856  */
840 void
841 Preadauxvec(struct ps_prochandle *P)
842 {
860         char auxfile[64];
861         struct stat statb;
862         ssize_t naux;
863         int fd;

865         if (P->state == PS_DEAD)
866                 return; /* Already read during Pgrab_core() */
867         if (P->state == PS_IDLE)
868                 return; /* No aux vec for Pgrab_file() */

843         if (P->auxv != NULL) {
844                 free(P->auxv);
845                 P->auxv = NULL;
846                 P->nauxv = 0;
847         }

849         P->ops.pop_read_aux(P, &P->auxv, &P->nauxv, P->data);
876         (void) snprintf(auxfile, sizeof (auxfile), "%s/%d/auxv",
877             procfs_path, (int)P->pid);
878         if ((fd = open(auxfile, O_RDONLY)) < 0)
879                 return;

881         if (fstat(fd, &statb) == 0 &&
882             statb.st_size >= sizeof (auxv_t) &&
```

```
 883             (P->auxv = malloc(statb.st_size + sizeof (auxv_t))) != NULL) {
 884                 if ((naux = read(fd, P->auxv, statb.st_size)) < 0 ||
 885                     (naux /= sizeof (auxv_t)) < 1) {
 886                         free(P->auxv);
 887                         P->auxv = NULL;
 888                 } else {
 889                         P->auxv[naux].a_type = AT_NULL;
 890                         P->auxv[naux].a_un.a_val = 0L;
 891                         P->nauxv = (int)naux;
 892                 }
 893         }

 895         (void) close(fd);
 850 }
```

_____unchanged_portion_omitted_

```
1569 /*
1570  * Build the symbol table for the given mapped file.
1571  */
1572 void
1573 Pbuild_file_symtab(struct ps_prochandle *P, file_info_t *fptr)
1574 {
1575         char objectfile[PATH_MAX];
1576         uint_t i;

1578         GElf_Ehdr ehdr;
1579         GElf_Sym s;

1581         Elf_Data *shdata;
1582         Elf_Scn *scn;
1583         Elf *elf;
1584         size_t nshdrs, shstrndx;

1586         struct {
1587                 GElf_Shdr c_shdr;
1588                 Elf_Data *c_data;
1589                 const char *c_name;
1590         } *cp, *cache = NULL, *dyn = NULL, *plt = NULL, *ctf = NULL;

1592         if (fptr->file_init)
1593                 return; /* We've already processed this file */

1595         /*
1596          * Mark the file_info struct as having the symbol table initialized
1597          * even if we fail below.  We tried once; we don't try again.
1598          */
1599         fptr->file_init = 1;

1601         if (elf_version(EV_CURRENT) == EV_NONE) {
1602                 dprintf("libproc ELF version is more recent than libelf\n");
1603                 return;
1604         }

1606         if (P->state == PS_DEAD || P->state == PS_IDLE) {
1607                 char *name;
1608                 /*
1609                  * If we're a not live, we can't open files from the /proc
1610                  * object directory; we have only the mapping and file names
1611                  * to guide us.  We prefer the file_lname, but need to handle
1612                  * the case of it being NULL in order to bootstrap: we first
1613                  * come here during rd_new() when the only information we have
1614                  * is interpreter name associated with the AT_BASE mapping.
1615                  *
1616                  * Also, if the zone associated with the core file seems
1617                  * to exists on this machine we'll try to open the object
1618                  * file within the zone.
```



```
1619                  */
1620                 if (fptr->file_rname != NULL)
1621                         name = fptr->file_rname;
1622                 else if (fptr->file_lname != NULL)
1623                         name = fptr->file_lname;
1624                 else
1625                         name = fptr->file_pname;
1626                 (void) strlcpy(objectfile, name, sizeof (objectfile));
1627         } else {
1628                 (void) snprintf(objectfile, sizeof (objectfile),
1629                     "%s/%d/object/%s",
1630                     procfs_path, (int)P->pid, fptr->file_pname);
1631         }

1633         /*
1634          * Open the object file, create the elf file, and then get the elf
1635          * header and .shstrtab data buffer so we can process sections by
1636          * name. If anything goes wrong try to fake up an elf file from
1637          * the in-core elf image.
1638          */

1640         if (_libproc_incore_elf || (P->flags & INCORE)) {
1686         if (_libproc_incore_elf) {
1641                 dprintf("Pbuild_file_symtab: using in-core data for: %s\n",
1642                     fptr->file_pname);

1644                 if ((elf = build_fake_elf(P, fptr, &ehdr, &nshdrs, &shdata)) ==
1645                     NULL)
1646                         return;

1648         } else if ((fptr->file_fd = open(objectfile, O_RDONLY)) < 0) {
1649                 dprintf("Pbuild_file_symtab: failed to open %s: %s\n",
1650                     objectfile, strerror(errno));

1652                 if ((elf = build_fake_elf(P, fptr, &ehdr, &nshdrs, &shdata)) ==
1653                     NULL)
1654                         return;

1656         } else if ((elf = elf_begin(fptr->file_fd, ELF_C_READ, NULL)) == NULL ||
1657             elf_kind(elf) != ELF_K_ELF ||
1658             gelf_getehdr(elf, &ehdr) == NULL ||
1659             elf_getshdrnum(elf, &nshdrs) == -1 ||
1660             elf_getshdrstrndx(elf, &shstrndx) == -1 ||
1661             (scn = elf_getscn(elf, shstrndx)) == NULL ||
1662             (shdata = elf_getdata(scn, NULL)) == NULL) {
1663                 int err = elf_errno();

1665                 dprintf("failed to process ELF file %s: %s\n",
1666                     objectfile, (err == 0) ? "<null>" : elf_errmsg(err));
1667                 (void) elf_end(elf);

1669                 if ((elf = build_fake_elf(P, fptr, &ehdr, &nshdrs, &shdata)) ==
1670                     NULL)
1671                         return;

1673         } else if (file_differs(P, elf, fptr)) {
1674                 Elf *newelf;

1676                 /*
1677                  * Before we get too excited about this elf file, we'll check
1678                  * its checksum value against the value we have in memory. If
1679                  * they don't agree, we try to fake up a new elf file and
1680                  * proceed with that instead.
1681                  */
1682                 dprintf("ELF file %s (%lx) doesn't match in-core image\n",
1683                     fptr->file_pname,
```

```
1684                    (ulong_t)fptr->file_map->map_pmap.pr_vaddr);

1686                if ((newelf = build_fake_elf(P, fptr, &ehdr, &nshdrs, &shdata))
1687                    != NULL) {
1688                    (void) elf_end(elf);
1689                    elf = newelf;
1690                    dprintf("switched to faked up ELF file\n");

1692                    /*
1693                     * Check to see if the file that we just discovered
1694                     * to be an imposter matches the execname that was
1695                     * determined by Pfindexec().  If it does, we (clearly)
1696                     * don't have the right binary, and we zero out
1697                     * execname before anyone gets hurt.
1698                     */
1699                    if (fptr->file_rname != NULL && P->execname != NULL &&
1700                        strcmp(fptr->file_rname, P->execname) == 0) {
1701                            dprintf("file/in-core image mismatch was "
1702                                "on P->execname; discarding\n");
1703                            free(P->execname);
1704                            P->execname = NULL;
1705                    }
1706                }
1707        }

1709        if ((cache = malloc(nshdrs * sizeof (*cache))) == NULL) {
1710                dprintf("failed to malloc section cache for %s\n", objectfile);
1711                goto bad;
1712        }

1714        dprintf("processing ELF file %s\n", objectfile);
1715        fptr->file_class = ehdr.e_ident[EI_CLASS];
1716        fptr->file_etype = ehdr.e_type;
1717        fptr->file_elf = elf;
1718        fptr->file_shstrs = shdata->d_buf;
1719        fptr->file_shstrsz = shdata->d_size;

1721        /*
1722         * Iterate through each section, caching its section header, data
1723         * pointer, and name.  We use this for handling sh_link values below.
1724         */
1725        for (cp = cache + 1, scn = NULL; scn = elf_nextscn(elf, scn); cp++) {
1726                if (gelf_getshdr(scn, &cp->c_shdr) == NULL) {
1727                        dprintf("Pbuild_file_symtab: Failed to get section "
1728                            "header\n");
1729                        goto bad; /* Failed to get section header */
1730                }

1732                if ((cp->c_data = elf_getdata(scn, NULL)) == NULL) {
1733                        dprintf("Pbuild_file_symtab: Failed to get section "
1734                            "data\n");
1735                        goto bad; /* Failed to get section data */
1736                }

1738                if (cp->c_shdr.sh_name >= shdata->d_size) {
1739                        dprintf("Pbuild_file_symtab: corrupt section name");
1740                        goto bad; /* Corrupt section name */
1741                }

1743                cp->c_name = (const char *)shdata->d_buf + cp->c_shdr.sh_name;
1744        }

1746        /*
1747         * Now iterate through the section cache in order to locate info
1748         * for the .symtab, .dynsym, .SUNW_ldynsym, .dynamic, .plt,
1749         * and .SUNW_ctf sections:
```



```
1750         */
1751        for (i = 1, cp = cache + 1; i < nshdrs; i++, cp++) {
1752                GElf_Shdr *shp = &cp->c_shdr;

1754                if (shp->sh_type == SHT_SYMTAB || shp->sh_type == SHT_DYNSYM) {
1755                        sym_tbl_t *symp = shp->sh_type == SHT_SYMTAB ?
1756                            &fptr->file_symtab : &fptr->file_dynsym;
1757                        /*
1758                         * It's possible that the we already got the symbol
1759                         * table from the core file itself. Either the file
1760                         * differs in which case our faked up elf file will
1761                         * only contain the dynsym (not the symtab) or the
1762                         * file matches in which case we'll just be replacing
1763                         * the symbol table we pulled out of the core file
1764                         * with an equivalent one. In either case, this
1765                         * check isn't essential, but it's a good idea.
1766                         */
1767                        if (symp->sym_data_pri == NULL) {
1768                                dprintf("Symbol table found for %s\n",
1769                                    objectfile);
1770                                symp->sym_data_pri = cp->c_data;
1771                                symp->sym_symn +=
1772                                    shp->sh_size / shp->sh_entsize;
1773                                symp->sym_strs =
1774                                    cache[shp->sh_link].c_data->d_buf;
1775                                symp->sym_strsz =
1776                                    cache[shp->sh_link].c_data->d_size;
1777                                symp->sym_hdr_pri = cp->c_shdr;
1778                                symp->sym_strhdr = cache[shp->sh_link].c_shdr;
1779                        } else {
1780                                dprintf("Symbol table already there for %s\n",
1781                                    objectfile);
1782                        }
1783                } else if (shp->sh_type == SHT_SUNW_LDYNSYM) {
1784                        /* .SUNW_ldynsym section is auxiliary to .dynsym */
1785                        if (fptr->file_dynsym.sym_data_aux == NULL) {
1786                                dprintf(".SUNW_ldynsym symbol table"
1787                                    " found for %s\n", objectfile);
1788                                fptr->file_dynsym.sym_data_aux = cp->c_data;
1789                                fptr->file_dynsym.sym_symn_aux =
1790                                    shp->sh_size / shp->sh_entsize;
1791                                fptr->file_dynsym.sym_symn +=
1792                                    fptr->file_dynsym.sym_symn_aux;
1793                                fptr->file_dynsym.sym_hdr_aux = cp->c_shdr;
1794                        } else {
1795                                dprintf(".SUNW_ldynsym symbol table already"
1796                                    " there for %s\n", objectfile);
1797                        }
1798                } else if (shp->sh_type == SHT_DYNAMIC) {
1799                        dyn = cp;
1800                } else if (strcmp(cp->c_name, ".plt") == 0) {
1801                        plt = cp;
1802                } else if (strcmp(cp->c_name, ".SUNW_ctf") == 0) {
1803                        /*
1804                         * Skip over bogus CTF sections so they don't come back
1805                         * to haunt us later.
1806                         */
1807                        if (shp->sh_link == 0 ||
1808                            shp->sh_link >= nshdrs ||
1809                            (cache[shp->sh_link].c_shdr.sh_type != SHT_DYNSYM &&
1810                            cache[shp->sh_link].c_shdr.sh_type != SHT_SYMTAB)) {
1811                                dprintf("Bad sh_link %d for "
1812                                    "CTF\n", shp->sh_link);
1813                                continue;
1814                        }
1815                        ctf = cp;
```

```
1816                }
1817        }

1819        /*
1820         * At this point, we've found all the symbol tables we're ever going
1821         * to find: the ones in the loop above and possibly the symtab that
1822         * was included in the core file. Before we perform any lookups, we
1823         * create sorted versions to optimize for lookups.
1824         */
1825        optimize_symtab(&fptr->file_symtab);
1826        optimize_symtab(&fptr->file_dynsym);

1828        /*
1829         * Fill in the base address of the text mapping for shared libraries.
1830         * This allows us to translate symbols before librtld_db is ready.
1831         */
1832        if (fptr->file_etype == ET_DYN) {
1833                fptr->file_dyn_base = fptr->file_map->map_pmap.pr_vaddr -
1834                    fptr->file_map->map_pmap.pr_offset;
1835                dprintf("setting file_dyn_base for %s to %lx\n",
1836                    objectfile, (long)fptr->file_dyn_base);
1837        }

1839        /*
1840         * Record the CTF section information in the file info structure.
1841         */
1842        if (ctf != NULL) {
1843                fptr->file_ctf_off = ctf->c_shdr.sh_offset;
1844                fptr->file_ctf_size = ctf->c_shdr.sh_size;
1845                if (ctf->c_shdr.sh_link != 0 &&
1846                    cache[ctf->c_shdr.sh_link].c_shdr.sh_type == SHT_DYNSYM)
1847                        fptr->file_ctf_dyn = 1;
1848        }

1850        if (fptr->file_lo == NULL)
1851                goto done; /* Nothing else to do if no load object info */

1853        /*
1854         * If the object is a shared library and we have a different rl_base
1855         * value, reset file_dyn_base according to librtld_db's information.
1856         */
1857        if (fptr->file_etype == ET_DYN &&
1858            fptr->file_lo->rl_base != fptr->file_dyn_base) {
1859                dprintf("resetting file_dyn_base for %s to %lx\n",
1860                    objectfile, (long)fptr->file_lo->rl_base);
1861                fptr->file_dyn_base = fptr->file_lo->rl_base;
1862        }

1864        /*
1865         * Fill in the PLT information for this file if a PLT symbol is found.
1866         */
1867        if (sym_by_name(&fptr->file_dynsym, "_PROCEDURE_LINKAGE_TABLE_", &s,
1868            NULL) != NULL) {
1869                fptr->file_plt_base = s.st_value + fptr->file_dyn_base;
1870                fptr->file_plt_size = (plt != NULL) ? plt->c_shdr.sh_size : 0;

1872                /*
1873                 * Bring the load object up to date; it is the only way the
1874                 * user has to access the PLT data. The PLT information in the
1875                 * rd_loadobj_t is not set in the call to map_iter() (the
1876                 * callback for rd_loadobj_iter) where we set file_lo.
1877                 */
1878                fptr->file_lo->rl_plt_base = fptr->file_plt_base;
1879                fptr->file_lo->rl_plt_size = fptr->file_plt_size;

1881                dprintf("PLT found at %p, size = %lu\n",
```



```
1882                    (void *)fptr->file_plt_base, (ulong_t)fptr->file_plt_size);
1883        }

1885        /*
1886         * Fill in the PLT information.
1887         */
1888        if (dyn != NULL) {
1889                uintptr_t dynaddr = dyn->c_shdr.sh_addr + fptr->file_dyn_base;
1890                size_t ndyn = dyn->c_shdr.sh_size / dyn->c_shdr.sh_entsize;
1891                GElf_Dyn d;

1893                for (i = 0; i < ndyn; i++) {
1894                        if (gelf_getdyn(dyn->c_data, i, &d) == NULL)
1895                                continue;

1897                        switch (d.d_tag) {
1898                        case DT_JMPREL:
1899                                dprintf("DT_JMPREL is %p\n",
1900                                    (void *)(uintptr_t)d.d_un.d_ptr);
1901                                fptr->file_jmp_rel =
1902                                    d.d_un.d_ptr + fptr->file_dyn_base;
1903                                break;
1904                        case DT_STRTAB:
1905                                dprintf("DT_STRTAB is %p\n",
1906                                    (void *)(uintptr_t)d.d_un.d_ptr);
1907                                break;
1908                        case DT_PLTGOT:
1909                                dprintf("DT_PLTGOT is %p\n",
1910                                    (void *)(uintptr_t)d.d_un.d_ptr);
1911                                break;
1912                        case DT_SUNW_SYMTAB:
1913                                dprintf("DT_SUNW_SYMTAB is %p\n",
1914                                    (void *)(uintptr_t)d.d_un.d_ptr);
1915                                break;
1916                        case DT_SYMTAB:
1917                                dprintf("DT_SYMTAB is %p\n",
1918                                    (void *)(uintptr_t)d.d_un.d_ptr);
1919                                break;
1920                        case DT_HASH:
1921                                dprintf("DT_HASH is %p\n",
1922                                    (void *)(uintptr_t)d.d_un.d_ptr);
1923                                break;
1924                        }
1925                }

1927                dprintf("_DYNAMIC found at %p, %lu entries, DT_JMPREL = %p\n",
1928                    (void *)dynaddr, (ulong_t)ndyn, (void *)fptr->file_jmp_rel);
1929        }

1931 done:
1932        free(cache);
1933        return;

1935 bad:
1936        if (cache != NULL)
1937                free(cache);

1939        (void) elf_end(elf);
1940        fptr->file_elf = NULL;
1941        if (fptr->file_elfmem != NULL) {
1942                free(fptr->file_elfmem);
1943                fptr->file_elfmem = NULL;
1944        }
1945        (void) close(fptr->file_fd);
1946        fptr->file_fd = -1;
1947 }
```

_____***unchanged_portion_omitted_***

```
2925 /*
2926  * Get the platform string.
2972  * Get the platform string from the core file if we have it;
2973  * just perform the system call for the caller if this is a live process.
2927  */
2928 char *
2929 Pplatform(struct ps_prochandle *P, char *s, size_t n)
2930 {
2931         return (P->ops.pop_platform(P, s, n, P->data));
2978         if (P->state == PS_IDLE) {
2979                 errno = ENODATA;
2980                 return (NULL);
2981         }

2983         if (P->state == PS_DEAD) {
2984                 if (P->core->core_platform == NULL) {
2985                         errno = ENODATA;
2986                         return (NULL);
2987                 }
2988                 (void) strncpy(s, P->core->core_platform, n - 1);
2989                 s[n - 1] = '\0';

2991         } else if (sysinfo(SI_PLATFORM, s, n) == -1)
2992                 return (NULL);

2994         return (s);
2932 }

2934 /*
2935  * Get the uname(2) information.
2998  * Get the uname(2) information from the core file if we have it;
2999  * just perform the system call for the caller if this is a live process.
2936  */
2937 int
2938 Puname(struct ps_prochandle *P, struct utsname *u)
2939 {
2940         return (P->ops.pop_uname(P, u, P->data));
3004         if (P->state == PS_IDLE) {
3005                 errno = ENODATA;
3006                 return (-1);
3007         }

3009         if (P->state == PS_DEAD) {
3010                 if (P->core->core_uts == NULL) {
3011                         errno = ENODATA;
3012                         return (-1);
3013                 }
3014                 (void) memcpy(u, P->core->core_uts, sizeof (struct utsname));
3015                 return (0);
3016         }
3017         return (uname(u));
2941 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    6400 Wed Aug 21 14:46:34 2013
new/usr/src/lib/libproc/common/Putil.c
3946 ::gcore
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Matthew Ahrens <mahrens@delphix.com>
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License, Version 1.0 only
 * (the "License").  You may not use this file except in compliance
 * with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */
/*
 * Copyright (c) 2013 by Delphix. All rights reserved.
 */

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

#include <limits.h>
#include <string.h>
#include <stdarg.h>
#include <stdio.h>
#include <errno.h>

#include "Pcontrol.h"
#include "Putil.h"

/*
 * Place the new element on the list prior to the existing element.
 */
void
list_link(void *new, void *existing)
{
        plist_t *p = new;
        plist_t *q = existing;

        if (q) {
                p->list_forw = q;
                p->list_back = q->list_back;
                q->list_back->list_forw = p;
                q->list_back = p;
        } else {
                p->list_forw = p->list_back = p;
        }
}
_____unchanged_portion_omitted_
```



```
/*
 * Printf-style error reporting function.  This is used to supplement the error
 * return codes from various libproc functions with additional text.  Since we
 * are a library, and should not be spewing messages to stderr, we provide a
 * default version of this function that does nothing, but by calling this
 * function we allow the client program to define its own version of the
 * function that will interpose on our empty default.  This may be useful for
 * clients that wish to display such messages to the user.
 */
/*ARGSUSED*/
/*PRINTFLIKE2*/
void
Perror_printf(struct ps_prochandle *P, const char *format, ...)
{
        /* nothing to do here */
}

/*
 * Default operations.
 */
static ssize_t
Pdefault_ssizet()
{
        return (-1);
}

static int
Pdefault_int()
{
        return (-1);
}

static void
Pdefault_void()
{
}

static void *
Pdefault_voidp()
{
        return (NULL);
}

static const ps_ops_t P_default_ops = {
        .pop_pread      = (pop_pread_t)Pdefault_ssizet,
        .pop_pwrite     = (pop_pwrite_t)Pdefault_ssizet,
        .pop_read_maps  = (pop_read_maps_t)Pdefault_int,
        .pop_read_aux   = (pop_read_aux_t)Pdefault_void,
        .pop_cred       = (pop_cred_t)Pdefault_int,
        .pop_priv       = (pop_priv_t)Pdefault_int,
        .pop_psinfo     = (pop_psinfo_t)Pdefault_voidp,
        .pop_status     = (pop_status_t)Pdefault_void,
        .pop_lstatus    = (pop_lstatus_t)Pdefault_voidp,
        .pop_lpsinfo    = (pop_lpsinfo_t)Pdefault_voidp,
        .pop_fini       = (pop_fini_t)Pdefault_void,
        .pop_platform   = (pop_platform_t)Pdefault_voidp,
        .pop_uname      = (pop_uname_t)Pdefault_int,
        .pop_zonename   = (pop_zonename_t)Pdefault_voidp,
        .pop_execname   = (pop_execname_t)Pdefault_voidp,
#if defined(__i386) || defined(__amd64)
        .pop_ldt        = (pop_ldt_t)Pdefault_int
#endif
};

/*
```

```
 * Initialize the destination ops vector with functions from the source.
 * Functions which are NULL in the source ops vector are set to corresponding
 * default function in the destination vector.
 */
void
Pinit_ops(ps_ops_t *dst, const ps_ops_t *src)
{
        *dst = P_default_ops;

        if (src->pop_pread != NULL)
                dst->pop_pread = src->pop_pread;
        if (src->pop_pwrite != NULL)
                dst->pop_pwrite = src->pop_pwrite;
        if (src->pop_read_maps != NULL)
                dst->pop_read_maps = src->pop_read_maps;
        if (src->pop_read_aux != NULL)
                dst->pop_read_aux = src->pop_read_aux;
        if (src->pop_cred != NULL)
                dst->pop_cred = src->pop_cred;
        if (src->pop_priv != NULL)
                dst->pop_priv = src->pop_priv;
        if (src->pop_psinfo != NULL)
                dst->pop_psinfo = src->pop_psinfo;
        if (src->pop_status != NULL)
                dst->pop_status = src->pop_status;
        if (src->pop_lstatus != NULL)
                dst->pop_lstatus = src->pop_lstatus;
        if (src->pop_lpsinfo != NULL)
                dst->pop_lpsinfo = src->pop_lpsinfo;
        if (src->pop_fini != NULL)
                dst->pop_fini = src->pop_fini;
        if (src->pop_platform != NULL)
                dst->pop_platform = src->pop_platform;
        if (src->pop_uname != NULL)
                dst->pop_uname = src->pop_uname;
        if (src->pop_zonename != NULL)
                dst->pop_zonename = src->pop_zonename;
        if (src->pop_execname != NULL)
                dst->pop_execname = src->pop_execname;
#if defined(__i386) || defined(__amd64)
        if (src->pop_ldt != NULL)
                dst->pop_ldt = src->pop_ldt;
#endif
}
```

*unchanged_portion_omitted_*

```
**********************************************************
    1948 Wed Aug 21 14:46:35 2013
new/usr/src/lib/libproc/common/Putil.h
3946 ::gcore
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Matthew Ahrens <mahrens@delphix.com>
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License, Version 1.0 only
 * (the "License").  You may not use this file except in compliance
 * with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */
/*
 * Copyright (c) 2013 by Delphix. All rights reserved.
 */

#ifndef _PUTIL_H
#define _PUTIL_H

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

#ifdef  __cplusplus
extern "C" {
#endif

/*
 * Circular doubly-linked list:
 */
typedef struct P_list {
        struct P_list   *list_forw;
        struct P_list   *list_back;
} plist_t;

/*
 * Routines to manipulate linked lists:
 */
extern void list_link(void *, void *);
extern void list_unlink(void *);

#define list_next(elem) (void *)(((plist_t *)(elem))->list_forw)
#define list_prev(elem) (void *)(((plist_t *)(elem))->list_back)

/*
 * Routines to manipulate sigset_t, fltset_t, or sysset_t.
 */
extern void prset_fill(void *, size_t);
```



```
extern void prset_empty(void *, size_t);
extern void prset_add(void *, size_t, uint_t);
extern void prset_del(void *, size_t, uint_t);
extern int prset_ismember(void *, size_t, uint_t);

/*
 * Routine to print debug messages:
 */
extern void dprintf(const char *, ...);

extern void Pinit_ops(ps_ops_t *, const ps_ops_t *);

#ifdef  __cplusplus
}
#endif
```

*unchanged_portion_omitted_*

```
**********************************************************
   23281 Wed Aug 21 14:46:36 2013
new/usr/src/lib/libproc/common/Pzone.c
3946 ::gcore
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Matthew Ahrens <mahrens@delphix.com>
**********************************************************
```

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

  22 /*
  23  * Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
  24  * Use is subject to license terms.
  25  */
  26 /*
  27  * Copyright (c) 2013, Joyent, Inc. All rights reserved.
  28  * Copyright (c) 2013 by Delphix. All rights reserved.
  29  */

  31 #include <assert.h>
  32 #include <dlfcn.h>
  33 #include <errno.h>
  34 #include <libzonecfg.h>
  35 #include <link.h>
  36 #include <string.h>
  37 #include <strings.h>
  38 #include <sys/list.h>
  39 #include <sys/types.h>
  40 #include <sys/mkdev.h>
  41 #include <sys/mman.h>
  42 #include <sys/mnttab.h>

  44 #include "Pcontrol.h"

  46 struct path_node {
  47         struct path_node        *pn_next;
  48         char                    *pn_path;
  49 };
```

```
_____unchanged_portion_omitted_
```

```
 228 /*
 229  * Get the zone name from the core file if we have it; look up the
 230  * name based on the zone id if this is a live process.
 231  */
 232 char *
 233 Pzonename(struct ps_prochandle *P, char *s, size_t n)
 234 {
 235         return (P->ops.pop_zonename(P, s, n, P->data));
```



```
 234         if (P->state == PS_IDLE) {
 235                 errno = ENODATA;
 236                 return (NULL);
 237         }

 239         if (P->state == PS_DEAD) {
 240                 if (P->core->core_zonename == NULL) {
 241                         errno = ENODATA;
 242                         return (NULL);
 243                 }
 244                 (void) strlcpy(s, P->core->core_zonename, n);
 245         } else {
 246                 if (getzonenamebyid(P->status.pr_zoneid, s, n) < 0)
 247                         return (NULL);
 248                 s[n - 1] = '\0';
 249         }
 250         return (s);
 236 }
```

```
_____unchanged_portion_omitted_
```

```
**********************************************************
   31454 Wed Aug 21 14:46:37 2013
new/usr/src/lib/libproc/common/libproc.h
3946 ::gcore
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Matthew Ahrens <mahrens@delphix.com>
**********************************************************
```

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

  22 /*
  23  * Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
  24  * Use is subject to license terms.
  25  *
  26  * Portions Copyright 2007 Chad Mynhier
  27  * Copyright 2012 DEY Storage Systems, Inc.  All rights reserved.
  28  * Copyright (c) 2013, Joyent, Inc. All rights reserved.
  29  * Copyright (c) 2013 by Delphix. All rights reserved.
  30  */

  32 /*
  33  * Interfaces available from the process control library, libproc.
  34  *
  35  * libproc provides process control functions for the /proc tools
  36  * (commands in /usr/proc/bin), /usr/bin/truss, and /usr/bin/gcore.
  37  * libproc is a private support library for these commands only.
  38  * It is _not_ a public interface, although it might become one
  39  * in the fullness of time, when the interfaces settle down.
  40  *
  41  * In the meantime, be aware that any program linked with libproc in this
  42  * release of Solaris is almost guaranteed to break in the next release.
  43  *
  44  * In short, do not use this header file or libproc for any purpose.
  45  */

  47 #ifndef _LIBPROC_H
  48 #define _LIBPROC_H

  50 #include <stdlib.h>
  51 #include <unistd.h>
  52 #include <fcntl.h>
  53 #include <nlist.h>
  54 #include <door.h>
  55 #include <gelf.h>
  56 #include <proc_service.h>
  57 #include <rtld_db.h>
  58 #include <procfs.h>
  59 #include <ucred.h>
```



```
  60 #include <rctl.h>
  61 #include <libctf.h>
  62 #include <sys/stat.h>
  63 #include <sys/statvfs.h>
  64 #include <sys/auxv.h>
  65 #include <sys/resource.h>
  66 #include <sys/socket.h>
  67 #include <sys/utsname.h>
  68 #include <sys/corectl.h>
  69 #if defined(__i386) || defined(__amd64)
  70 #include <sys/sysi86.h>
  71 #endif

  73 #ifdef  __cplusplus
  74 extern "C" {
  75 #endif

  77 /*
  78  * Opaque structure tag reference to a process control structure.
  79  * Clients of libproc cannot look inside the process control structure.
  80  * The implementation of struct ps_prochandle can change w/o affecting clients.
  81  */
  82 struct ps_prochandle;

  84 /*
  85  * Opaque structure tag reference to an lwp control structure.
  86  */
  87 struct ps_lwphandle;

  89 extern  int     _libproc_debug; /* set non-zero to enable debugging fprintfs */
  90 extern  int     _libproc_no_qsort;      /* set non-zero to inhibit sorting */
  91                                         /* of symbol tables */
  92 extern  int     _libproc_incore_elf;    /* only use in-core elf data */

  94 #if defined(__sparc)
  95 #define R_RVAL1 R_O0            /* register holding a function return value */
  96 #define R_RVAL2 R_O1            /* 32 more bits for a 64-bit return value */
  97 #endif  /* __sparc */

  99 #if defined(__amd64)
 100 #define R_PC    REG_RIP
 101 #define R_SP    REG_RSP
 102 #define R_RVAL1 REG_RAX         /* register holding a function return value */
 103 #define R_RVAL2 REG_RDX         /* 32 more bits for a 64-bit return value */
 104 #elif defined(__i386)
 105 #define R_PC    EIP
 106 #define R_SP    UESP
 107 #define R_RVAL1 EAX             /* register holding a function return value */
 108 #define R_RVAL2 EDX             /* 32 more bits for a 64-bit return value */
 109 #endif  /* __amd64 || __i386 */

 111 #define R_RVAL  R_RVAL1         /* simple function return value register */

 113 /* maximum sizes of things */
 114 #define PRMAXSIG        (32 * sizeof (sigset_t) / sizeof (uint32_t))
 115 #define PRMAXFAULT      (32 * sizeof (fltset_t) / sizeof (uint32_t))
 116 #define PRMAXSYS        (32 * sizeof (sysset_t) / sizeof (uint32_t))

 118 /* State values returned by Pstate() */
 119 #define PS_RUN          1       /* process is running */
 120 #define PS_STOP         2       /* process is stopped */
 121 #define PS_LOST         3       /* process is lost to control (EAGAIN) */
 122 #define PS_UNDEAD       4       /* process is terminated (zombie) */
 123 #define PS_DEAD         5       /* process is terminated (core file) */
 124 #define PS_IDLE         6       /* process has not been run */
```

```
126 /* Flags accepted by Pgrab() */
127 #define PGRAB_RETAIN    0x01    /* Retain tracing flags, else clear flags */
128 #define PGRAB_FORCE     0x02    /* Open the process w/o O_EXCL */
129 #define PGRAB_RDONLY    0x04    /* Open the process or core w/ O_RDONLY */
130 #define PGRAB_NOSTOP    0x08    /* Open the process but do not stop it */
131 #define PGRAB_INCORE    0x10    /* Use in-core data to build symbol tables */

133 /* Error codes from Pcreate() */
134 #define C_STRANGE       -1      /* Unanticipated error, errno is meaningful */
135 #define C_FORK          1       /* Unable to fork */
136 #define C_PERM          2       /* No permission (file set-id or unreadable) */
137 #define C_NOEXEC        3       /* Cannot execute file */
138 #define C_INTR          4       /* Interrupt received while creating */
139 #define C_LP64          5       /* Program is _LP64, self is _ILP32 */
140 #define C_NOENT         6       /* Cannot find executable file */

142 /* Error codes from Pgrab(), Pfgrab_core(), and Pgrab_core() */
143 #define G_STRANGE       -1      /* Unanticipated error, errno is meaningful */
144 #define G_NOPROC        1       /* No such process */
145 #define G_NOCORE        2       /* No such core file */
146 #define G_NOPROCORCORE  3       /* No such proc or core (for proc_arg_grab) */
147 #define G_NOEXEC        4       /* Cannot locate executable file */
148 #define G_ZOMB          5       /* Zombie process */
149 #define G_PERM          6       /* No permission */
150 #define G_BUSY          7       /* Another process has control */
151 #define G_SYS           8       /* System process */
152 #define G_SELF          9       /* Process is self */
153 #define G_INTR          10      /* Interrupt received while grabbing */
154 #define G_LP64          11      /* Process is _LP64, self is ILP32 */
155 #define G_FORMAT        12      /* File is not an ELF format core file */
156 #define G_ELF           13      /* Libelf error, elf_errno() is meaningful */
157 #define G_NOTE          14      /* Required PT_NOTE Phdr not present in core */
158 #define G_ISAINVAL      15      /* Wrong ELF machine type */
159 #define G_BADLWPS       16      /* Bad '/lwps' specification */
160 #define G_NOFD          17      /* No more file descriptors */


163 /* Flags accepted by Prelease */
164 #define PRELEASE_CLEAR  0x10    /* Clear all tracing flags */
165 #define PRELEASE_RETAIN 0x20    /* Retain final tracing flags */
166 #define PRELEASE_HANG   0x40    /* Leave the process stopped */
167 #define PRELEASE_KILL   0x80    /* Terminate the process */

169 typedef struct {        /* argument descriptor for system call (Psyscall) */
170         long    arg_value;      /* value of argument given to system call */
171         void    *arg_object;    /* pointer to object in controlling process */
172         char    arg_type;       /* AT_BYVAL, AT_BYREF */
173         char    arg_inout;      /* AI_INPUT, AI_OUTPUT, AI_INOUT */
174         ushort_t arg_size;      /* if AT_BYREF, size of object in bytes */
175 } argdes_t;

177 /* values for type */
178 #define AT_BYVAL        1
179 #define AT_BYREF        2

181 /* values for inout */
182 #define AI_INPUT        1
183 #define AI_OUTPUT       2
184 #define AI_INOUT        3

186 /* maximum number of syscall arguments */
187 #define MAXARGS         8

189 /* maximum size in bytes of a BYREF argument */
190 #define MAXARGL         (4*1024)
```



```
192 /*
193  * Ops vector definition for the Pgrab_ops().
194  */
195 typedef ssize_t (*pop_pread_t)(struct ps_prochandle *, void *, size_t,
196     uintptr_t, void *);
197 typedef ssize_t (*pop_pwrite_t)(struct ps_prochandle *, const void *, size_t,
198     uintptr_t, void *);
199 typedef int (*pop_read_maps_t)(struct ps_prochandle *, prmap_t **, ssize_t *,
200     void *);
201 typedef void (*pop_read_aux_t)(struct ps_prochandle *, auxv_t **, int *,
202     void *);
203 typedef int (*pop_cred_t)(struct ps_prochandle *, prcred_t *, int,
204     void *);
205 typedef int (*pop_priv_t)(struct ps_prochandle *, prpriv_t **, void *);
206 typedef const psinfo_t *(*pop_psinfo_t)(struct ps_prochandle *, psinfo_t *,
207     void *);
208 typedef void (*pop_status_t)(struct ps_prochandle *, pstatus_t *, void *);
209 typedef prheader_t *(*pop_lstatus_t)(struct ps_prochandle *, void *);
210 typedef prheader_t *(*pop_lpsinfo_t)(struct ps_prochandle *, void *);
211 typedef void (*pop_fini_t)(struct ps_prochandle *, void *);
212 typedef char *(*pop_platform_t)(struct ps_prochandle *, char *, size_t, void *);
213 typedef int (*pop_uname_t)(struct ps_prochandle *, struct utsname *, void *);
214 typedef char *(*pop_zonename_t)(struct ps_prochandle *, char *, size_t, void *);
215 typedef char *(*pop_execname_t)(struct ps_prochandle *, char *, size_t, void *);
216 #if defined(__i386) || defined(__amd64)
217 typedef int (*pop_ldt_t)(struct ps_prochandle *, struct ssd *, int, void *);
218 #endif

220 typedef struct ps_ops {
221         pop_pread_t             pop_pread;
222         pop_pwrite_t            pop_pwrite;
223         pop_read_maps_t         pop_read_maps;
224         pop_read_aux_t          pop_read_aux;
225         pop_cred_t              pop_cred;
226         pop_priv_t              pop_priv;
227         pop_psinfo_t            pop_psinfo;
228         pop_status_t            pop_status;
229         pop_lstatus_t           pop_lstatus;
230         pop_lpsinfo_t           pop_lpsinfo;
231         pop_fini_t              pop_fini;
232         pop_platform_t          pop_platform;
233         pop_uname_t             pop_uname;
234         pop_zonename_t          pop_zonename;
235         pop_execname_t          pop_execname;
236 #if defined(__i386) || defined(__amd64)
237         pop_ldt_t               pop_ldt;
238 #endif
239 } ps_ops_t;

241 /*
242  * Function prototypes for routines in the process control package.
243  */
244 extern struct ps_prochandle *Pcreate(const char *, char *const *,
245     int *, char *, size_t);
246 extern struct ps_prochandle *Pxcreate(const char *, char *const *,
247     char *const *, int *, char *, size_t);

249 extern const char *Pcreate_error(int);

251 extern struct ps_prochandle *Pgrab(pid_t, int, int *);
252 extern struct ps_prochandle *Pgrab_core(const char *, const char *, int, int *);
253 extern struct ps_prochandle *Pfgrab_core(int, const char *, int *);
254 extern struct ps_prochandle *Pgrab_file(const char *, int *);
255 extern struct ps_prochandle *Pgrab_ops(pid_t, void *, const ps_ops_t *, int);
256 extern const char *Pgrab_error(int);
```

```
258 extern  int     Preopen(struct ps_prochandle *);
259 extern  void    Prelease(struct ps_prochandle *, int);
260 extern  void    Pfree(struct ps_prochandle *);

262 extern  int     Pasfd(struct ps_prochandle *);
263 extern  char   *Pbrandname(struct ps_prochandle *, char *, size_t);
264 extern  int     Pctlfd(struct ps_prochandle *);
265 extern  int     Pcreate_agent(struct ps_prochandle *);
266 extern  void    Pdestroy_agent(struct ps_prochandle *);
267 extern  int     Pstopstatus(struct ps_prochandle *, long, uint_t);
268 extern  int     Pwait(struct ps_prochandle *, uint_t);
269 extern  int     Pstop(struct ps_prochandle *, uint_t);
270 extern  int     Pdstop(struct ps_prochandle *);
271 extern  int     Pstate(struct ps_prochandle *);
272 extern  const psinfo_t *Ppsinfo(struct ps_prochandle *);
273 extern  const pstatus_t *Pstatus(struct ps_prochandle *);
274 extern  int     Pcred(struct ps_prochandle *, prcred_t *, int);
275 extern  int     Psetcred(struct ps_prochandle *, const prcred_t *);
276 extern  int     Ppriv(struct ps_prochandle *, prpriv_t **);
224 extern  ssize_t Ppriv(struct ps_prochandle *, prpriv_t *, size_t);
277 extern  int     Psetpriv(struct ps_prochandle *, prpriv_t *);
278 extern  void   *Pprivinfo(struct ps_prochandle *);
279 extern  int     Psetzoneid(struct ps_prochandle *, zoneid_t);
280 extern  int     Pgetareg(struct ps_prochandle *, int, prgreg_t *);
281 extern  int     Pputareg(struct ps_prochandle *, int, prgreg_t);
282 extern  int     Psetrun(struct ps_prochandle *, int, int);
283 extern  ssize_t Pread(struct ps_prochandle *, void *, size_t, uintptr_t);
284 extern  ssize_t Pread_string(struct ps_prochandle *, char *, size_t, uintptr_t);
285 extern  ssize_t Pwrite(struct ps_prochandle *, const void *, size_t, uintptr_t);
286 extern  int     Pclearsig(struct ps_prochandle *);
287 extern  int     Pclearfault(struct ps_prochandle *);
288 extern  int     Psetbkpt(struct ps_prochandle *, uintptr_t, ulong_t *);
289 extern  int     Pdelbkpt(struct ps_prochandle *, uintptr_t, ulong_t);
290 extern  int     Pxecbkpt(struct ps_prochandle *, ulong_t);
291 extern  int     Psetwapt(struct ps_prochandle *, const prwatch_t *);
292 extern  int     Pdelwapt(struct ps_prochandle *, const prwatch_t *);
293 extern  int     Pxecwapt(struct ps_prochandle *, const prwatch_t *);
294 extern  int     Psetflags(struct ps_prochandle *, long);
295 extern  int     Punsetflags(struct ps_prochandle *, long);
296 extern  int     Psignal(struct ps_prochandle *, int, int);
297 extern  int     Pfault(struct ps_prochandle *, int, int);
298 extern  int     Psysentry(struct ps_prochandle *, int, int);
299 extern  int     Psysexit(struct ps_prochandle *, int, int);
300 extern  void    Psetsignal(struct ps_prochandle *, const sigset_t *);
301 extern  void    Psetfault(struct ps_prochandle *, const fltset_t *);
302 extern  void    Psetsysentry(struct ps_prochandle *, const sysset_t *);
303 extern  void    Psetsysexit(struct ps_prochandle *, const sysset_t *);

305 extern  void    Psync(struct ps_prochandle *);
306 extern  int     Psyscall(struct ps_prochandle *, sysret_t *,
307                          int, uint_t, argdes_t *);
308 extern  int     Pisprocdir(struct ps_prochandle *, const char *);

310 /*
311  * Function prototypes for lwp-specific operations.
312  */
313 extern  struct ps_lwphandle *Lgrab(struct ps_prochandle *, lwpid_t, int *);
314 extern  const char *Lgrab_error(int);

316 extern  struct ps_prochandle *Lprochandle(struct ps_lwphandle *);
317 extern  void    Lfree(struct ps_lwphandle *);

319 extern  int     Lctlfd(struct ps_lwphandle *);
320 extern  int     Lwait(struct ps_lwphandle *, uint_t);
321 extern  int     Lstop(struct ps_lwphandle *, uint_t);
322 extern  int     Ldstop(struct ps_lwphandle *);
```



```
323 extern  int     Lstate(struct ps_lwphandle *);
324 extern  const lwpsinfo_t *Lpsinfo(struct ps_lwphandle *);
325 extern  const lwpstatus_t *Lstatus(struct ps_lwphandle *);
326 extern  int     Lgetareg(struct ps_lwphandle *, int, prgreg_t *);
327 extern  int     Lputareg(struct ps_lwphandle *, int, prgreg_t);
328 extern  int     Lsetrun(struct ps_lwphandle *, int, int);
329 extern  int     Lclearsig(struct ps_lwphandle *);
330 extern  int     Lclearfault(struct ps_lwphandle *);
331 extern  int     Lxecbkpt(struct ps_lwphandle *, ulong_t);
332 extern  int     Lxecwapt(struct ps_lwphandle *, const prwatch_t *);
333 extern  void    Lsync(struct ps_lwphandle *);

335 extern  int     Lstack(struct ps_lwphandle *, stack_t *);
336 extern  int     Lmain_stack(struct ps_lwphandle *, stack_t *);
337 extern  int     Lalt_stack(struct ps_lwphandle *, stack_t *);

339 /*
340  * Function prototypes for system calls forced on the victim process.
341  */
342 extern  int     pr_open(struct ps_prochandle *, const char *, int, mode_t);
343 extern  int     pr_creat(struct ps_prochandle *, const char *, mode_t);
344 extern  int     pr_close(struct ps_prochandle *, int);
345 extern  int     pr_access(struct ps_prochandle *, const char *, int);
346 extern  int     pr_door_info(struct ps_prochandle *, int, struct door_info *);
347 extern  void    *pr_mmap(struct ps_prochandle *,
348                          void *, size_t, int, int, int, off_t);
349 extern  void    *pr_zmap(struct ps_prochandle *,
350                          void *, size_t, int, int);
351 extern  int     pr_munmap(struct ps_prochandle *, void *, size_t);
352 extern  int     pr_memcntl(struct ps_prochandle *,
353                          caddr_t, size_t, int, caddr_t, int, int);
354 extern  int     pr_meminfo(struct ps_prochandle *, const uint64_t *addrs,
355                          int addr_count, const uint_t *info, int info_count,
356                          uint64_t *outdata, uint_t *validity);
357 extern  int     pr_sigaction(struct ps_prochandle *,
358                          int, const struct sigaction *, struct sigaction *);
359 extern  int     pr_getitimer(struct ps_prochandle *,
360                          int, struct itimerval *);
361 extern  int     pr_setitimer(struct ps_prochandle *,
362                          int, const struct itimerval *, struct itimerval *);
363 extern  int     pr_ioctl(struct ps_prochandle *, int, int, void *, size_t);
364 extern  int     pr_fcntl(struct ps_prochandle *, int, int, void *);
365 extern  int     pr_stat(struct ps_prochandle *, const char *, struct stat *);
366 extern  int     pr_lstat(struct ps_prochandle *, const char *, struct stat *);
367 extern  int     pr_fstat(struct ps_prochandle *, int, struct stat *);
368 extern  int     pr_stat64(struct ps_prochandle *, const char *,
369                          struct stat64 *);
370 extern  int     pr_lstat64(struct ps_prochandle *, const char *,
371                          struct stat64 *);
372 extern  int     pr_fstat64(struct ps_prochandle *, int, struct stat64 *);
373 extern  int     pr_statvfs(struct ps_prochandle *, const char *, statvfs_t *);
374 extern  int     pr_fstatvfs(struct ps_prochandle *, int, statvfs_t *);
375 extern  projid_t pr_getprojid(struct ps_prochandle *Pr);
376 extern  taskid_t pr_gettaskid(struct ps_prochandle *Pr);
377 extern  taskid_t pr_settaskid(struct ps_prochandle *Pr, projid_t project,
378                          int flags);
379 extern  zoneid_t pr_getzoneid(struct ps_prochandle *Pr);
380 extern  int     pr_getrctl(struct ps_prochandle *,
381                          const char *, rctlblk_t *, rctlblk_t *, int);
382 extern  int     pr_setrctl(struct ps_prochandle *,
383                          const char *, rctlblk_t *, rctlblk_t *, int);
384 extern  int     pr_getrlimit(struct ps_prochandle *,
385                          int, struct rlimit *);
386 extern  int     pr_setrlimit(struct ps_prochandle *,
387                          int, const struct rlimit *);
388 extern  int     pr_setprojrctl(struct ps_prochandle *, const char *,
```

```
389                         rctlblk_t *, size_t, int);
390 #if defined(_LARGEFILE64_SOURCE)
391 extern  int     pr_getrlimit64(struct ps_prochandle *,
392                         int, struct rlimit64 *);
393 extern  int     pr_setrlimit64(struct ps_prochandle *,
394                         int, const struct rlimit64 *);
395 #endif  /* _LARGEFILE64_SOURCE */
396 extern  int     pr_lwp_exit(struct ps_prochandle *);
397 extern  int     pr_exit(struct ps_prochandle *, int);
398 extern  int     pr_waitid(struct ps_prochandle *,
399                         idtype_t, id_t, siginfo_t *, int);
400 extern  off_t   pr_lseek(struct ps_prochandle *, int, off_t, int);
401 extern  offset_t pr_llseek(struct ps_prochandle *, int, offset_t, int);
402 extern  int     pr_rename(struct ps_prochandle *, const char *, const char *);
403 extern  int     pr_link(struct ps_prochandle *, const char *, const char *);
404 extern  int     pr_unlink(struct ps_prochandle *, const char *);
405 extern  int     pr_getpeerucred(struct ps_prochandle *, int, ucred_t **);
406 extern  int     pr_getpeername(struct ps_prochandle *,
407                         int, struct sockaddr *, socklen_t *);
408 extern  int     pr_getsockname(struct ps_prochandle *,
409                         int, struct sockaddr *, socklen_t *);
410 extern  int     pr_getsockopt(struct ps_prochandle *,
411                         int, int, int, void *, int *);
412 extern  int     pr_processor_bind(struct ps_prochandle *,
413                         idtype_t, id_t, int, int *);

415 /*
416  * Function prototypes for accessing per-LWP register information.
417  */
418 extern int Plwp_getregs(struct ps_prochandle *, lwpid_t, prgregset_t);
419 extern int Plwp_setregs(struct ps_prochandle *, lwpid_t, const prgregset_t);

421 extern int Plwp_getfpregs(struct ps_prochandle *, lwpid_t, prfpregset_t *);
422 extern int Plwp_setfpregs(struct ps_prochandle *, lwpid_t,
423     const prfpregset_t *);

425 #if defined(__sparc)

427 extern int Plwp_getxregs(struct ps_prochandle *, lwpid_t, prxregset_t *);
428 extern int Plwp_setxregs(struct ps_prochandle *, lwpid_t, const prxregset_t *);

430 extern int Plwp_getgwindows(struct ps_prochandle *, lwpid_t, gwindows_t *);

432 #if defined(__sparcv9)
433 extern int Plwp_getasrs(struct ps_prochandle *, lwpid_t, asrset_t);
434 extern int Plwp_setasrs(struct ps_prochandle *, lwpid_t, const asrset_t);
435 #endif  /* __sparcv9 */

437 #endif  /* __sparc */

439 #if defined(__i386) || defined(__amd64)
440 extern  int     Pldt(struct ps_prochandle *, struct ssd *, int);
441 extern  int     proc_get_ldt(pid_t, struct ssd *, int);
442 #endif  /* __i386 || __amd64 */

444 extern int Plwp_getpsinfo(struct ps_prochandle *, lwpid_t, lwpsinfo_t *);
445 extern int Plwp_getspymaster(struct ps_prochandle *, lwpid_t, psinfo_t *);

447 extern int Plwp_stack(struct ps_prochandle *, lwpid_t, stack_t *);
448 extern int Plwp_main_stack(struct ps_prochandle *, lwpid_t, stack_t *);
449 extern int Plwp_alt_stack(struct ps_prochandle *, lwpid_t, stack_t *);

451 /*
452  * LWP iteration interface; iterate over all active LWPs.
453  */
454 typedef int proc_lwp_f(void *, const lwpstatus_t *);
```



```
455 extern int Plwp_iter(struct ps_prochandle *, proc_lwp_f *, void *);

457 /*
458  * LWP iteration interface; iterate over all LWPs, active and zombie.
459  */
460 typedef int proc_lwp_all_f(void *, const lwpstatus_t *, const lwpsinfo_t *);
461 extern int Plwp_iter_all(struct ps_prochandle *, proc_lwp_all_f *, void *);

463 /*
464  * Process iteration interface; iterate over all non-system processes.
465  */
466 typedef int proc_walk_f(psinfo_t *, lwpsinfo_t *, void *);
467 extern int proc_walk(proc_walk_f *, void *, int);

469 #define PR_WALK_PROC    0               /* walk processes only */
470 #define PR_WALK_LWP     1               /* walk all lwps */

472 /*
473  * Determine if an lwp is in a set as returned from proc_arg_xgrab().
474  */
475 extern int proc_lwp_in_set(const char *, lwpid_t);
476 extern int proc_lwp_range_valid(const char *);

478 /*
479  * Symbol table interfaces.
480  */

482 /*
483  * Pseudo-names passed to Plookup_by_name() for well-known load objects.
484  * NOTE: It is required that PR_OBJ_EXEC and PR_OBJ_LDSO exactly match
485  * the definitions of PS_OBJ_EXEC and PS_OBJ_LDSO from <proc_service.h>.
486  */
487 #define PR_OBJ_EXEC     ((const char *)0)       /* search the executable file */
488 #define PR_OBJ_LDSO     ((const char *)1)       /* search ld.so.1 */
489 #define PR_OBJ_EVERY    ((const char *)-1)      /* search every load object */

491 /*
492  * Special Lmid_t passed to Plookup_by_lmid() to search all link maps.  The
493  * special values LM_ID_BASE and LM_ID_LDSO from <link.h> may also be used.
494  * If PR_OBJ_EXEC is used as the object name, the lmid must be PR_LMID_EVERY
495  * or LM_ID_BASE in order to return a match.  If PR_OBJ_LDSO is used as the
496  * object name, the lmid must be PR_LMID_EVERY or LM_ID_LDSO to return a match.
497  */
498 #define PR_LMID_EVERY   ((Lmid_t)-1UL)          /* search every link map */

500 /*
501  * 'object_name' is the name of a load object obtained from an
502  * iteration over the process's address space mappings (Pmapping_iter),
503  * or an iteration over the process's mapped objects (Pobject_iter),
504  * or else it is one of the special PR_OBJ_* values above.
505  */
506 extern int Plookup_by_name(struct ps_prochandle *,
507     const char *, const char *, GElf_Sym *);

509 extern int Plookup_by_addr(struct ps_prochandle *,
510     uintptr_t, char *, size_t, GElf_Sym *);

512 typedef struct prsyminfo {
513         const char      *prs_object;            /* object name */
514         const char      *prs_name;              /* symbol name */
515         Lmid_t          prs_lmid;               /* link map id */
516         uint_t          prs_id;                 /* symbol id */
517         uint_t          prs_table;              /* symbol table id */
518 } prsyminfo_t;
```

unchanged_portion_omitted_

```
**********************************************************
   16074 Wed Aug 21 14:46:38 2013
new/usr/src/lib/libproc/common/llib-lproc
3946 ::gcore
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Matthew Ahrens <mahrens@delphix.com>
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/* LINTLIBRARY */
/* PROTOLIB1 */

/*
 * Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */
/*
 * Copyright (c) 2013 by Delphix. All rights reserved.
 */
#include "libproc.h"

/*
 * usr/src/lib/libproc
 */

/* Pcontrol.c */
int     _libproc_debug;
struct ps_prochandle *Pcreate(const char *file, char *const *argv,
                int *perr, char *path, size_t len);
const char *Pcreate_error(int error);
void    Pcreate_callback(struct ps_prochandle *Pr);
struct ps_prochandle *Pgrab(pid_t pid, int gflag, int *perr);
const char *Pgrab_error(int error);
void    Pfree(struct ps_prochandle *Pr);
int     Pstate(struct ps_prochandle *Pr);
int     Pasfd(struct ps_prochandle *Pr);
int     Pctlfd(struct ps_prochandle *Pr);
const psinfo_t *Ppsinfo(struct ps_prochandle *Pr);
const pstatus_t *Pstatus(struct ps_prochandle *Pr);
int     Pcred(struct ps_prochandle *Pr, prcred_t *pcrp, int ngroups);
int     Ppriv(struct ps_prochandle *Pr, prpriv_t **pprivp);
ssize_t Ppriv(struct ps_prochandle *Pr, prpriv_t *pprivp, size_t);
void    Psync(struct ps_prochandle *Pr);
int     Pcreate_agent(struct ps_prochandle *Pr);
void    Pdestroy_agent(struct ps_prochandle *Pr);
int     Preopen(struct ps_prochandle *Pr);
void    Prelease(struct ps_prochandle *Pr, int flags);
int     Pstopstatus(struct ps_prochandle *Pr, long cmd, uint_t msec);
```



```
int     Pwait(struct ps_prochandle *Pr, uint_t msec);
int     Pstop(struct ps_prochandle *Pr, uint_t msec);
int     Pdstop(struct ps_prochandle *Pr);
int     Pgetareg(struct ps_prochandle *Pr, int regno, prgreg_t *preg);
int     Pputareg(struct ps_prochandle *Pr, int regno, prgreg_t reg);
int     Psetrun(struct ps_prochandle *Pr, int sig, int flags);
ssize_t Pread(struct ps_prochandle *Pr,
                void *buf, size_t nbyte, uintptr_t address);
ssize_t Pread_string(struct ps_prochandle *Pr,
                char *buf, size_t nbyte, uintptr_t address);
ssize_t Pwrite(struct ps_prochandle *Pr,
                const void *buf, size_t nbyte, uintptr_t address);
int     Pclearsig(struct ps_prochandle *Pr);
int     Pclearfault(struct ps_prochandle *Pr);
int     Psetbkpt(struct ps_prochandle *Pr, uintptr_t address, ulong_t *saved);
int     Pdelbkpt(struct ps_prochandle *Pr, uintptr_t address, ulong_t saved);
int     Pxecbkpt(struct ps_prochandle *Pr, ulong_t saved);
int     Psetwapt(struct ps_prochandle *Pr, const prwatch_t *wp);
int     Pdelwapt(struct ps_prochandle *Pr, const prwatch_t *wp);
int     Pxecwapt(struct ps_prochandle *Pr, const prwatch_t *wp);
int     Psetflags(struct ps_prochandle *Pr, long flags);
int     Punsetflags(struct ps_prochandle *Pr, long flags);
int     Psignal(struct ps_prochandle *Pr, int which, int stop);
void    Psetsignal(struct ps_prochandle *Pr, const sigset_t *set);
int     Pfault(struct ps_prochandle *Pr, int which, int stop);
void    Psetfault(struct ps_prochandle *Pr, const fltset_t *set);
int     Psysentry(struct ps_prochandle *Pr, int which, int stop);
void    Psetsysentry(struct ps_prochandle *Pr, const sysset_t *set);
int     Psysexit(struct ps_prochandle *Pr, int which, int stop);
void    Psetsysexit(struct ps_prochandle *Pr, const sysset_t *set);
int     Plwp_iter(struct ps_prochandle *Pr, proc_lwp_f *func, void *cd);
int     Psyscall(struct ps_prochandle *Pr, sysret_t *,
                int sysindex, uint_t nargs, argdes_t *argp);

struct ps_lwphandle *Lgrab(struct ps_prochandle *P, lwpid_t lwpid, int *perr);
const char *Lgrab_error(int error);
struct ps_prochandle *Lprochandle(struct ps_lwphandle *Lwp);
void    Lfree(struct ps_lwphandle *Lwp);
int     Lctlfd(struct ps_lwphandle *Lwp);
int     Lwait(struct ps_lwphandle *Lwp, uint_t msec);
int     Lstop(struct ps_lwphandle *Lwp, uint_t msec);
int     Ldstop(struct ps_lwphandle *Lwp);
int     Lstate(struct ps_lwphandle *Lwp);
const lwpsinfo_t *Lpsinfo(struct ps_lwphandle *Lwp);
const lwpstatus_t *Lstatus(struct ps_lwphandle *Lwp);
int     Lgetareg(struct ps_lwphandle *Lwp, int regno, prgreg_t *preg);
int     Lputareg(struct ps_lwphandle *Lwp, int regno, prgreg_t reg);
int     Lsetrun(struct ps_lwphandle *Lwp, int sig, int flags);
int     Lclearsig(struct ps_lwphandle *Lwp);
int     Lclearfault(struct ps_lwphandle *Lwp);
int     Lxecbkpt(struct ps_lwphandle *Lwp, ulong_t saved);
int     Lxecwapt(struct ps_lwphandle *Lwp, const prwatch_t *wp);
void    Lsync(struct ps_lwphandle *Lwp);

/* Plwpregs.c */
int Plwp_getregs(struct ps_prochandle *Pr, lwpid_t i, prgregset_t gr);
int Plwp_setregs(struct ps_prochandle *Pr, lwpid_t i, const prgregset_t gr);
int Plwp_getfpregs(struct ps_prochandle *Pr, lwpid_t i, prfpregset_t *fp);
int Plwp_setfpregs(struct ps_prochandle *Pr, lwpid_t i, const prfpregset_t *fp);
#if defined(sparc) || defined(__sparc)
int Plwp_getxregs(struct ps_prochandle *Pr, lwpid_t i, prxregset_t *xr);
int Plwp_setxregs(struct ps_prochandle *Pr, lwpid_t i, const prxregset_t *xr);
#if defined(__sparcv9)
int Plwp_getasrs(struct ps_prochandle *Pr, lwpid_t i, asrset_t asrs);
int Plwp_setasrs(struct ps_prochandle *Pr, lwpid_t i, const asrset_t asrs);
#endif  /* __sparcv9 */
```

```
125 #endif  /* __sparc */
126 int Plwp_getpsinfo(struct ps_prochandle *Pr, lwpid_t i, lwpsinfo_t *lps);

128 /* Pcore.c */
129 struct ps_prochandle *Pfgrab_core(int fd, const char *aout, int *perr);
130 struct ps_prochandle *Pgrab_core(const char *core, const char *aout,
131         int gflag, int *perr);

133 /* Pisprocdir.c */
134 int     Pisprocdir(struct ps_prochandle *Pr, const char *dir);

136 /* Pservice.c */
137 ps_err_e ps_pdmodel(struct ps_prochandle *Pr, int *modelp);
138 ps_err_e ps_pread(struct ps_prochandle *Pr,
139                 psaddr_t addr, void *buf, size_t size);
140 ps_err_e ps_pwrite(struct ps_prochandle *Pr,
141                 psaddr_t addr, const void *buf, size_t size);
142 ps_err_e ps_pdread(struct ps_prochandle *Pr,
143                 psaddr_t addr, void *buf, size_t size);
144 ps_err_e ps_pdwrite(struct ps_prochandle *Pr,
145                 psaddr_t addr, const void *buf, size_t size);
146 ps_err_e ps_ptread(struct ps_prochandle *Pr,
147                 psaddr_t addr, void *buf, size_t size);
148 ps_err_e ps_ptwrite(struct ps_prochandle *Pr,
149                 psaddr_t addr, const void *buf, size_t size);
150 ps_err_e ps_pstop(struct ps_prochandle *Pr);
151 ps_err_e ps_pcontinue(struct ps_prochandle *Pr);
152 ps_err_e ps_lstop(struct ps_prochandle *Pr, lwpid_t lwpid);
153 ps_err_e ps_lcontinue(struct ps_prochandle *Pr, lwpid_t lwpid);
154 ps_err_e ps_lgetregs(struct ps_prochandle *Pr,
155                 lwpid_t lwpid, prgregset_t regs);
156 ps_err_e ps_lsetregs(struct ps_prochandle *Pr,
157                 lwpid_t lwpid, const prgregset_t regs);
158 ps_err_e ps_lgetfpregs(struct ps_prochandle *Pr,
159                 lwpid_t lwpid, prfpregset_t *regs);
160 ps_err_e ps_lsetfpregs(struct ps_prochandle *Pr,
161                 lwpid_t lwpid, const prfpregset_t *regs);
162 #if defined(sparc) || defined(__sparc)
163 ps_err_e ps_lgetxregsize(struct ps_prochandle *Pr,
164                 lwpid_t lwpid, int *xrsize);
165 ps_err_e ps_lgetxregs(struct ps_prochandle *Pr,
166                 lwpid_t lwpid, caddr_t xregs);
167 ps_err_e ps_lsetxregs(struct ps_prochandle *Pr,
168                 lwpid_t lwpid, caddr_t xregs);
169 #endif  /* sparc */
170 #if defined(__i386) || defined(__amd64)
171 ps_err_e ps_lgetLDT(struct ps_prochandle *Pr,
172                 lwpid_t lwpid, struct ssd *ldt);
173 #endif  /* __i386 || __amd6464 */
174 void    ps_plog(const char *fmt, ...);

176 /* Psymtab.c */
177 void    Pupdate_maps(struct ps_prochandle *Pr);
178 void    Pupdate_syms(struct ps_prochandle *Pr);
179 rd_agent_t *Prd_agent(struct ps_prochandle *Pr);
180 const prmap_t *Paddr_to_map(struct ps_prochandle *Pr, uintptr_t addr);
181 const prmap_t *Paddr_to_text_map(struct ps_prochandle *Pr, uintptr_t addr);
182 const prmap_t *Pname_to_map(struct ps_prochandle *Pr, const char *name);
183 const prmap_t *Plmid_to_map(struct ps_prochandle *Pr, Lmid_t lmid,
184                 const char *name);
185 int     Plookup_by_addr(struct ps_prochandle *Pr, uintptr_t addr,
186                 char *sym_name_buffer, size_t bufsize, GElf_Sym *symbolp);
187 int     Plookup_by_name(struct ps_prochandle *Pr,
188                 const char *object_name, const char *symbol_name,
189                 GElf_Sym *sym);
190 int     Plookup_by_lmid(struct ps_prochandle *Pr,
```



```
191                 Lmid_t lmid, const char *object_name, const char *symbol_name,
192                 GElf_Sym *sym);
193 const rd_loadobj_t *Paddr_to_loadobj(struct ps_prochandle *, uintptr_t);
194 const rd_loadobj_t *Pname_to_loadobj(struct ps_prochandle *, const char *);
195 const rd_loadobj_t *Plmid_to_loadobj(struct ps_prochandle *, Lmid_t,
196                 const char *);
197 int     Pmapping_iter(struct ps_prochandle *Pr, proc_map_f *func, void *cd);
198 int     Pmapping_iter_resolved(struct ps_prochandle *Pr, proc_map_f *func,
199                 void *cd);
200 int     Pobject_iter(struct ps_prochandle *Pr, proc_map_f *func, void *cd);
201 int     Pobject_iter_resolved(struct ps_prochandle *Pr, proc_map_f *func,
202                 void *cd);
203 char    *Pobjname(struct ps_prochandle *Pr, uintptr_t addr,
204                 char *buffer, size_t bufsize);
205 char    *Pobjname_resolved(struct ps_prochandle *Pr, uintptr_t addr,
206                 char *buffer, size_t bufsize);
207 int     Plmid(struct ps_prochandle *Pr, uintptr_t addr, Lmid_t *lmidp);
208 int     Psymbol_iter(struct ps_prochandle *Pr, const char *object_name,
209                 int which, int type, proc_sym_f *func, void *cd);
210 int     Psymbol_iter_by_lmid(struct ps_prochandle *Pr, Lmid_t lmid,
211                 const char *object_name, int which, int type,
212                 proc_sym_f *func, void *cd);
213 char    *Pgetenv(struct ps_prochandle *Pr, const char *name,
214                 char *buffer, size_t bufsize);
215 char    *Pplatform(struct ps_prochandle *Pr, char *s, size_t n);
216 int     Puname(struct ps_prochandle *Pr, struct utsname *u);
217 char    *Pzonename(struct ps_prochandle *Pr, char *s, size_t n);
218 char    *Pfindobj(struct ps_prochandle *Pr, const char *path,
219                 char *s, size_t n);
220 char    *Pexecname(struct ps_prochandle *Pr, char *buffer, size_t bufsize);
221 void    Preset_maps(struct ps_prochandle *Pr);

223 ps_err_e ps_pglobal_lookup(struct ps_prochandle *Pr,
224                 const char *object_name, const char *sym_name,
225                 psaddr_t *sym_addr);

227 ps_err_e ps_pglobal_sym(struct ps_prochandle *Pr,
228                 const char *object_name, const char *sym_name,
229                 ps_sym_t *symp);

231 long    Pgetauxval(struct ps_prochandle *Pr, int type);
232 const auxv_t *Pgetauxvec(struct ps_prochandle *Pr);
233 ps_err_e ps_pauxv(struct ps_prochandle *Pr, const auxv_t **aux);

235 /* Putil.c */
236 void    Perror_printf(struct ps_prochandle *Pr, const char *format, ...);

238 /* pr_door.c */
239 int     pr_door_info(struct ps_prochandle *Pr, int did, door_info_t *di);

241 /* pr_exit.c */
242 int     pr_exit(struct ps_prochandle *Pr, int status);
243 int     pr_lwp_exit(struct ps_prochandle *Pr);

245 /* pr_fcntl.c */
246 int     pr_fcntl(struct ps_prochandle *Pr, int fd, int cmd, void *argp);

248 /* pr_getitimer.c */
249 int     pr_getitimer(struct ps_prochandle *Pr,
250                 int which, struct itimerval *itv);
251 int     pr_setitimer(struct ps_prochandle *Pr,
252                 int which, const struct itimerval *itv, struct itimerval *oitv);

254 /* pr_getrctl.c */
255 int     pr_getrctl(struct ps_prochandle *Pr, const char *rname,
256                 rctlblk_t *old_blk, rctlblk_t *new_blk, int rflag);
```

```
257 int   pr_setrctl(struct ps_prochandle *Pr, const char *rname,
258                 rctlblk_t *old_blk, rctlblk_t *new_blk, int rflag);
259 int   pr_setprojrctl(struct ps_prochandle *Pr, const char *rname,
260                 rctlblk_t *new_blk, size_t size, int rflag);

262 /* pr_getrlimit.c */
263 int     pr_getrlimit(struct ps_prochandle *Pr,
264                 int resource, struct rlimit *rlp);
265 int     pr_setrlimit(struct ps_prochandle *Pr,
266                 int resource, const struct rlimit *rlp);
267 int     pr_getrlimit64(struct ps_prochandle *Pr,
268                 int resource, struct rlimit64 *rlp);
269 int     pr_setrlimit64(struct ps_prochandle *Pr,
270                 int resource, const struct rlimit64 *rlp);

272 /* pr_getsockname.c */
273 int     pr_getsockname(struct ps_prochandle *Pr,
274                 int sock, struct sockaddr *name, socklen_t *namelen);
275 int     pr_getpeername(struct ps_prochandle *Pr,
276                 int sock, struct sockaddr *name, socklen_t *namelen);

278 /* pr_ioctl.c */
279 int     pr_ioctl(struct ps_prochandle *Pr,
280                 int fd, int code, void *buf, size_t size);

282 /* pr_lseek.c */
283 off_t   pr_lseek(struct ps_prochandle *Pr,
284                 int filedes, off_t offset, int whence);
285 offset_t pr_llseek(struct ps_prochandle *Pr,
286                 int filedes, offset_t offset, int whence);

288 /* pr_memcntl.c */
289 int     pr_memcntl(struct ps_prochandle *Pr,
290         caddr_t addr, size_t len, int cmd, caddr_t arg, int attr, int mask);

292 /* pr_mmap.c */
293 void    *pr_mmap(struct ps_prochandle *Pr,
294                 void *addr, size_t len, int prot, int flags, int fd, off_t off);
295 int     pr_munmap(struct ps_prochandle *Pr,
296                 void *addr, size_t len);
297 void    *pr_zmap(struct ps_prochandle *Pr,
298                 void *addr, size_t len, int prot, int flags);

300 /* pr_open.c */
301 int     pr_open(struct ps_prochandle *Pr,
302                 const char *filename, int flags, mode_t mode);
303 int     pr_creat(struct ps_prochandle *Pr,
304                 const char *filename, mode_t mode);
305 int     pr_close(struct ps_prochandle *Pr, int fd);
306 int     pr_access(struct ps_prochandle *Pr, const char *path, int amode);

308 /* pr_pbind.c */
309 int     pr_processor_bind(struct ps_prochandle *Pr, idtype_t, id_t, int, int *);

311 /* pr_rename.c */
312 int     pr_rename(struct ps_prochandle *Pr, const char *old, const char *new);
313 int     pr_link(struct ps_prochandle *Pr, const char *exist, const char *new);
314 int     pr_unlink(struct ps_prochandle *Pr, const char *);

316 /* pr_sigaction.c */
317 int     pr_sigaction(struct ps_prochandle *Pr,
318                 int sig, const struct sigaction *act, struct sigaction *oact);

320 /* pr_stat.c */
321 int     pr_stat(struct ps_prochandle *Pr, const char *path, struct stat *buf);
322 int     pr_lstat(struct ps_prochandle *Pr, const char *path, struct stat *buf);
```



```
323 int     pr_fstat(struct ps_prochandle *Pr, int fd, struct stat *buf);
324 int     pr_stat64(struct ps_prochandle *Pr, const char *path,
325                 struct stat64 *buf);
326 int     pr_lstat64(struct ps_prochandle *Pr, const char *path,
327                 struct stat64 *buf);
328 int     pr_fstat64(struct ps_prochandle *Pr, int fd, struct stat64 *buf);

330 /* pr_statvfs.c */
331 int     pr_statvfs(struct ps_prochandle *Pr, const char *path, statvfs_t *buf);
332 int     pr_fstatvfs(struct ps_prochandle *Pr, int fd, statvfs_t *buf);

334 /* pr_tasksys.c */
335 projid_t pr_getprojid(struct ps_prochandle *Pr);
336 taskid_t pr_gettaskid(struct ps_prochandle *Pr);
337 taskid_t pr_settaskid(struct ps_prochandle *Pr, projid_t project, int flags);

339 /* pr_waitid.c */
340 int     pr_waitid(struct ps_prochandle *Pr,
341                 idtype_t idtype, id_t id, siginfo_t *infop, int options);

343 /* proc_get_info.c */
344 int     proc_get_cred(pid_t pid, prcred_t *credp, int ngroups);
345 prpriv_t *proc_get_priv(pid_t pid);
346 int     proc_get_psinfo(pid_t pid, psinfo_t *psp);
347 int     proc_get_status(pid_t pid, pstatus_t *psp);
348 int     proc_get_auxv(pid_t pid, auxv_t *pauxv, int naux);

350 /* proc_names.c */
351 char    *proc_fltname(int flt, char *buf, size_t bufsz);
352 char    *proc_signame(int sig, char *buf, size_t bufsz);
353 char    *proc_sysname(int sys, char *buf, size_t bufsz);

355 int     proc_str2flt(const char *str, int *fltnum);
356 int     proc_str2sig(const char *str, int *signum);
357 int     proc_str2sys(const char *str, int *sysnum);

359 char    *proc_fltset2str(const fltset_t *set, const char *delim, int members,
360                 char *buf, size_t nbytes);
361 char    *proc_sigset2str(const sigset_t *set, const char *delim, int members,
362                 char *buf, size_t nbytes);
363 char    *proc_sysset2str(const sysset_t *set, const char *delim, int members,
364                 char *buf, size_t nbytes);

366 char    *proc_str2fltset(const char *str, const char *delim, int members,
367                 fltset_t *set);
368 char    *proc_str2sigset(const char *str, const char *delim, int members,
369                 sigset_t *set);
370 char    *proc_str2sysset(const char *str, const char *delim, int members,
371                 sysset_t *set);

373 int     proc_walk(proc_walk_f *func, void *arg, int flags);

375 /* proc_arg.c */
376 struct ps_prochandle *proc_arg_grab(const char *arg,
377         int oflag, int gflag, int *perr);

379 pid_t   proc_arg_psinfo(const char *arg, int oflag, psinfo_t *psp, int *perr);
380 void    proc_unctrl_psinfo(psinfo_t *psp);

382 /* proc_set.c */
383 int     Psetcred(struct ps_prochandle *Pr, const prcred_t *pcred);

385 /* Pstack.c */
386 int     Pstack_iter(struct ps_prochandle *Pr,
387                 const prgregset_t regs, proc_stack_f *func, void *arg);
```

```
 389 /* Pisadep.c */
 390 const char *Ppltdest(struct ps_prochandle *Pr, uintptr_t addr);
```

```
**********************************************************
    5588 Wed Aug 21 14:46:39 2013
new/usr/src/lib/libproc/common/mapfile-vers
3946 ::gcore
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Matthew Ahrens <mahrens@delphix.com>
**********************************************************
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#
#
# Copyright (c) 2006, 2010, Oracle and/or its affiliates. All rights reserved.
# Copyright 2012 DEY Storage Systems, Inc.  All rights reserved.
# Copyright (c) 2013, Joyent, Inc. All rights reserved.
# Copyright (c) 2013 by Delphix. All rights reserved.
#

#
# MAPFILE HEADER START
#
# WARNING:  STOP NOW.  DO NOT MODIFY THIS FILE.
# Object versioning must comply with the rules detailed in
#
#       usr/src/lib/README.mapfiles
#
# You should not be making modifications here until you've read the most current
# copy of that file. If you need help, contact a gatekeeper for guidance.
#
# MAPFILE HEADER END
#

$mapfile_version 2

# Due to mistakes made early in the history of this library, there are no
# SUNW_1.1 through SUNW_1.4 symbols, but they are now kept as placeholders.
# Don't add any symbols to these versions.

SYMBOL_VERSION SUNW_1.4 {
    global:
        SUNW_1.4;
} SUNW_1.3;
______unchanged_portion_omitted_

SYMBOL_VERSION SUNWprivate_1.1 {
    global:
        Lalt_stack;
        Lclearfault;
        Lclearsig;
        Lctlfd;
```



```
        Ldstop;
        Lfree;
        Lgetareg;
        Lgrab;
        Lgrab_error;
        _libproc_debug;
        Lmain_stack;
        Lprochandle;
        Lpsinfo;
        Lputareg;
        Lsetrun;
        Lstack;
        Lstate;
        Lstatus;
        Lstop;
        Lsync;
        Lwait;
        Lxecbkpt;
        Lxecwapt;
        Paddr_to_ctf;
        Paddr_to_loadobj;
        Paddr_to_map;
        Paddr_to_text_map;
        Pasfd;
        Pclearfault;
        Pclearsig;
        Pcontent;
        Pcreate;
        Pcreate_agent;
        Pcreate_callback;
        Pcreate_error;
        Pcred;
        Pctlfd;
        Pdelbkpt;
        Pdelwapt;
        Pdestroy_agent;
        Pdstop;
        Penv_iter;
        Perror_printf;
        Pexecname;
        Pfault;
        Pfgcore;
        Pfgrab_core;
        Pfree;
        Pgcore;
        Pgetareg;
        Pgetauxval;
        Pgetauxvec;
        Pgetenv;
        Pgrab;
        Pgrab_core;
        Pgrab_error;
        Pgrab_file;
        Pgrab_ops;
        Pisprocdir;
        Pissyscall_prev;
        Plmid;
        Plmid_to_loadobj;
        Plmid_to_map;
        Plookup_by_addr;
        Plookup_by_name;
        Plwp_alt_stack;
        Plwp_getfpregs;
        Plwp_getpsinfo;
        Plwp_getregs;
        Plwp_getspymaster;
```

```
140         Plwp_iter;
141         Plwp_iter_all;
142         Plwp_main_stack;
143         Plwp_setfpregs;
144         Plwp_setregs;
145         Plwp_stack;
146         Pmapping_iter;
147         Pmapping_iter_resolved;
148         Pname_to_ctf;
149         Pname_to_loadobj;
150         Pname_to_map;
151         Pobject_iter;
152         Pobject_iter_resolved;
153         Pobjname;
154         Pobjname_resolved;
155         Pplatform;
156         Ppltdest;
157         Ppriv;
158         Pprivinfo;
159         Ppsinfo;
160         Pputareg;
161         pr_access;
162         pr_close;
163         pr_creat;
164         Prd_agent;
165         pr_door_info;
166         Pread;
167         Pread_string;
168         Prelease;
169         Preopen;
170         Preset_maps;
171         pr_exit;
172         pr_fcntl;
173         pr_fstat;
174         pr_fstat64;
175         pr_fstatvfs;
176         pr_getitimer;
177         pr_getpeername;
178         pr_getpeerucred;
179         pr_getprojid;
180         pr_getrctl;
181         pr_getrlimit;
182         pr_getrlimit64;
183         pr_getsockname;
184         pr_getsockopt;
185         pr_gettaskid;
186         pr_getzoneid;
187         pr_ioctl;
188         pr_link;
189         pr_llseek;
190         pr_lseek;
191         pr_lstat;
192         pr_lstat64;
193         pr_lwp_exit;
194         pr_memcntl;
195         pr_meminfo;
196         pr_mmap;
197         pr_munmap;
198         proc_arg_grab;
199         proc_arg_psinfo;
200         proc_arg_xgrab;
201         proc_arg_xpsinfo;
202         proc_content2str;
203         proc_finistdio;
204         proc_fltname;
205         proc_fltset2str;
```



```
206         proc_flushstdio;
207         proc_get_auxv;
208         proc_get_cred;
209         proc_get_priv;
210         proc_get_psinfo;
211         proc_get_status;
212         proc_initstdio;
213         proc_lwp_in_set;
214         proc_lwp_range_valid;
215         proc_signame;
216         proc_sigset2str;
217         proc_str2content;
218         proc_str2flt;
219         proc_str2fltset;
220         proc_str2sig;
221         proc_str2sigset;
222         proc_str2sys;
223         proc_str2sysset;
224         proc_sysname;
225         proc_sysset2str;
226         proc_unctrl_psinfo;
227         proc_walk;
228         pr_open;
229         pr_processor_bind;
230         pr_rename;
231         pr_setitimer;
232         pr_setprojrctl;
233         pr_setrctl;
234         pr_setrlimit;
235         pr_setrlimit64;
236         pr_settaskid;
237         pr_sigaction;
238         pr_stat;
239         pr_stat64;
240         pr_statvfs;
241         pr_unlink;
242         pr_waitid;
243         pr_zmap;
244         Pset_procfs_path;
245         Psetbkpt;
246         Psetcred;
247         Psetfault;
248         Psetflags;
249         Psetpriv;
250         Psetrun;
251         Psetsignal;
252         Psetsysentry;
253         Psetsysexit;
254         Psetwapt;
255         Psetzoneid;
256         Psignal;
257         ps_lcontinue;
258         ps_lgetfpregs;
259         ps_lgetregs;
260         ps_lsetfpregs;
261         ps_lsetregs;
262         ps_lstop;
263         ps_pauxv;
264         ps_pbrandname;
265         ps_pcontinue;
266         ps_pdmodel;
267         ps_pdread               { FLAGS = NODYNSORT };  # Alias of ps_pread
268         ps_pdwrite              { FLAGS = NODYNSORT };  # Alias of ps_pwrite
269         ps_pglobal_lookup;
270         ps_pglobal_sym;
271         ps_plog;
```

```
272         ps_pread;
273         ps_pstop;
274         ps_ptread               { FLAGS = NODYNSORT };  # Alias of ps_pread
275         ps_ptwrite              { FLAGS = NODYNSORT };  # Alias of ps_pwrite
276         ps_pwrite;
277         Pstack_iter;
278         Pstate;
279         Pstatus;
280         Pstop;
281         Pstopstatus;
282         Psymbol_iter;
283         Psymbol_iter_by_addr;
284         Psymbol_iter_by_lmid;
285         Psymbol_iter_by_name;
286         Psync;
287         Psyscall;
288         Psysentry;
289         Psysexit;
290         Puname;
291         Punsetflags;
292         Pupdate_maps;
293         Pupdate_syms;
294         Pwait;
295         Pwrite;
296         Pxcreate;
297         Pxecbkpt;
298         Pxecwapt;
299         Pxlookup_by_addr;
300         Pxlookup_by_addr_resolved;
301         Pxlookup_by_name;
302         Pxsymbol_iter;
303         Pzonename;
304         Pzonepath;
305         Pzoneroot;
306         Pfdinfo_iter;

308 $if _x86 && _ELF32
309         Pldt;
310         proc_get_ldt;
311         ps_lgetLDT;
312 $endif

314 $if _sparc
315         Plwp_getgwindows;
316         Plwp_getxregs;
317         Plwp_setxregs;
318         ps_lgetxregs;
319         ps_lgetxregsize;
320         ps_lsetxregs;

322 $if _ELF64
323         Plwp_getasrs;
324         Plwp_setasrs;
325 $endif
326 $endif

328     local:
329         *;
330 };
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    1979 Wed Aug 21 14:46:40 2013
new/usr/src/uts/common/sys/lwp.h
3946 ::gcore
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Matthew Ahrens <mahrens@delphix.com>
**********************************************************
```

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

/*
 * Copyright 2008 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */
/*
 * Copyright (c) 2013 by Delphix. All rights reserved.
 */

#ifndef _SYS_LWP_H
#define _SYS_LWP_H

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

#include <sys/synch.h>
#include <sys/ucontext.h>

#ifdef  __cplusplus
extern "C" {
#endif

/*
 * lwp create flags
 */
#define LWP_DAEMON      0x00000020
#define LWP_DETACHED    0x00000040
#define LWP_SUSPENDED   0x00000080

/*
 * Definitions for user programs calling into the _lwp interface.
 */
struct lwpinfo {
        timestruc_t lwp_utime;
        timestruc_t lwp_stime;
        long        lwpinfo_pad[64];
};
_____unchanged_portion_omitted_

#endif  /* _SYSCALL32 */
```



```
typedef uint_t lwpid_t;

#define _LWP_FSBASE     0
#define _LWP_GSBASE     1

#define _LWP_SETPRIVATE 0
#define _LWP_GETPRIVATE 1

#ifndef _KERNEL

typedef uint_t lwpid_t;

int             _lwp_kill(lwpid_t, int);
int             _lwp_info(struct lwpinfo *);
lwpid_t         _lwp_self(void);
int             _lwp_suspend(lwpid_t);
int             _lwp_continue(lwpid_t);

#endif  /* _KERNEL */

#ifdef  __cplusplus
}
_____unchanged_portion_omitted_
```

```
**********************************************************
   13585 Wed Aug 21 14:46:42 2013
new/usr/src/uts/common/sys/syscall.h
3946 ::gcore
Reviewed by: Adam Leventhal <ahl@delphix.com>
Reviewed by: Matthew Ahrens <mahrens@delphix.com>
**********************************************************
```

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

/*
 * Copyright (c) 1989, 2010, Oracle and/or its affiliates. All rights reserved.
 * Copyright (c) 2013 by Delphix. All rights reserved.
 */

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

#ifndef _SYS_SYSCALL_H
#define _SYS_SYSCALL_H

#ifdef  __cplusplus
extern "C" {
#endif

/*
 *      system call numbers
 *              syscall(SYS_xxxx, ...)
 */

        /* syscall enumeration MUST begin with 1 */

        /*
         * SunOS/SPARC uses 0 for the indirect system call SYS_syscall
         * but this doesn't count because it is just another way
         * to specify the real system call number.
         */

#define SYS_syscall     0
#define SYS_exit        1
#define SYS_read        3
#define SYS_write       4
#define SYS_open        5
#define SYS_close       6
#define SYS_linkat      7
#define SYS_link        9
#define SYS_unlink      10
#define SYS_symlinkat   11
```



```
#define SYS_chdir       12
#define SYS_time        13
#define SYS_mknod       14
#define SYS_chmod       15
#define SYS_chown       16
#define SYS_brk         17
#define SYS_stat        18
#define SYS_lseek       19
#define SYS_getpid      20
#define SYS_mount       21
#define SYS_readlinkat  22
#define SYS_setuid      23
#define SYS_getuid      24
#define SYS_stime       25
#define SYS_pcsample    26
#define SYS_alarm       27
#define SYS_fstat       28
#define SYS_pause       29
#define SYS_stty        31
#define SYS_gtty        32
#define SYS_access      33
#define SYS_nice        34
#define SYS_statfs      35
#define SYS_sync        36
#define SYS_kill        37
#define SYS_fstatfs     38
#define SYS_pgrpsys     39
        /*
         * subcodes:
         *      getpgrp()         :: syscall(39,0)
         *      setpgrp()         :: syscall(39,1)
         *      getsid(pid)       :: syscall(39,2,pid)
         *      setsid()          :: syscall(39,3)
         *      getpgid(pid)      :: syscall(39,4,pid)
         *      setpgid(pid,pgid) :: syscall(39,5,pid,pgid)
         */
#define SYS_uucopystr   40
#define SYS_pipe        42
#define SYS_times       43
#define SYS_profil      44
#define SYS_faccessat   45
#define SYS_setgid      46
#define SYS_getgid      47
#define SYS_mknodat     48
#define SYS_msgsys      49
        /*
         * subcodes:
         *      msgget(...)  :: msgsys(0, ...)
         *      msgctl(...)  :: msgsys(1, ...)
         *      msgrcv(...)  :: msgsys(2, ...)
         *      msgsnd(...)  :: msgsys(3, ...)
         *      msgids(...)  :: msgsys(4, ...)
         *      msgsnap(...) :: msgsys(5, ...)
         *      see <sys/msg.h>
         */
#define SYS_sysi86      50
        /*
         * subcodes:
         *      sysi86(code, ...)
         */
#define SYS_acct        51
#define SYS_shmsys      52
        /*
         * subcodes:
         *      shmat (...) :: shmsys(0, ...)
         *      shmctl(...) :: shmsys(1, ...)
```

```
126          *      shmdt (...) :: shmsys(2, ...)
127          *      shmget(...) :: shmsys(3, ...)
128          *      shmids(...) :: shmsys(4, ...)
129          *      see <sys/shm.h>
130          */
131 #define SYS_semsys      53
132         /*
133          * subcodes:
134          *      semctl(...) :: semsys(0, ...)
135          *      semget(...) :: semsys(1, ...)
136          *      semop (...) :: semsys(2, ...)
137          *      semids(...) :: semsys(3, ...)
138          *      semtimedop(...) :: semsys(4, ...)
139          *      see <sys/sem.h>
140          */
141 #define SYS_ioctl       54
142 #define SYS_uadmin      55
143 #define SYS_fchownat    56
144 #define SYS_utssys      57
145         /*
146          * subcodes (third argument):
147          *      uname(obuf)  (obsolete)   :: syscall(57, obuf, ign, 0)
148          *                                        subcode 1 unused
149          *      ustat(dev, obuf)          :: syscall(57, obuf, dev, 2)
150          *      fusers(path, flags, obuf) :: syscall(57, path, flags, 3, obuf)
151          *      see <sys/utssys.h>
152          */
153 #define SYS_fdsync      58
154 #define SYS_execve      59
155 #define SYS_umask       60
156 #define SYS_chroot      61
157 #define SYS_fcntl       62
158 #define SYS_ulimit      63
159 #define SYS_renameat    64
160 #define SYS_unlinkat    65
161 #define SYS_fstatat     66
162 #define SYS_fstatat64   67
163 #define SYS_openat      68
164 #define SYS_openat64    69
165 #define SYS_tasksys     70
166         /*
167          * subcodes:
168          *      settaskid(...) :: tasksys(0, ...)
169          *      gettaskid(...) :: tasksys(1, ...)
170          *      getprojid(...) :: tasksys(2, ...)
171          */
172 #define SYS_acctctl     71
173 #define SYS_exacctsys   72
174         /*
175          * subcodes:
176          *      getacct(...) :: exacct(0, ...)
177          *      putacct(...) :: exacct(1, ...)
178          *      wracct(...) :: exacct(2, ...)
179          */
180 #define SYS_getpagesizes        73
181         /*
182          * subcodes:
183          *      getpagesizes2(...)      :: getpagesizes(0, ...)
184          *      getpagesizes(...)       :: getpagesizes(1, ...) legacy
185          */
186 #define SYS_rctlsys     74
187         /*
188          * subcodes:
189          *      getrctl(...) :: rctlsys(0, ...)
190          *      setrctl(...) :: rctlsys(1, ...)
191          *      rctllist(...) :: rctlsys(2, ...)
```



```
192          *      rctlctl(...) :: rctlsys(3, ...)
193          */
194 #define SYS_sidsys      75
195         /*
196          * subcodes:
197          *      allocids(...)           :: sidsys(0, ...)
198          *      idmap_reg(...)          :: sidsys(1, ...)
199          *      idmap_unreg(...)        :: sidsys(2, ...)
200          */
201 #define SYS_lwp_park    77
202         /*
203          * subcodes:
204          *      _lwp_park(timespec_t *, lwpid_t)        :: syslwp_park(0, ...)
205          *      _lwp_unpark(lwpid_t, int)               :: syslwp_park(1, ...)
206          *      _lwp_unpark_all(lwpid_t *, int)         :: syslwp_park(2, ...)
207          *      _lwp_unpark_cancel(lwpid_t *, int)      :: syslwp_park(3, ...)
208          *      _lwp_set_park(lwpid_t *, int)           :: syslwp_park(4, ...)
209          */
210 #define SYS_sendfilev   78
211         /*
212          * subcodes :
213          *      sendfilev()   :: sendfilev(0, ...)
214          *      sendfilev64() :: sendfilev(1, ...)
215          */
216 #define SYS_rmdir       79
217 #define SYS_mkdir       80
218 #define SYS_getdents    81
219 #define SYS_privsys     82
220         /*
221          * subcodes:
222          *      setppriv(...)           :: privsys(0, ...)
223          *      getppriv(...)           :: privsys(1, ...)
224          *      getimplinfo(...)        :: privsys(2, ...)
225          *      setpflags(...)          :: privsys(3, ...)
226          *      getpflags(...)          :: privsys(4, ...)
227          *      issetugid();            :: privsys(5)
228          */
229 #define SYS_ucredsys    83
230         /*
231          * subcodes:
232          *      ucred_get(...)          :: ucredsys(0, ...)
233          *      getpeerucred(...)       :: ucredsys(1, ...)
234          */
235 #define SYS_sysfs       84
236         /*
237          * subcodes:
238          *      sysfs(code, ...)
239          *      see <sys/fstyp.h>
240          */
241 #define SYS_getmsg      85
242 #define SYS_putmsg      86
243 #define SYS_lstat       88
244 #define SYS_symlink     89
245 #define SYS_readlink    90
246 #define SYS_setgroups   91
247 #define SYS_getgroups   92
248 #define SYS_fchmod      93
249 #define SYS_fchown      94
250 #define SYS_sigprocmask 95
251 #define SYS_sigsuspend  96
252 #define SYS_sigaltstack 97
253 #define SYS_sigaction   98
254 #define SYS_sigpending  99
255         /*
256          * subcodes:
257          *                       subcode 0 unused
```

```
 258          *      sigpending(...) :: syscall(99, 1, ...)
 259          *      sigfillset(...) :: syscall(99, 2, ...)
 260          */
 261 #define SYS_context     100
 262         /*
 263          * subcodes:
 264          *      getcontext(...) :: syscall(100, 0, ...)
 265          *      setcontext(...) :: syscall(100, 1, ...)
 266          */
 267 #define SYS_fchmodat    101
 268 #define SYS_mkdirat     102
 269 #define SYS_statvfs     103
 270 #define SYS_fstatvfs    104
 271 #define SYS_getloadavg  105
 272 #define SYS_nfssys      106
 273 #define SYS_waitid      107
 274 #define SYS_waitsys     SYS_waitid      /* historical */
 275 #define SYS_sigsendsys  108
 276 #define SYS_hrtsys      109
 277 #define SYS_utimesys    110
 278         /*
 279          * subcodes:
 280          *      futimens(...)   :: syscall(110, 0, ...)
 281          *      utimensat(...)  :: syscall(110, 1, ...)
 282          */
 283 #define SYS_sigresend   111
 284 #define SYS_priocntlsys 112
 285 #define SYS_pathconf    113
 286 #define SYS_mincore     114
 287 #define SYS_mmap        115
 288 #define SYS_mprotect    116
 289 #define SYS_munmap      117
 290 #define SYS_fpathconf   118
 291 #define SYS_vfork       119
 292 #define SYS_fchdir      120
 293 #define SYS_readv       121
 294 #define SYS_writev      122
 295 #define SYS_mmapobj     127
 296 #define SYS_setrlimit   128
 297 #define SYS_getrlimit   129
 298 #define SYS_lchown      130
 299 #define SYS_memcntl     131
 300 #define SYS_getpmsg     132
 301 #define SYS_putpmsg     133
 302 #define SYS_rename      134
 303 #define SYS_uname       135
 304 #define SYS_setegid     136
 305 #define SYS_sysconfig   137
 306 #define SYS_adjtime     138
 307 #define SYS_systeminfo  139
 308 #define SYS_sharefs     140
 309 #define SYS_seteuid     141
 310 #define SYS_forksys     142
 311         /*
 312          * subcodes:
 313          *      forkx(flags)    :: forksys(0, flags)
 314          *      forkallx(flags) :: forksys(1, flags)
 315          *      vforkx(flags)   :: forksys(2, flags)
 316          */
 317 #define SYS_sigtimedwait        144
 318 #define SYS_lwp_info    145
 319 #define SYS_yield       146
 320 #define SYS_lwp_sema_post       148
 321 #define SYS_lwp_sema_trywait    149
 322 #define SYS_lwp_detach  150
 323 #define SYS_corectl     151
```



```
 324 #define SYS_modctl      152
 325 #define SYS_fchroot     153
 326 #define SYS_vhangup     155
 327 #define SYS_gettimeofday        156
 328 #define SYS_getitimer           157
 329 #define SYS_setitimer           158
 330 #define SYS_lwp_create          159
 331 #define SYS_lwp_exit            160
 332 #define SYS_lwp_suspend         161
 333 #define SYS_lwp_continue        162
 334 #define SYS_lwp_kill            163
 335 #define SYS_lwp_self            164
 336 #define SYS_lwp_sigmask         165
 337 #define SYS_lwp_private         166
 338 #define SYS_lwp_wait            167
 339 #define SYS_lwp_mutex_wakeup    168
 340 #define SYS_lwp_cond_wait       170
 341 #define SYS_lwp_cond_signal     171
 342 #define SYS_lwp_cond_broadcast  172
 343 #define SYS_pread               173
 344 #define SYS_pwrite              174
 345 #define SYS_llseek              175
 346 #define SYS_inst_sync           176
 347 #define SYS_brand               177
 348 #define SYS_kaio                178
 349         /*
 350          * subcodes:
 351          *      aioread(...)    :: kaio(AIOREAD, ...)
 352          *      aiowrite(...)   :: kaio(AIOWRITE, ...)
 353          *      aiowait(...)    :: kaio(AIOWAIT, ...)
 354          *      aiocancel(...)  :: kaio(AIOCANCEL, ...)
 355          *      aionotify()     :: kaio(AIONOTIFY)
 356          *      aioinit()       :: kaio(AIOINIT)
 357          *      aiostart()      :: kaio(AIOSTART)
 358          *      see <sys/aio.h>
 359          */
 360 #define SYS_cpc                 179
 361 #define SYS_lgrpsys             180
 362 #define SYS_meminfosys          SYS_lgrpsys
 363         /*
 364          * subcodes:
 365          *      meminfo(...) :: meminfosys(MISYS_MEMINFO, ...)
 366          */
 367 #define SYS_rusagesys           181
 368         /*
 369          * subcodes:
 370          *      getrusage(...) :: rusagesys(RUSAGESYS_GETRUSAGE, ...)
 371          *      getvmusage(...)    :: rusagesys(RUSAGESYS_GETVMUSAGE, ...)
 372          */
 373 #define SYS_port                182
 374         /*
 375          * subcodes:
 376          *      port_create(...) :: portfs(PORT_CREATE, ...)
 377          *      port_associate(...) :: portfs(PORT_ASSOCIATE, ...)
 378          *      port_dissociate(...) :: portfs(PORT_DISSOCIATE, ...)
 379          *      port_send(...) :: portfs(PORT_SEND, ...)
 380          *      port_sendn(...) :: portfs(PORT_SENDN, ...)
 381          *      port_get(...) :: portfs(PORT_GET, ...)
 382          *      port_getn(...) :: portfs(PORT_GETN, ...)
 383          *      port_alert(...) :: portfs(PORT_ALERT, ...)
 384          *      port_dispatch(...) :: portfs(PORT_DISPATCH, ...)
 385          */
 386 #define SYS_pollsys             183
 387 #define SYS_labelsys            184
 388 #define SYS_acl                 185
 389 #define SYS_auditsys            186
```

```
390 #define SYS_processor_bind      187
391 #define SYS_processor_info      188
392 #define SYS_p_online            189
393 #define SYS_sigqueue            190
394 #define SYS_clock_gettime       191
395 #define SYS_clock_settime       192
396 #define SYS_clock_getres        193
397 #define SYS_timer_create        194
398 #define SYS_timer_delete        195
399 #define SYS_timer_settime       196
400 #define SYS_timer_gettime       197
401 #define SYS_timer_getoverrun    198
402 #define SYS_nanosleep           199
403 #define SYS_facl                200
404 #define SYS_door                201
405         /*
406          * Door Subcodes:
407          *      0       door_create
408          *      1       door_revoke
409          *      2       door_info
410          *      3       door_call
411          *      4       door_return
412          */
413 #define SYS_setreuid            202
414 #define SYS_setregid            203
415 #define SYS_install_utrap       204
416 #define SYS_signotify           205
417 #define SYS_schedctl            206
418 #define SYS_pset                207
419 #define SYS_sparc_utrap_install 208
420 #define SYS_resolvepath         209
421 #define SYS_lwp_mutex_timedlock 210
422 #define SYS_lwp_sema_timedwait  211
423 #define SYS_lwp_rwlock_sys      212
424         /*
425          * subcodes:
426          *      lwp_rwlock_rdlock(...)    :: syscall(212, 0, ...)
427          *      lwp_rwlock_wrlock(...)    :: syscall(212, 1, ...)
428          *      lwp_rwlock_tryrdlock(...) :: syscall(212, 2, ...)
429          *      lwp_rwlock_trywrlock(...) :: syscall(212, 3, ...)
430          *      lwp_rwlock_unlock(...)    :: syscall(212, 4, ...)
431          */
432 /* system calls for large file ( > 2 gigabyte) support */
433 #define SYS_getdents64          213
434 #define SYS_mmap64              214
435 #define SYS_stat64              215
436 #define SYS_lstat64             216
437 #define SYS_fstat64             217
438 #define SYS_statvfs64           218
439 #define SYS_fstatvfs64          219
440 #define SYS_setrlimit64         220
441 #define SYS_getrlimit64         221
442 #define SYS_pread64             222
443 #define SYS_pwrite64            223
444 #define SYS_open64              225
445 #define SYS_rpcsys              226
446 #define SYS_zone                227
447         /*
448          * subcodes:
449          *      zone_create(...) :: zone(ZONE_CREATE, ...)
450          *      zone_destroy(...) :: zone(ZONE_DESTROY, ...)
451          *      zone_getattr(...) :: zone(ZONE_GETATTR, ...)
452          *      zone_enter(...) :: zone(ZONE_ENTER, ...)
453          *      zone_list(...) :: zone(ZONE_LIST, ...)
454          *      zone_shutdown(...) :: zone(ZONE_SHUTDOWN, ...)
455          *      zone_lookup(...) :: zone(ZONE_LOOKUP, ...)
```



```
456          *      zone_boot(...) :: zone(ZONE_BOOT, ...)
457          *      zone_version(...) :: zone(ZONE_VERSION, ...)
458          *      zone_setattr(...) :: zone(ZONE_SETATTR, ...)
459          *      zone_add_datalink(...) :: zone(ZONE_ADD_DATALINK, ...)
460          *      zone_remove_datalink(...) :: zone(ZONE_DEL_DATALINK, ...)
461          *      zone_check_datalink(...) :: zone(ZONE_CHECK_DATALINK, ...)
462          *      zone_list_datalink(...) :: zone(ZONE_LIST_DATALINK, ...)
463          */
464 #define SYS_autofssys           228
465 #define SYS_getcwd              229
466 #define SYS_so_socket           230
467 #define SYS_so_socketpair       231
468 #define SYS_bind                232
469 #define SYS_listen              233
470 #define SYS_accept              234
471 #define SYS_connect             235
472 #define SYS_shutdown            236
473 #define SYS_recv                237
474 #define SYS_recvfrom            238
475 #define SYS_recvmsg             239
476 #define SYS_send                240
477 #define SYS_sendmsg             241
478 #define SYS_sendto              242
479 #define SYS_getpeername         243
480 #define SYS_getsockname         244
481 #define SYS_getsockopt          245
482 #define SYS_setsockopt          246
483 #define SYS_sockconfig          247
484         /*
485          * NTP codes
486          */
487 #define SYS_ntp_gettime         248
488 #define SYS_ntp_adjtime         249
489 #define SYS_lwp_mutex_unlock    250
490 #define SYS_lwp_mutex_trylock   251
491 #define SYS_lwp_mutex_register  252
492 #define SYS_cladm               253
493 #define SYS_uucopy              254
494 #define SYS_umount2             255

496 #ifndef _ASM

498 typedef struct {                /* syscall set type */
499         unsigned int    word[16];
500 } sysset_t;

501 #if !defined(_KERNEL)

502 typedef struct {        /* return values from system call */
503         long    sys_rval1;      /* primary return value from system call */
504         long    sys_rval2;      /* second return value from system call */
505 } sysret_t;

507 #if !defined(_KERNEL)

509 #if defined(__STDC__)
510 extern int      syscall(int, ...);
511 extern int      __systemcall(sysret_t *, int, ...);
512 extern int      __set_errno(int);
513 #else
514 extern int      syscall();
515 extern int      __systemcall();
516 extern int      __set_errno();
517 #endif

519 #endif  /* _KERNEL */
```

```
521 #endif  /* _ASM */

523 #ifdef  __cplusplus
524 }
```

unchanged_portion_omitted_